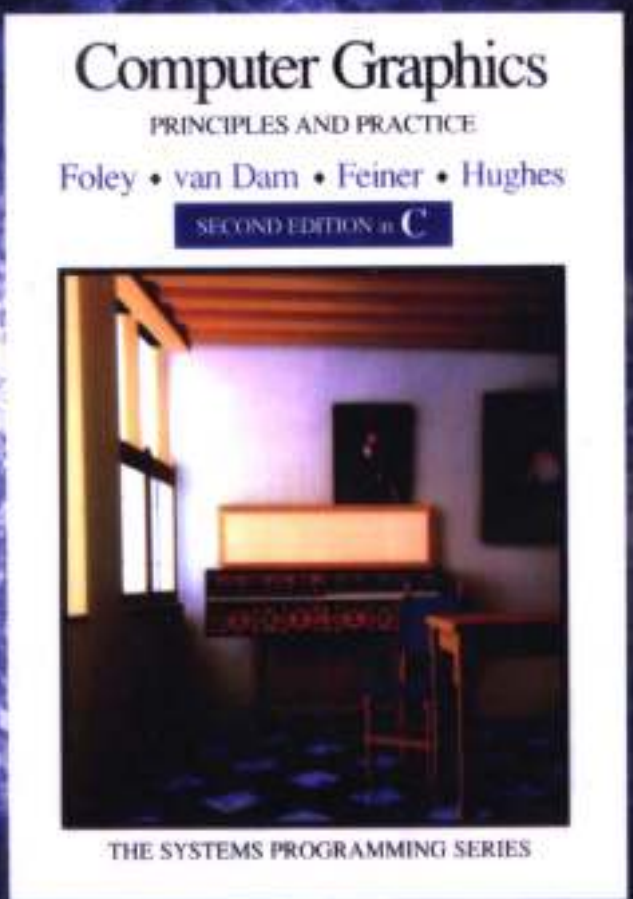
计 算 机 科 学 丛 书

原书第2版

计算机图形学原理及实践
C语言描述

(美) James D. Foley Andries van Dam Steven K. Feiner John F. Hughes 著 唐泽圣 董上海 李华 吴恩华 汪国平 等译



Computer Graphics
Principles and Practice
Second Edition in C

机械工业出版社
China Machine Press

这是计算机图形学领域的一部经典之作，作者Foley、van Dam等是国际图形学界的著名学者、学术带头人，而且本书英文版自出版以来，一直是各国大学计算机图形学课程的主要教科书。来自清华大学、北京大学、中国科学院计算技术研究所、中国科学院软件研究所的多位图形学领域的专家和精英花费了大量的时间和精力进行翻译，最终完成了这本中文版。

本书由基础知识、用户界面、模型定义和图像合成四个部分组成，内容覆盖了日趋成熟的计算机图形学领域各个方面，包括二维、三维图形学的数学基础，重要算法，光栅图形硬件和软件，交互技术及用户界面软件，真实感图形学，高级建模技术（分形、体绘制等），图像处理和存储，以及动画等。此外，书中包括了大量习题和参考文献，提供了大量的用C语言编写的实现算法的程序。

本书是高等院校计算机专业本科生、研究生计算机图形学课程的理想教材，是相关领域专业人员开展研究工作的优秀参考书。

作者简介

James D. Foley 于密歇根大学获得博士学位，是佐治亚理工学院教授，图形学、可视化及可用性研究中心创始人，现任该中心主任。他还是ACM、ACM SIGGRAPH、ACM SIGCHI和IEEE的成员。



Andries van Dam 于宾夕法尼亚大学获得博士学位，是布朗大学计算机科学系创始人之一，而且是该系的首任系主任，现为该系教授。他是IEEE计算机学会及ACM的成员，ACM SIGGRAPH的创始人之一。



Steven K. Feiner 于布朗大学获得博士学位，是哥伦比亚大学计算机科学系教授，计算机图形学与用户界面实验室负责人。他也是ACM SIGGRAPH和IEEE计算机学会的成员。



John F. Hughes 于加利福尼亚大学伯克利分校获得博士学位，现任布朗大学计算机科学系副教授。他也是ACM SIGGRAPH和IEEE计算机学会的成员。



ISBN 7-111-13026-X

9 787111 130260

华章图书

网上购书：www.china-pub.com

北京市西城区百万庄南街1号 100037

读者服务热线：(010)68995259，68995264

读者服务信箱：hzedu@hzbook.com

http://www.hzbook.com

ISBN 7-111-13026-X/TP · 2918

定价：95.00 元

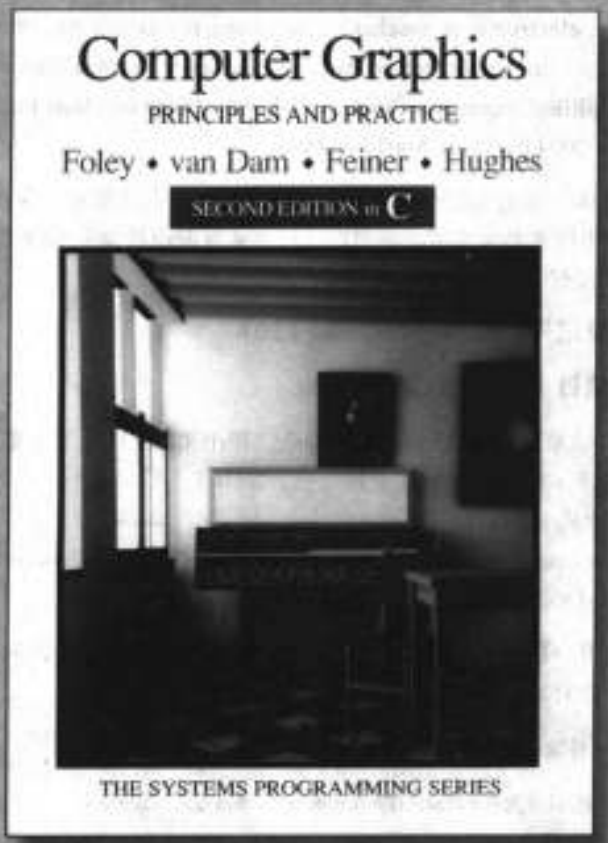
计 算 机 科 学 丛 书

原书第2版

计算机图形学原理及实践

C语言描述

(美) James D. Foley Andries van Dam Steven K. Feiner John F. Hughes 著 唐泽圣 董士海 李华 吴恩华 汪国平 等译



Computer Graphics
Principles and Practice
Second Edition in C

机械工业出版社
China Machine Press

本书是计算机图形学领域的经典著作。本书由基础知识、用户界面、模型定义和图像合成四个部分组成，包括SRGP的编程、画二维图元的基本光栅图形学算法、图形硬件、几何变换、三维空间的观察、对象的层次结构和SPHIGS系统、输入设备、交互技术与交互任务、对话设计、用户界面软件、实体造型、消色光与彩色光、可视图像的真实性、可见面判定、光照模型与光照计算、图像处理与存储、高级光栅图形体系结构、高级几何与光栅算法、高级建模技术和动画等内容。

本书内容全面，涉及图形学的各个领域，可以作为计算机专业本科生和研究生的教材，同时也可供相关技术人员阅读。

Authorized translation from the English language edition entitled *Computer Graphics: Principles and Practice, Second Edition in C* (ISBN: 0-201-84840-6) by James D. Foley et al., published by Pearson Education, Inc, publishing as Addison-Wesley, Copyright © 1996 by Bell Telephone Laboratories, Incorporated.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanic, including photocopying, recording, or by any information storage retrieval system, without permission of Pearson Education, Inc.

Chinese simplified language edition published by China Machine Press.

Copyright © 2004 by China Machine Press.

本书中文简体字版由美国Pearson Education培生教育出版集团授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

本书版权登记号：图字：01-2000-1161

图书在版编目（CIP）数据

计算机图形学原理及实践——C语言描述（原书第2版）/（美）福利（Foley, J. D.）等著；唐泽圣等译. - 北京：机械工业出版社，2004.3

（计算机科学丛书）

书名原文：Computer Graphics: Principles and Practice, Second Edition in C

ISBN 7-111-13026-X

Ⅰ. 计… Ⅱ. ① 福… ② 唐… Ⅲ. ① 计算机图形学 ② C语言-程序设计 Ⅳ. ① TP391.41 ② TP312

中国版本图书馆CIP数据核字（2003）第080765号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：杨海玲

北京中加印刷有限公司印刷 · 新华书店北京发行所发行

2004年3月第1版第1次印刷

787mm × 1092mm 1/16 · 59.25印张（彩插4.25印张）

印数：0 001-5 000册

定价：95.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294



彩图A "棕色熊"，由J.Kajiya、T.Kay和J.Snyder提供。（在加利福尼亚理工学院和IBM公司生产，加利福尼亚理工学院版权所有，1989。）

彩图B "祝福"，由F.K.Musgrave提供。（K.Musgrave和B.Mandelbrot版权所有，1989。）





彩图C "体素花园"。(经NYIT计算机图形学实验室的Ned Greene许可。)



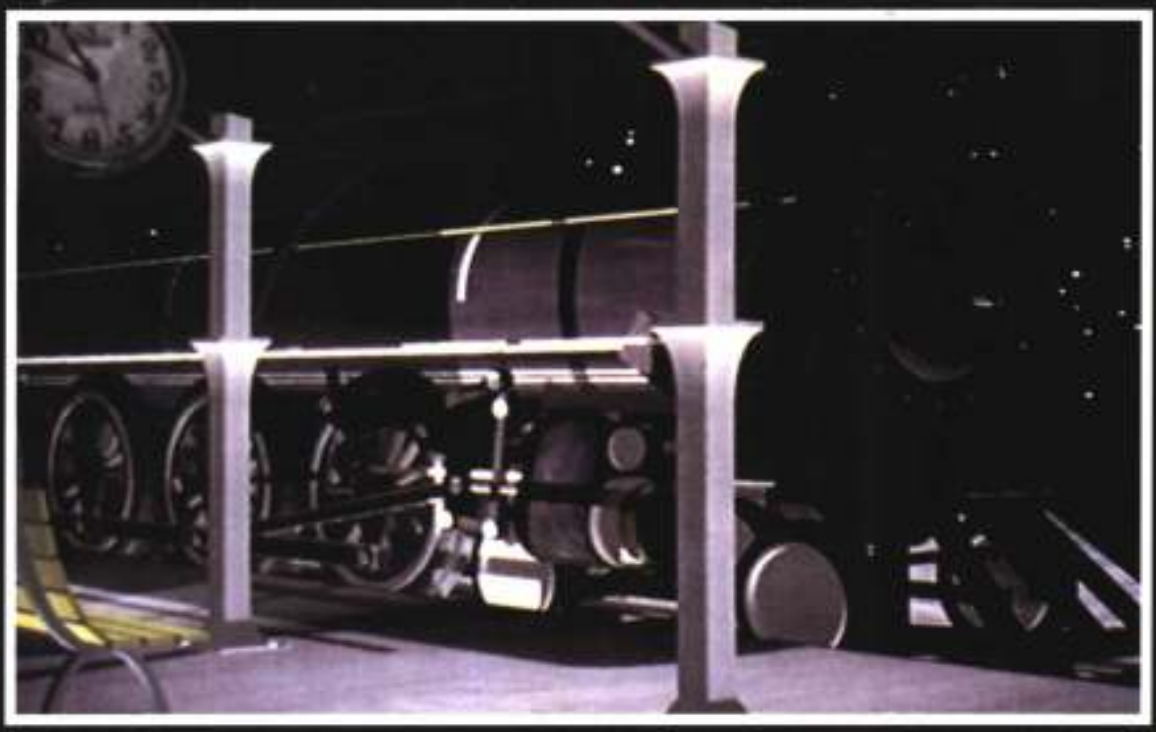
彩图 D "Luxo Jr." 中的一个场景，由J.Lasseter、W.Reeves、E.Ostby和S.Leffler制作。(Pixar公司版权所有，1986。)



彩图E 光线跟踪的辐射度模拟。由康奈尔大学计算机图形学组的K.Howie，B.Trumbore和D.P.Greenberg提供。（康奈尔大学计算机图形学组版权所有，1989。）

彩图F “Red's Dream”中的一个场景，由J. Lasseter, E.Osthy，W.Reeves和H.B.Siegel提供。（Pixar公司版权所有，1987。）





彩图G　“Onlyville”，由S.Snibbe和D.Robbins制作。（布朗大学计算机图形学组版权所有，1989。）

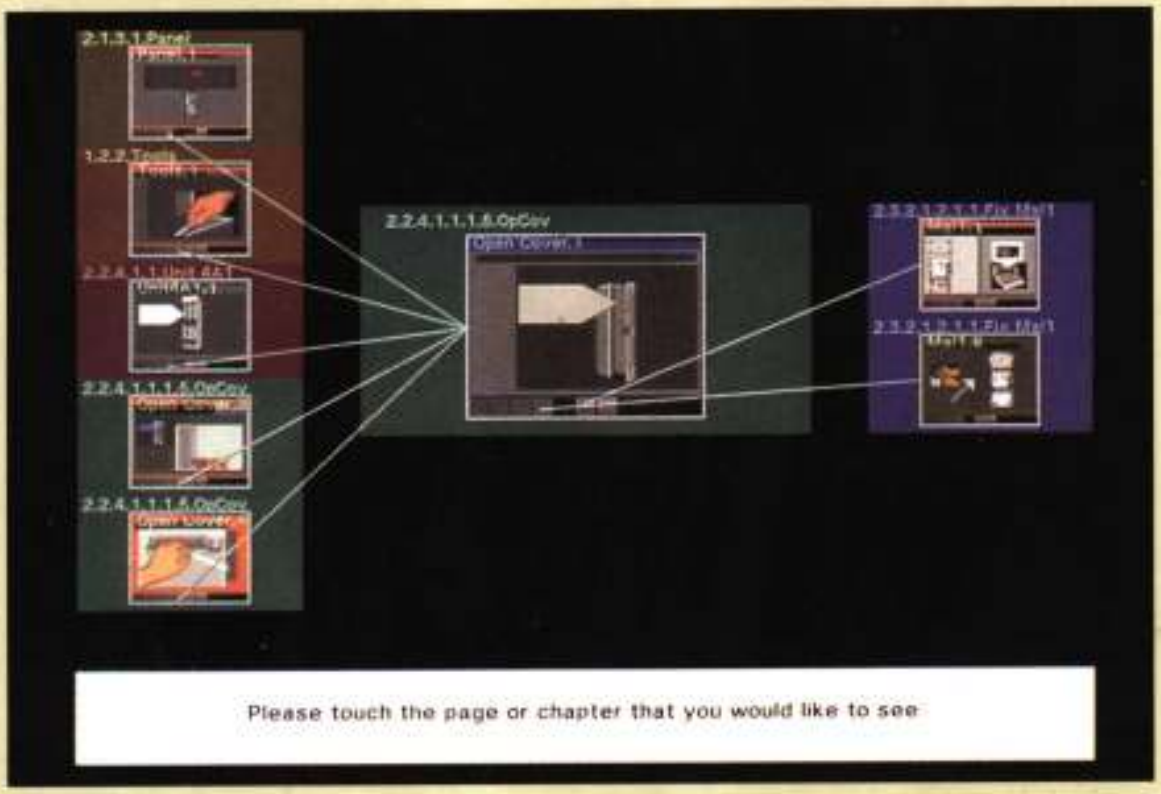
彩图H　“彩色的舞蹈”。四种不同的颜色空间分别是HSV 六棱锥、Ostwald双圆锥体、RGB立方体和CIE颜色空间。（图像由D.Laidlaw和B.Meier提供，布朗大学计算机图形学组的Laidlaw/Meier版权所有。）





彩图 I-1 放射治疗规划模拟。体绘制的方法用来显示放射线束与一个小孩大脑之间的交互。（由Silicon Graphics公司R.Drebin提供。Pixar公司版权所有，1988。CT扫描图像由N.V.Philips公司的F.Zonneveld提供。）

彩图 I-2 局部图像由IGD的超媒体系统自动产生。图像将当前正被访问的节点放在中间，可能的源节点放在左栏，可能的目的节点放在右栏。（经布朗大学的计算机图形学组的S.Feiner、S.Nagy和A.van Dam许可使用。）





彩图 I-3 将一个球体内部翻转出来的平滑处理过程中的一个阶段。这个球体被切成薄片来显示其内部结构。（布朗大学计算机图形学组的John Hughes版权所有，1989。）

彩图 I-4 球体与超球体分别在三维和四维空间中旋转的立体图投影的动画序列。更进一步的信息请参阅[KOCA87]。（由D.Laidlaw 和H.Kocak提供。）





a)

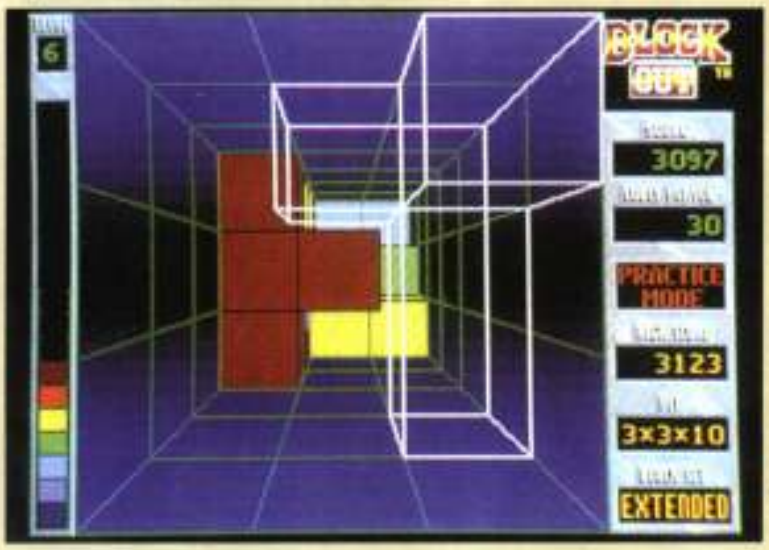
b)



彩图 I-5 a) 一架F5飞行模拟器的驾驶舱；驾驶员视图投影在驾驶舱圆顶上。b) 从飞行模拟器驾驶舱里所见的视图。地形是用照片贴的纹理，而战斗机是采用几何建模的。（经通用电器公司的R.Economy许可使用。）

彩图 I-6 一个视频游戏。在游戏里，玩家必须把一些三维形状组合放在一个小空间里。这里用到了深度的透视画法和线条图。（经加利福尼亚Dreams公司的Larry Lee许可使用。）

▼

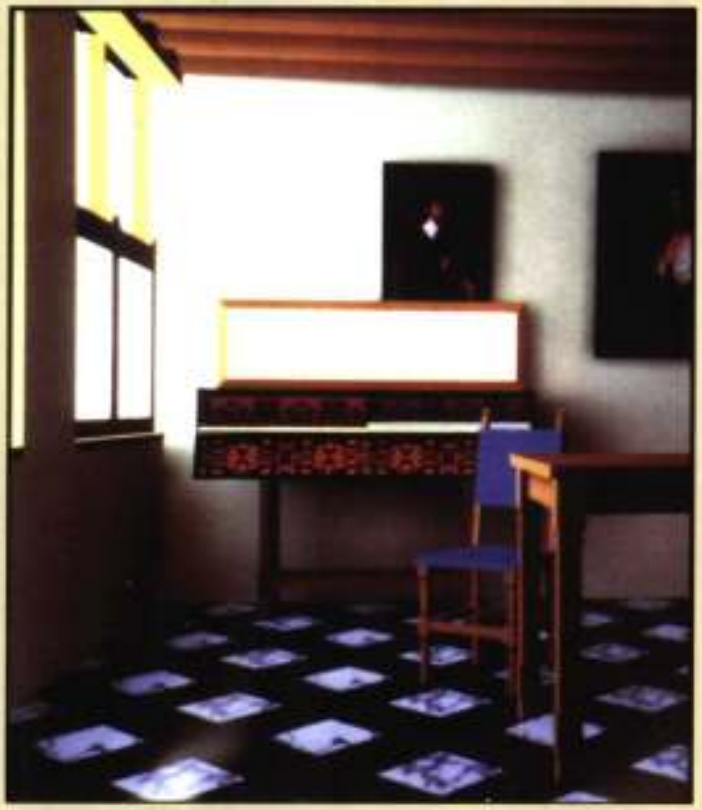




彩图 I-7 Hard Drivin' 娱乐厅视频游戏。（经Atari Games公司提供，Atari Games公司版权所有，1988。）

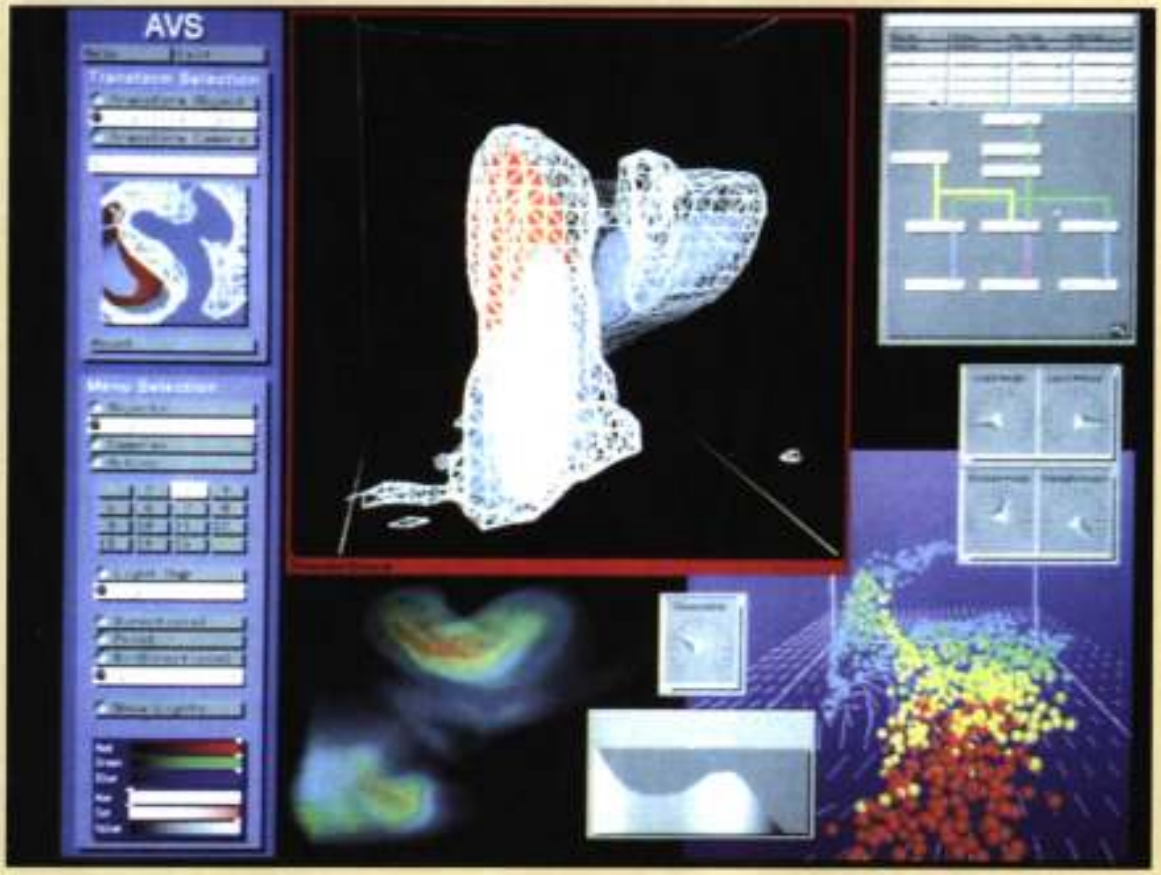
彩图 I-8 由Innovis公司研制的“改善家居设计”系统。（经位于华盛顿州塔科马的Innovis Interactive Technologies许可使用。）





彩图 I-9 佛梅尔的“荷兰的室内”，由康奈尔大学的J.Wallace、M.Cohen和D.Greenberg提供。（康奈尔大学计算机图形学组版权所有，1987。）

彩图 I-10 强龙卷风，由Illinois大学NCSA的R.Wilhelmson、L.Wicker和C.Shaw提供。（Stardent Computer公司的应用可视化系统——AVS系统。）





a)



b)

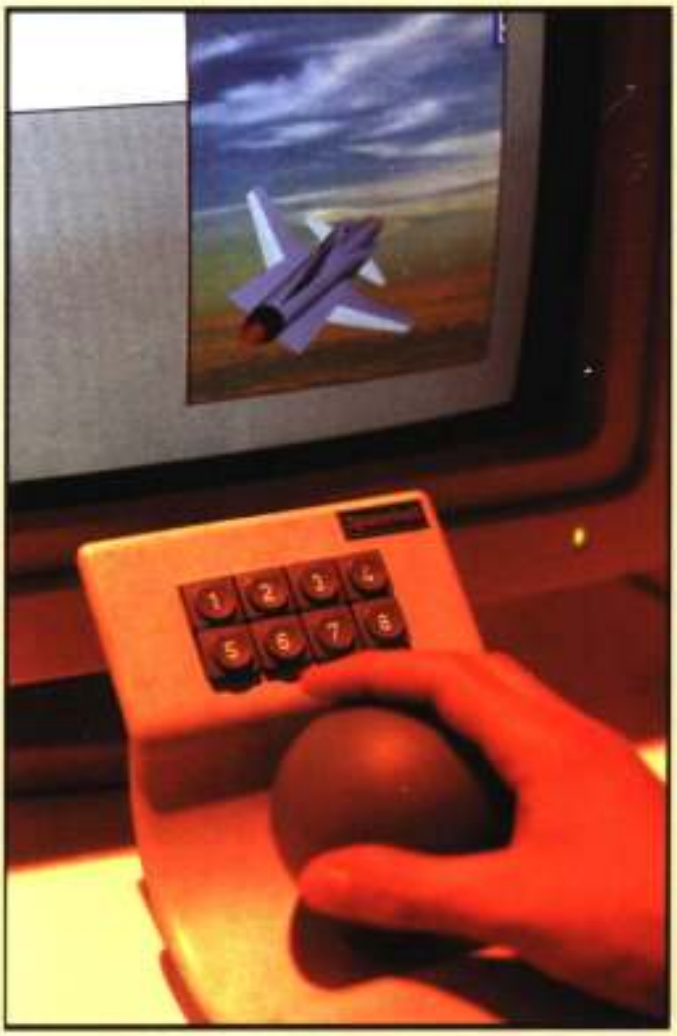
彩图 I-11 a) 雪佛兰高级货车，产品上市资料。（Digital Productions版权所有，1985。）b) 城堡内部，（Digital Productions 公司版权所有，1985。两幅图像均经G. Demos许可使用。）

彩图 I-12 The Abyss——假足动物游戏（Twentieth Century Fox公司版权所有，1989，保留所有权利。经Industrial Light & Magic 的计算机图形部门许可。）





彩图 I-13 《The Last Starfighter》中的指挥船。这艘经纹理映射后的船由超过450 000个多边形构成。（Digital Productions版权所有，1984。经G.Demos许可。）



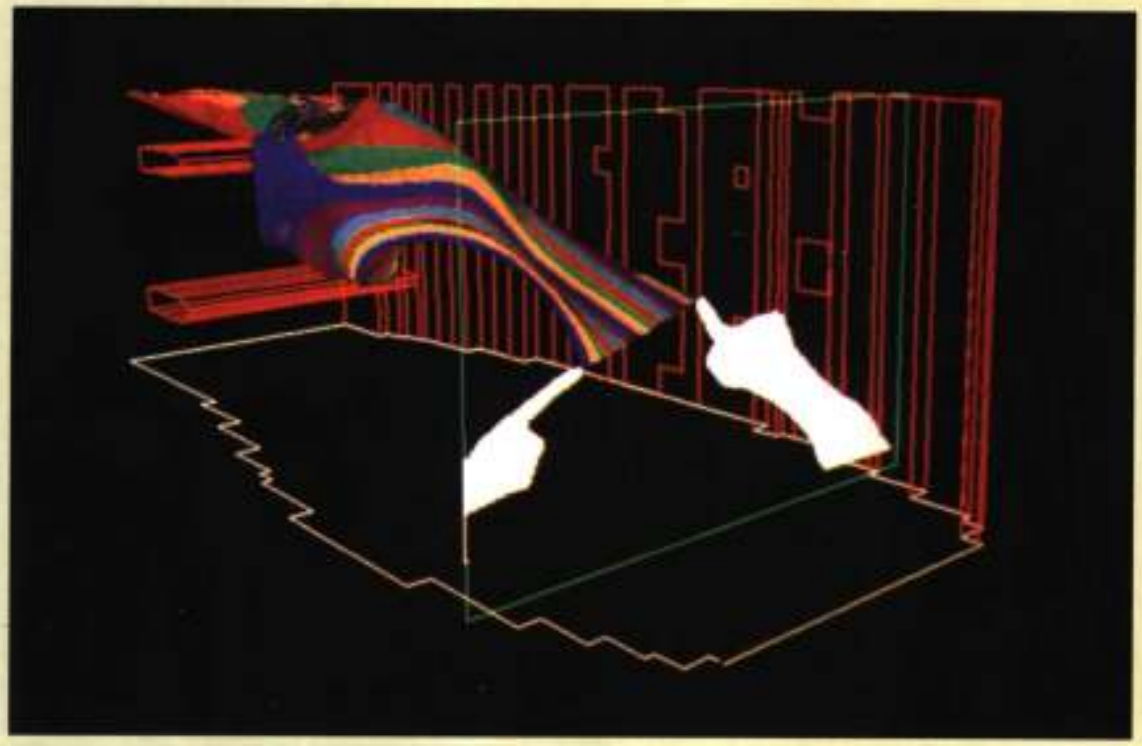
彩图 I-14 六自由度空间球定位装置。（经Spatial Systems公司许可。）



彩图 I-15 数据手套（DataGlove）（右）及其相应的计算机图像。数据手套用于测量手指位移量以及手的位置和朝向。相应的计算机图像也随之变化。（经VPL的Jaron Lanier许可。）

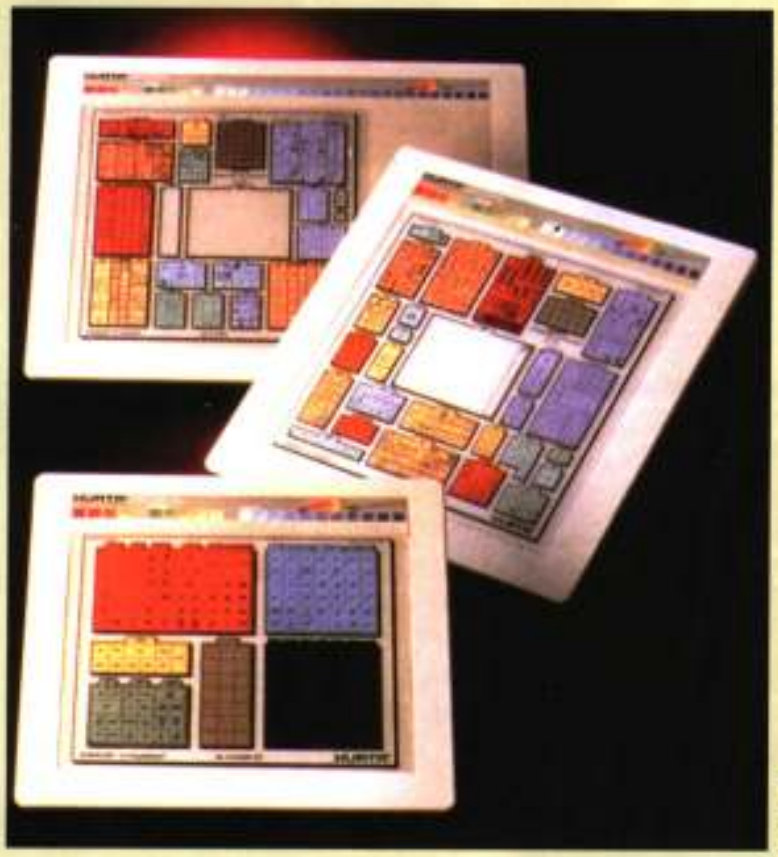
彩图 I-16 用户穿戴着一种头盔立体显示器、数据手套和用于发出命令的麦克风。这些设备用于构造虚拟现实环境，立体显示场景的变化通过头部的转动来实现，数据手套用于操纵计算机生成的物体。（经位于加利福尼亚州Moffett Field的NASA Ames研究中心的Michael McGreevy和Scott Fisher许可。）





▲

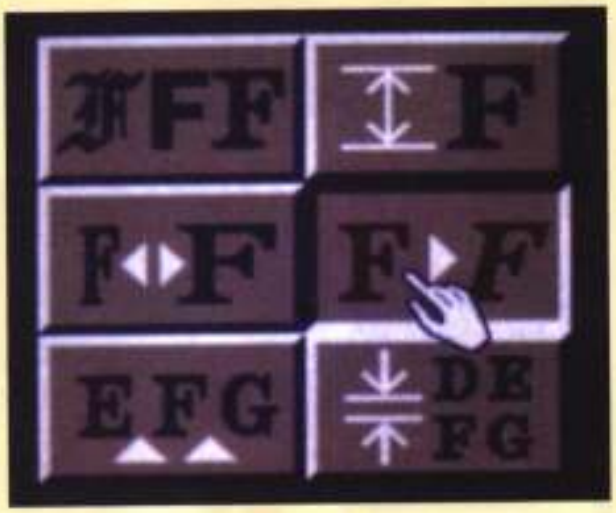
彩图 I-17 Krueger的Videotouch系统，其中用户用手的运动来操纵物体。手的轮廓与物体都显示在屏幕上，以提供自然的反馈。（经Artificial Reality公司许可。）



彩图 I-18 基于输入板的菜单。菜单项根据颜色来分组。（经Kurta公司许可。）



彩图 I-19 利用手形来显示操作含义的菜单。图中的第二行和第三行显示了物体操作前后的图形以表达操作的含义。图的顺序从左到右，再从上到下，操作命令依次为：File，Delete，Shear，Rotate，Move和Copy。（经Peter Tierney许可，Cybermation公司版权所有，1988。）



彩图 I-20 操作文本的菜单。从左到右，再从上到下的按钮含义分别为：选择字体，设置字体高度，设置字体的宽度（表示字体变化前后的宽度），变斜体（表示字体变化前后形状），设置字母间隔，设置行间隔。（经Peter Tierney许可，Cybermation公司版权所有，1988。）



彩图 I-21 对几何对象的操作菜单。所有的按钮都显示了操作前后的图形。从左到右，再从上到下，按钮的含义依次为：移动点，改变圆弧的半径（或者把直线段变为圆弧），添加线段的顶点（把一条线段变成两段），删除顶点（把两条线段变成一条），圆角和切角。（经Peter Tierney许可，Cybermation公司版权所有，1988。）



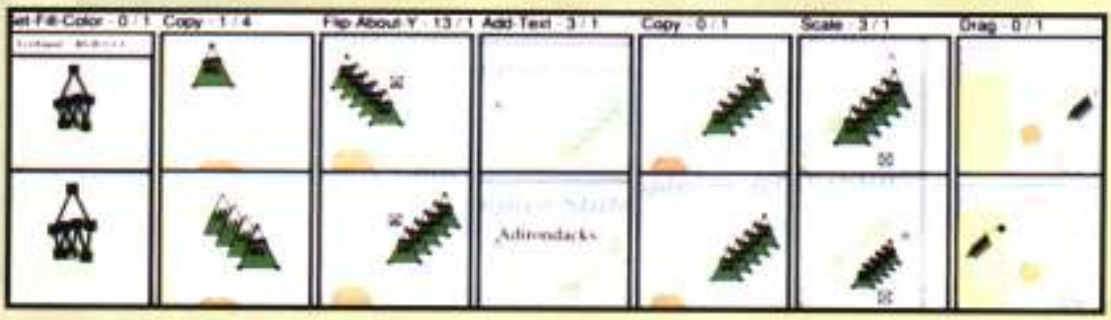
彩图 I-22 NeXT的用户界面。（经NeXT公司许可，NeXT公司版权所有，1989。）

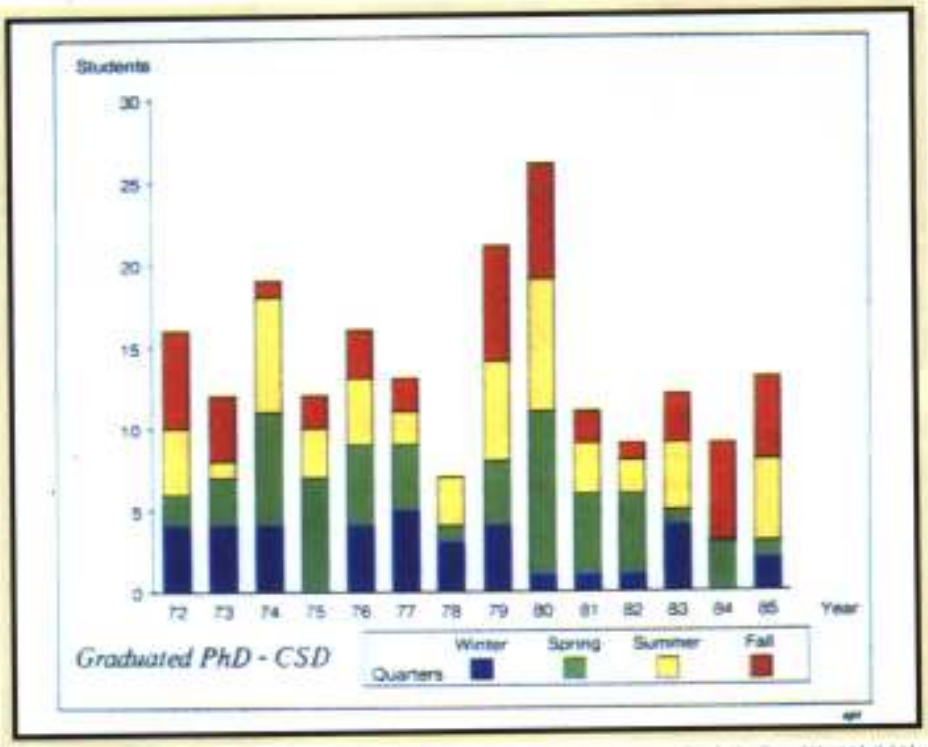
彩图 I-23 Chimera中的图形编辑过程。a) 编辑窗口显示了Chimera的用户制作的图片。b) 过程窗口显示了用Chimera制作的平面对前后的过程。（由哥伦比亚大学 David Kurlander和Steven Feiner提供。）



a)

b)





彩图 I-24 彩图堆叠的直方图，由A Presentation Tool（APT）自动生成，用于绘制从1972年到1985年期间每个季度的毕业生数量。用APT描述要比其他很多方法更有效。（经Jock Mackinlay许可。）

彩图 I-25 由IBIS产生的收音机的图片。图片显示了满足输入交流目标的功能刻度盘的位置及其状态变化。IBIS确定了所包含的对象、绘制样式、观察和光照的规格以及图片的组件。（经哥伦比亚大学Dorée Duncan Seligmann和Steven Feiner许可。）

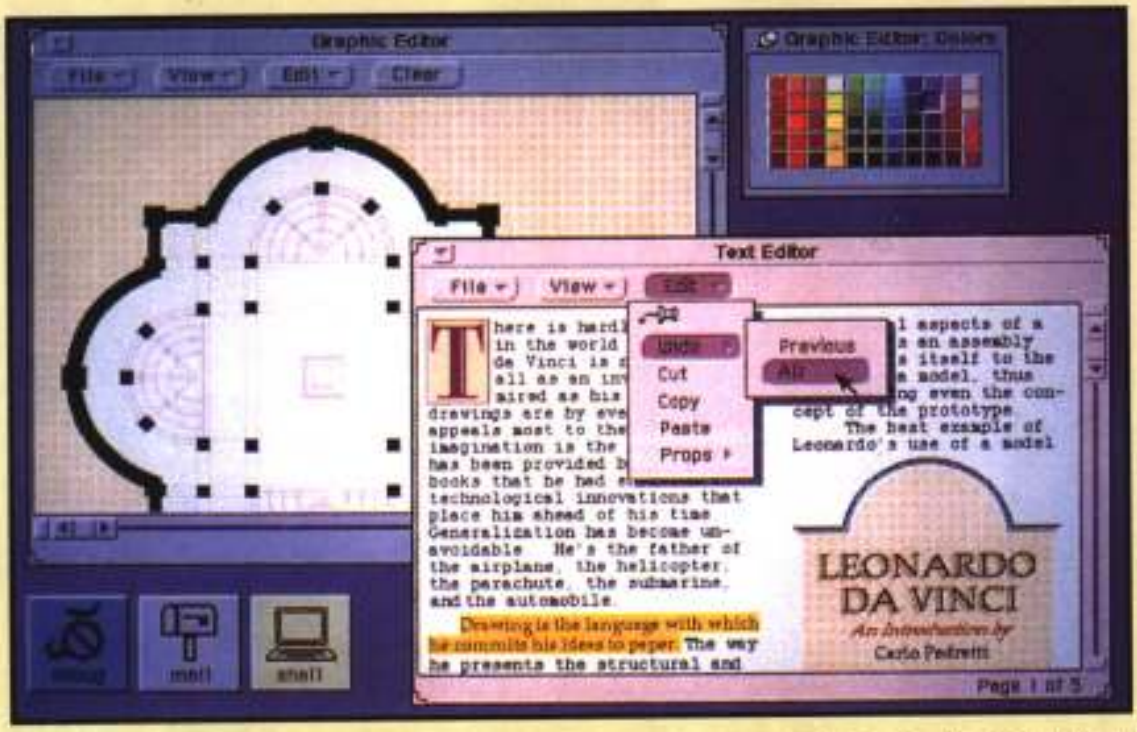




彩图 I-26　OSF/Motif的用户界面，在这幅图像中，深浅不同的蓝色用于区分不同的视觉元素。（经开放软件基金会（OSF）许可。）

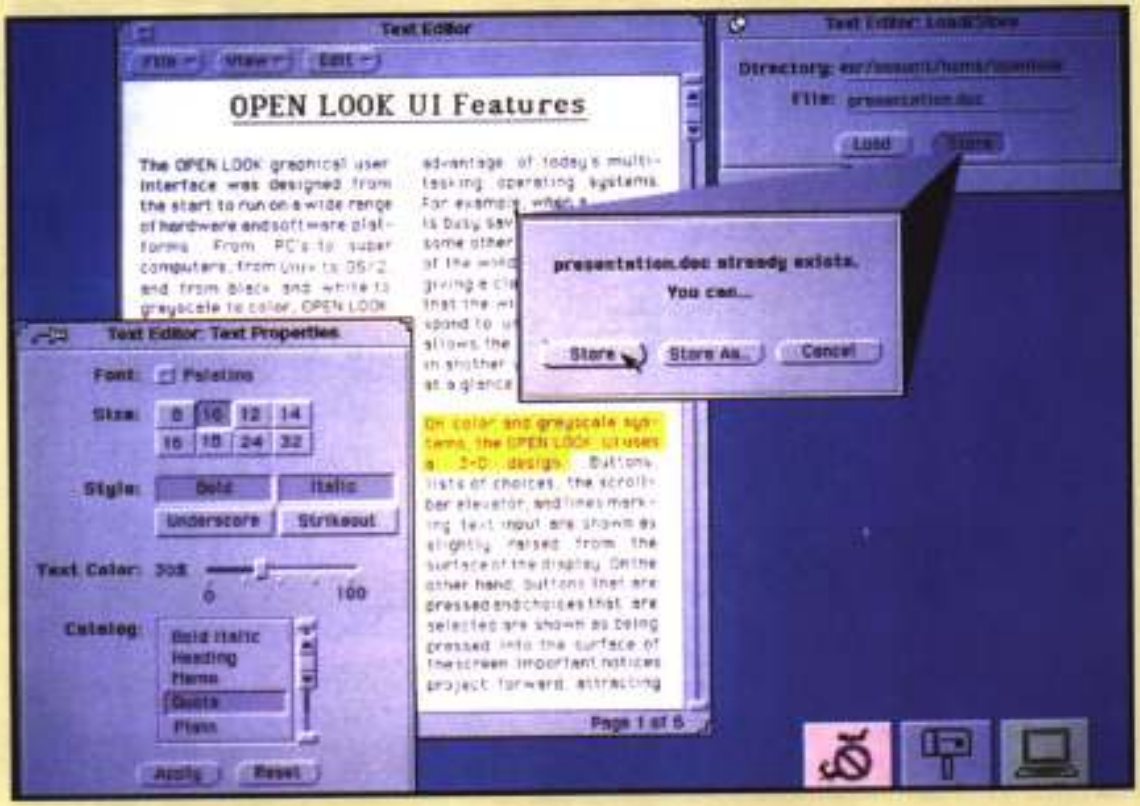
彩图 I-27　OSF/Motif的用户界面，其中颜色滚动条用于定义不同窗口的颜色。注意，在按钮、菜单等边缘处使用明暗处理以创建三维效果。（经开放软件基金会（OSF）许可。）

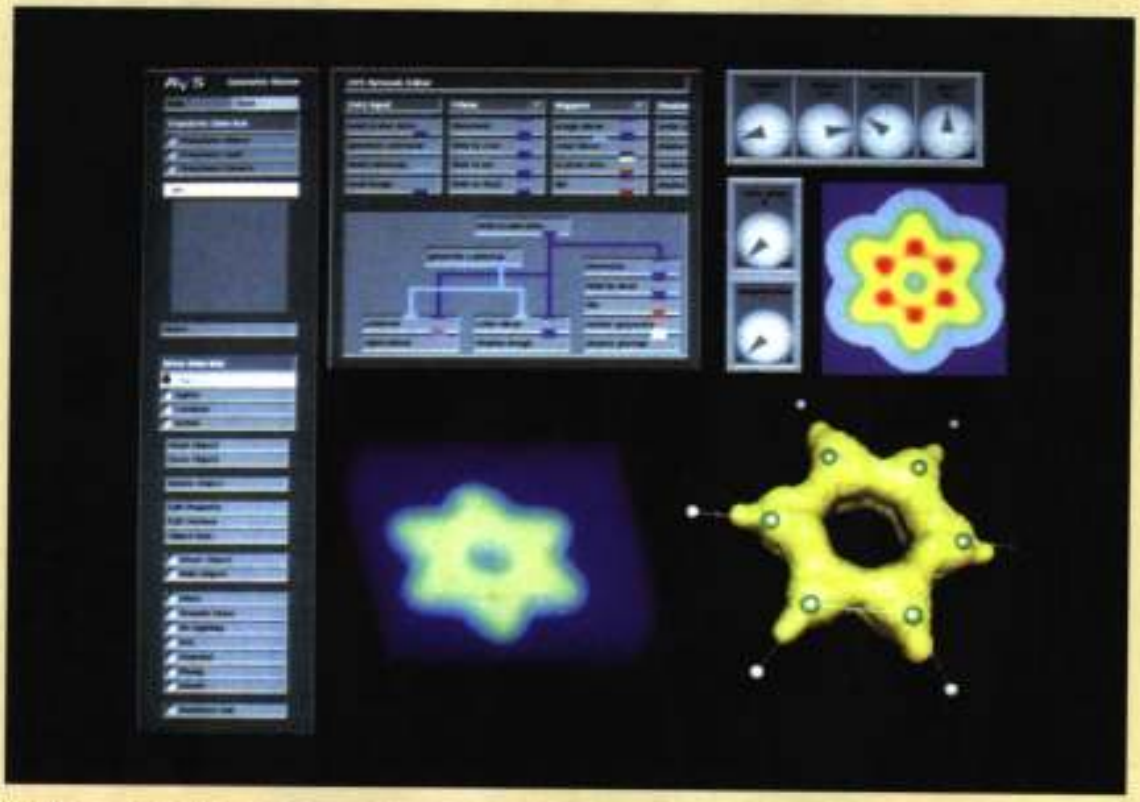




彩图 I-28 OPEN LOOK的用户界面。黄色用于加亮所选择的文本，柔和的阴影用于窗口的背景和边缘。（经Sun Microsystems公司许可。）

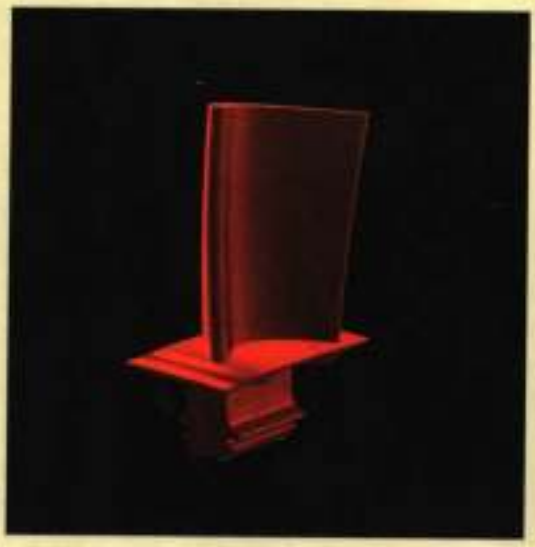
彩图 I-29 OPEN LOOK的用户界面。（经Sun Microsystems公司许可。）



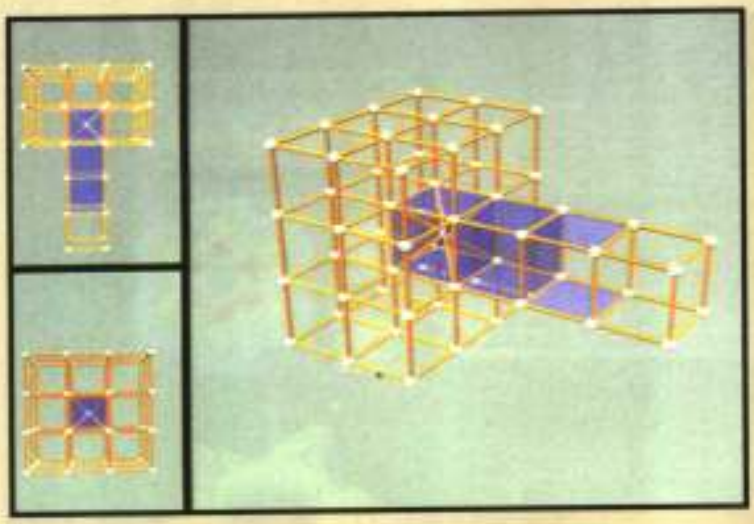


彩图 I-30 应用可视化系统（AVS）的用户界面。数据流程图（位于窗口中间）由用户从处理元素菜单（位于窗口顶部中间）获取并通过交互方式构造。在这个例子中，通过控制盘（到右边的）的输入操作控制其XYZ方向上的显示。（经Stardent公司许可。）

彩图 I-31 用NURBS构造的实体，用Alpha_1造型系统生成。（经犹他大学许可。）







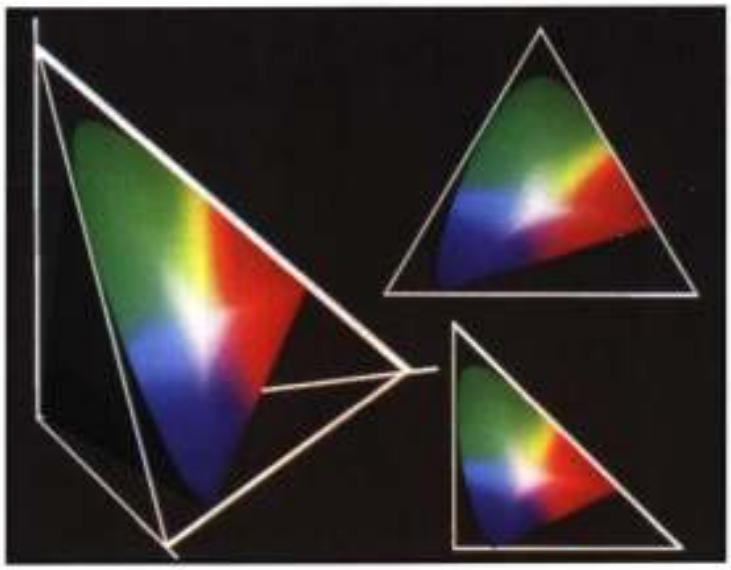
彩图 I-32　三维非流形实体的三个视图，其中蓝色表示实体的面，黄色表示边，白色表示顶点。在所示的模型中，有两个立方体共享一个面，最右边的立方体的线框上有两个悬挂面，最左边的立方体有内部和外部有限元方法的网格线框。（经K. Weiler、D.McLachlan, H. Thorvaldsdóttir许可；三维实体利用Dóre生成，Ardent Computer公司版权所有，1989。）



彩图 I-33　PANTONE®颜色说明书第747XR页的一部分。颜色名称（在这里已涂黑）随颜色样品显示。颜色名称提供了利用标准墨水混合配置这种颜色的线索。（经Pantone公司许可，PANTONE®是Pantone公司用于颜色再生和颜色再生材料的检验标准的注册商标。做颜色再生可能会与PANTONE®检验实体颜色标准不一致。参见最近PANTONE®颜色出版物可获得精确的颜色。）

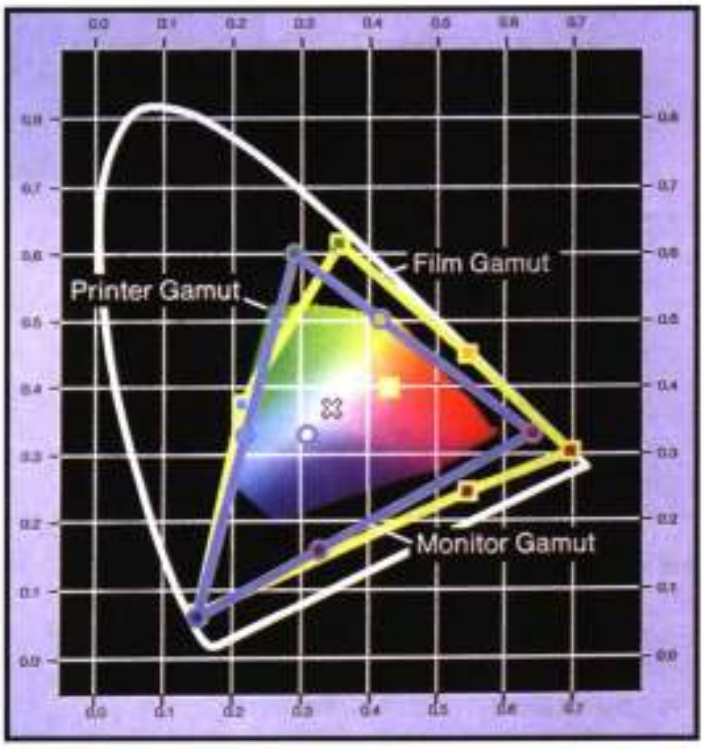
彩图 I-34　颜色色谱，从左到右依次为紫色到红色。曲线的高度表示发光物C的光谱能量分布。（经布朗大学Barbara Meier许可。）

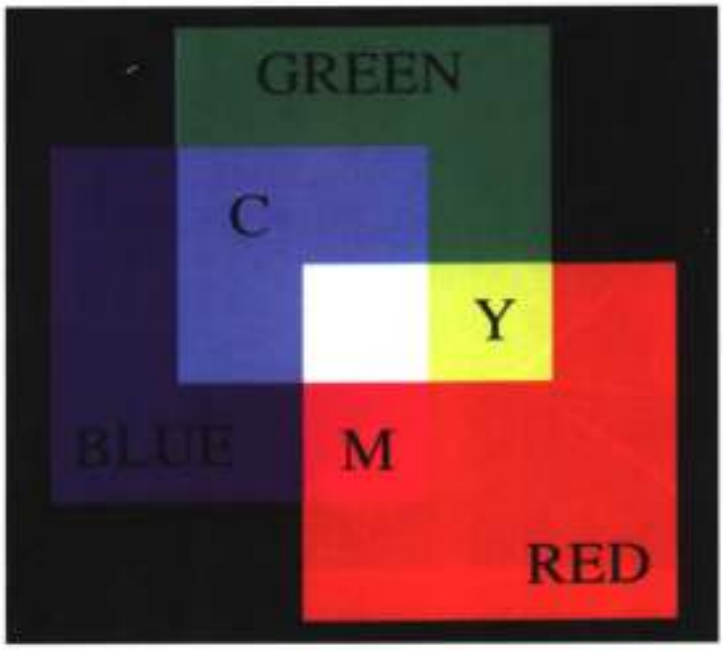




彩图 II-1 CIE空间的$X+Y+Z=1$平面的几个视图。左边是嵌入在CIE空间内的平面，右上是该平面的垂直正面视图，右下是在（X,Y）平面（即$Z=0$平面）上的投影图，即是CIE色度图。（经布朗大学Barbara Meier许可。）

彩图 II-2 显示了打印机、彩色监视器和电影胶片在CIE色度图上的典型颜色域。打印机的颜色代表一种在5000°K绝对温度的图形艺术光下测量的图形艺术技术基金会（GATF）的S.W.O.P标准颜色。彩色监视器的型号为Barco CTVM 3/51，它采用白色点集、工作在6500°K绝对温度下。电影胶片是Kodak Ektachrome 5017 ISO 64，在CIE的A类条件（即近似在一种Tungsten灯光的2653°K绝对温度的黑体条件）下感光的颜色域。叉号、小圆及小方块分别表示打印机、彩色监视器及电影胶片的白点。（经Xerox PARC的M. Stone许可。电影胶片颜色域是由滑铁卢大学图形实验室的A. Paeth 测量的，可参见[PAET89]的第一个附录。）





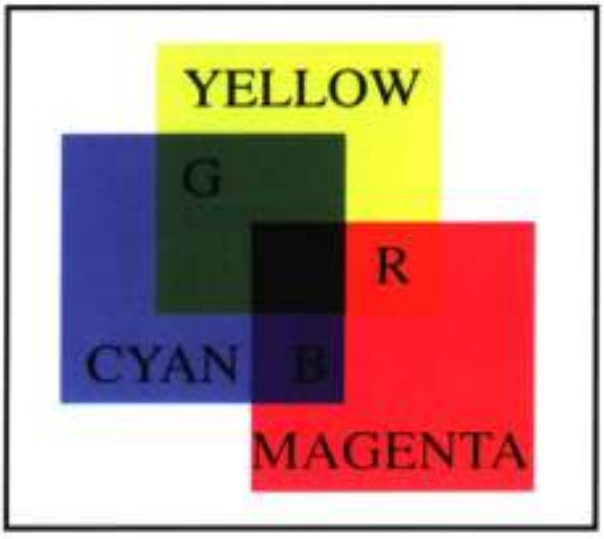
彩图 II-3 加性基色。红色加绿色形成黄色，红色加蓝色形成品红色，绿色加蓝色形成青色，红色加绿色加蓝色形成白色。



彩图 II-4 沿着白色到黑色的主对角线观察的RGB颜色空间。仅黑色顶点不可见。（经MIT媒体实验室，Cambridge，MA 02139的可视语言工作组David Small许可，MIT版权所有，1989。）

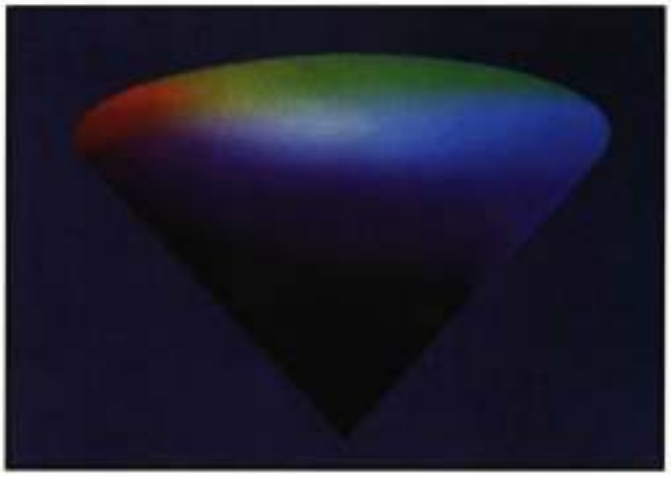


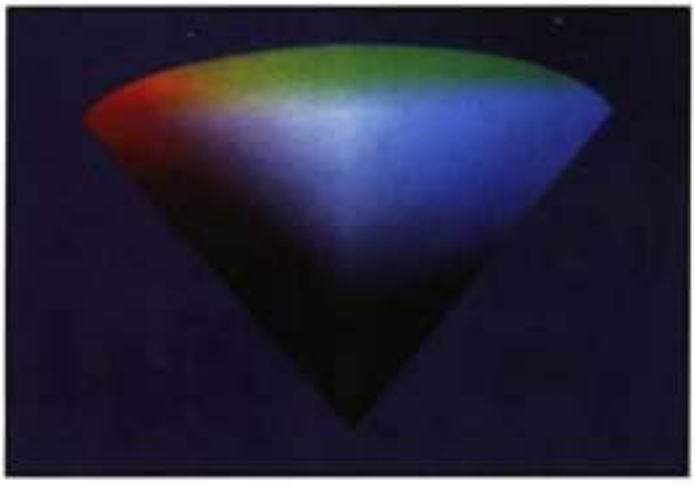
彩图 II-5 RGB颜色空间的一个内部子立方体。灰色顶点位于（0.5，0.5，0.5）。因此，该子立方体的长、宽、高是彩图II-4显示的整个颜色空间长、宽、高的一半。（经MIT媒体实验室，Cambridge，MA 02139的可视语言工作组David Small许可，MIT版权所有，1989。）



彩图 II-6 减性基色。从白色中减去黄色和品红色形成红色，从白色中减去黄色和青色形成绿色，从白色中减去青色和品红色形成蓝色。

彩图 II-7 HSV颜色空间。（经MIT媒体实验室,Cambridge, MA 02139的可视语言工作组David Small许可。MIT版权所有，1989。）





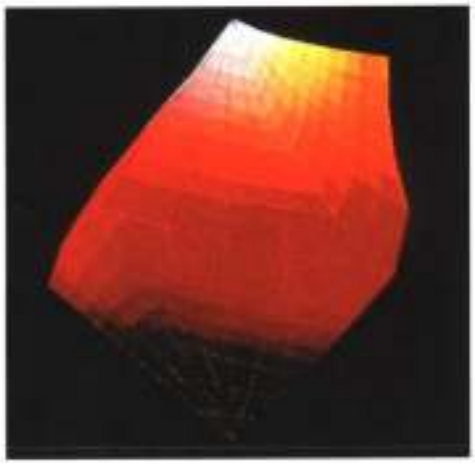
彩图 II-8 经过HSV颜色空间V轴的一个垂直横截面切片图。（经MIT媒体实验室,Cambridge, MA 02139的可视语言工作组David Small许可，MIT版权所有，1989。）



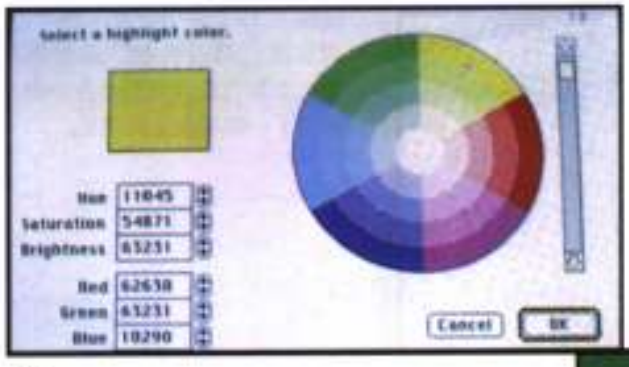
彩图 II-9　HLS颜色空间。（经MIT媒体实验室，Cambridge, MA 02139的可视语言工作组David Small许可，MIT版权所有，1989。）



彩图 II-10　经过HLS颜色空间L轴的一个垂直横截面切片图。（经MIT媒体实验室,Cambridge ,MA 02139的可视语言工作组David Small许可，MIT版权所有，1989。）



彩图 II-11　HVC颜色空间。（经Tektronix公司许可。）

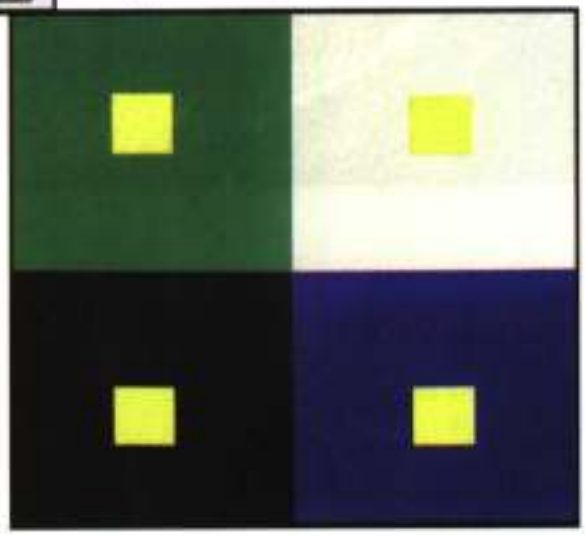


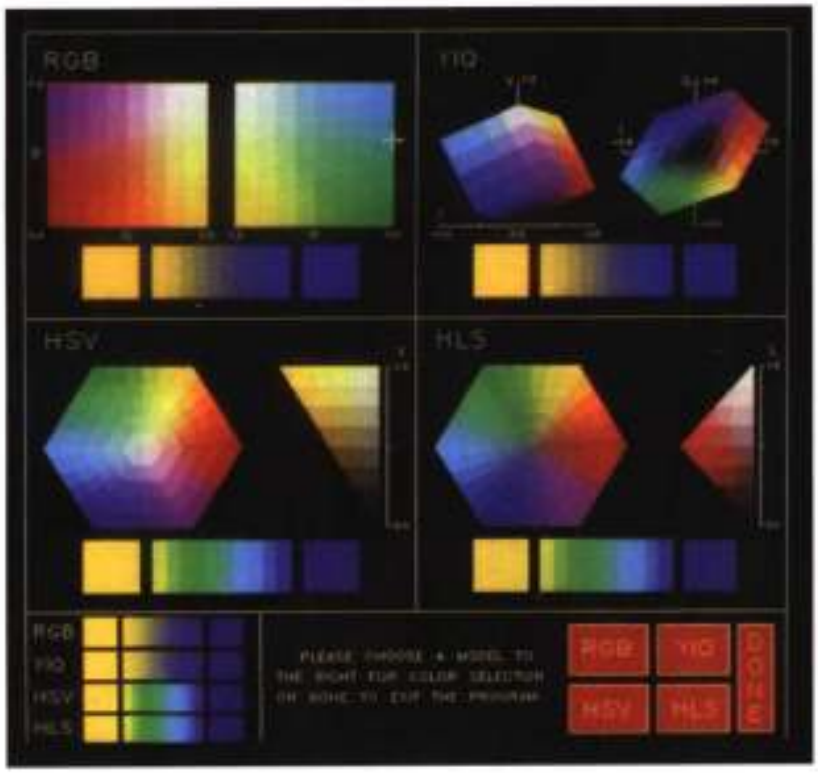
▲

彩图 II-12 Macintosh机器上使用的一种在HSV颜色空间内用于指定颜色的交互技术。色调和饱和度在圆形区域内给出。亮度在滑动条上给出。用户可移动圆形区域内的标记，改变滑动条上的标尺，也可键入新的HSV或RGB的数值。左上角正方形的彩色区域显示了当前的颜色及新的颜色。显示此图片的彩色监视器使用了每像素4位的位图，很多颜色是由抖动技术（见13.4节）生成的。（经Apple公司许可，版权所有，1984。）

彩图 II-13 在不同背景颜色包围下的同样的黄色，黄色深浅度好像不同。

▼

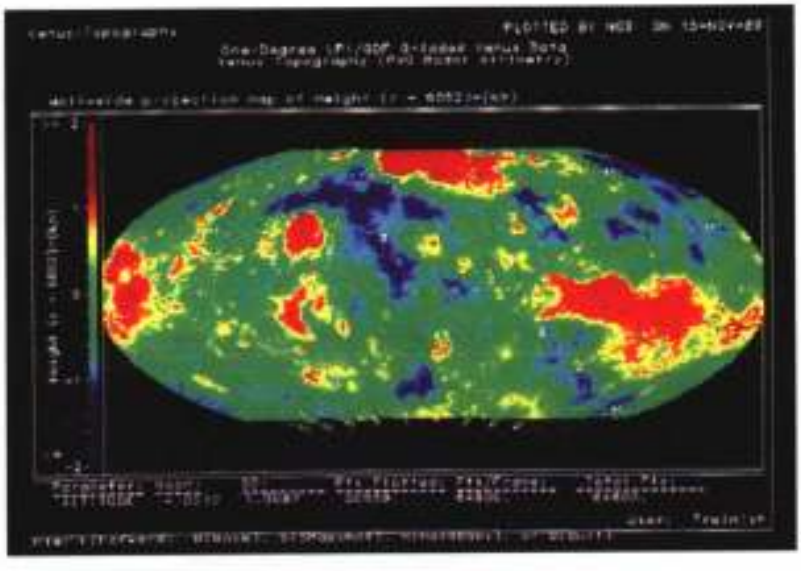




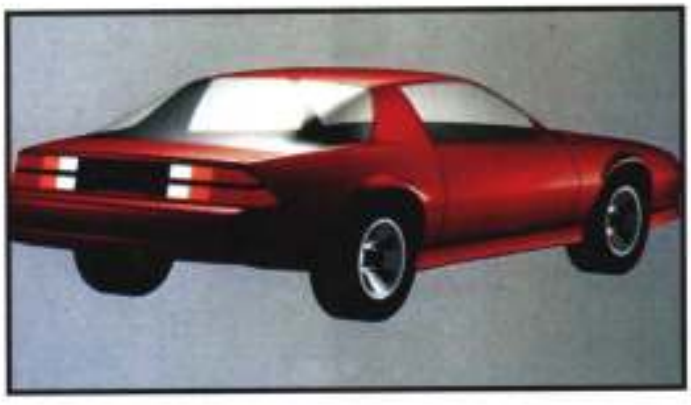
彩图 II-14 可在四种不同颜色空间（RGB、YIQ、HSV及HLS）中，指定颜色并对颜色插值的一个交互程序。线性插值的起点和终点颜色通过在颜色空间的不同投影处定义来指定。每个颜色空间下面显示了插值，并且在左下角给出了比较。（经乔治·华盛顿大学Paul Charlton许可。）



彩图 II-15 一个放大了的半色调彩图。具有青色、品红色、黄色及黑色的点集组合在一起，构建了一幅颜色全的图片。



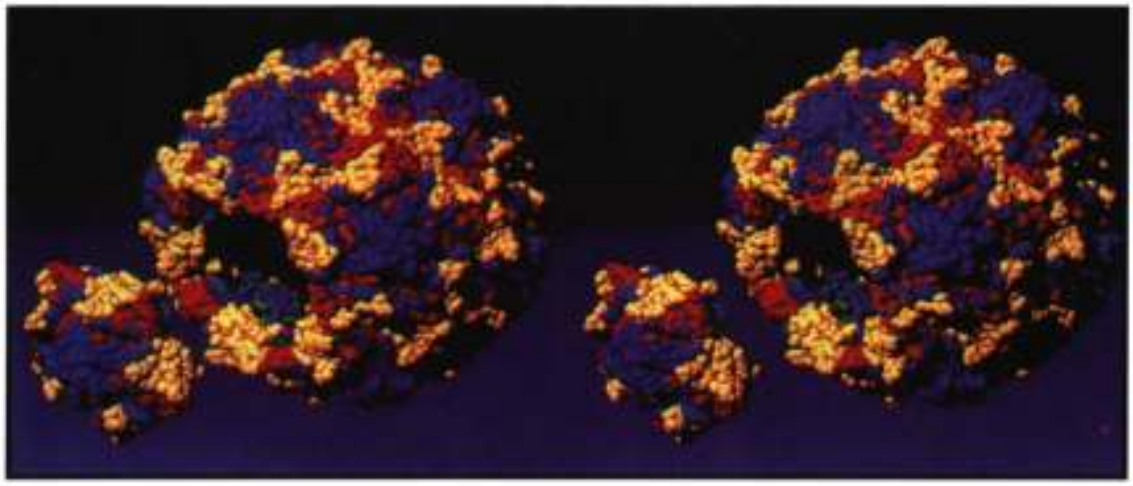
彩图 II-16 一幅显示金星地形的伪彩色图像。左边的颜色标尺指示了在金星平均半径（6052km）上下变化的高度（从－2km到＋2km）。该图像是由NASA的绕金星"先锋"太空船经雷达测量获取数据，由月球及行星研究所对数据进行计算，并由美国国家空间科学数据中心的图形系统绘制的。（经NASA的Goddard空间飞行中心Lloyd Treinish许可。）



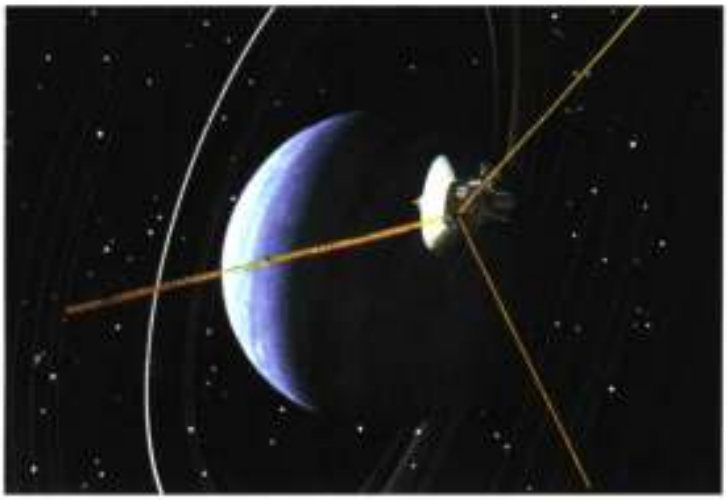
彩图 II-17 按照Warn的光照控制照亮的雪佛兰 Camaro。（经通用汽车公司研究实验室的David R.Warn许可。）



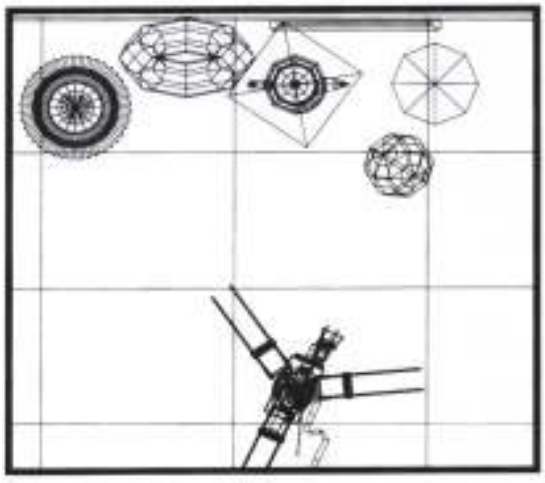
彩图 II-18 1987~1988年度美国NBC的网络节目。由从事商标交流的加州Sunnyvale太平洋数据图像公司的James Dixon（动画制作）和Glenn Entis（制片人）制作。



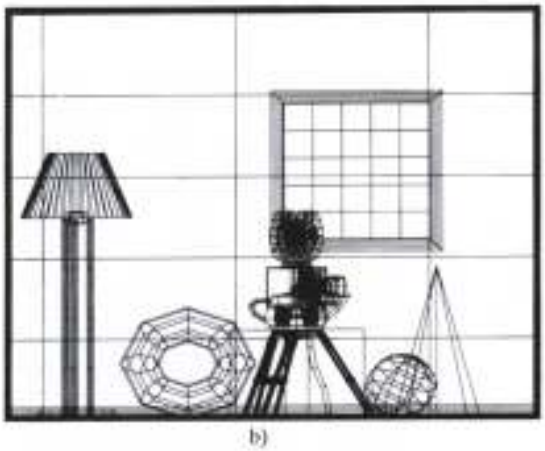
彩图 II-19 一对立体的脑灰质炎病毒的衣壳，通过在每一个Alpha碳元素上放一个半径为0.5nm的小球而得到。移走了一个五节聚化物来显示其内部。经J.Hogle许可。（经David Goodsell 和 Arthur J.Olson许可，Scripps临床研究院版权所有，1989。）



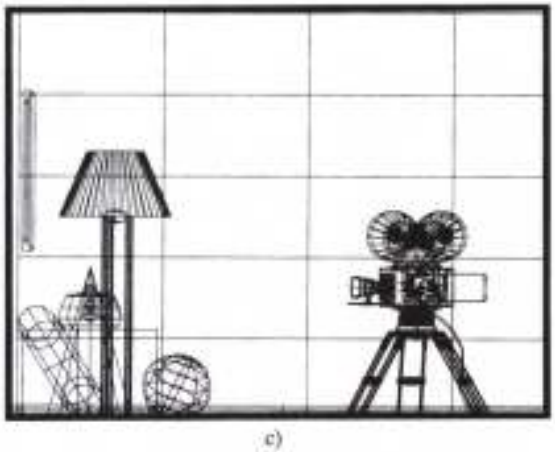
彩图 II-20 模拟的带有光环和轨道的天王星的飞行。（经加利福尼亚理工学院的计算机图形实验室和喷气推动实验室的Jim Blinn许可。）



a)



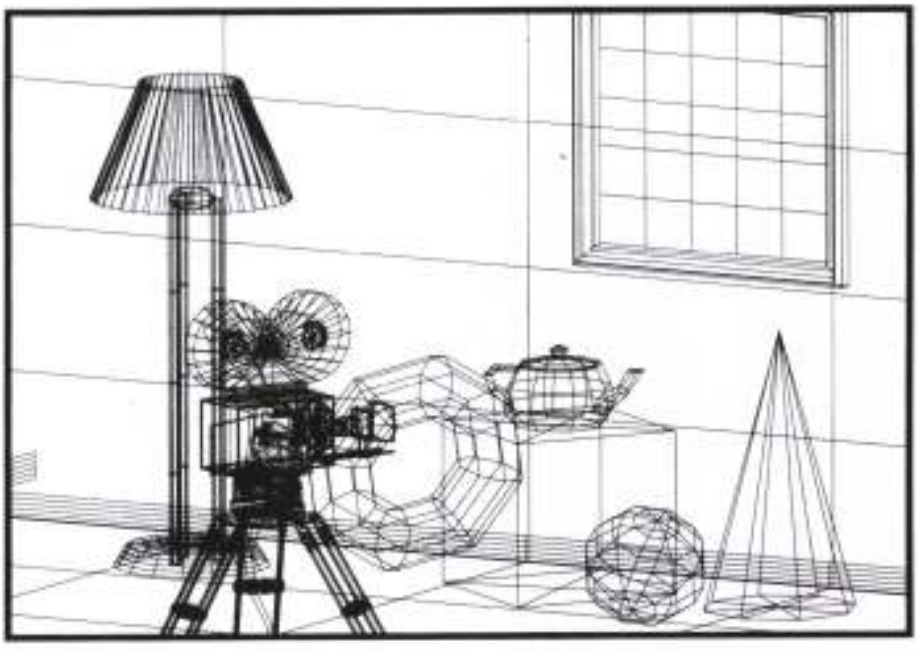
b)

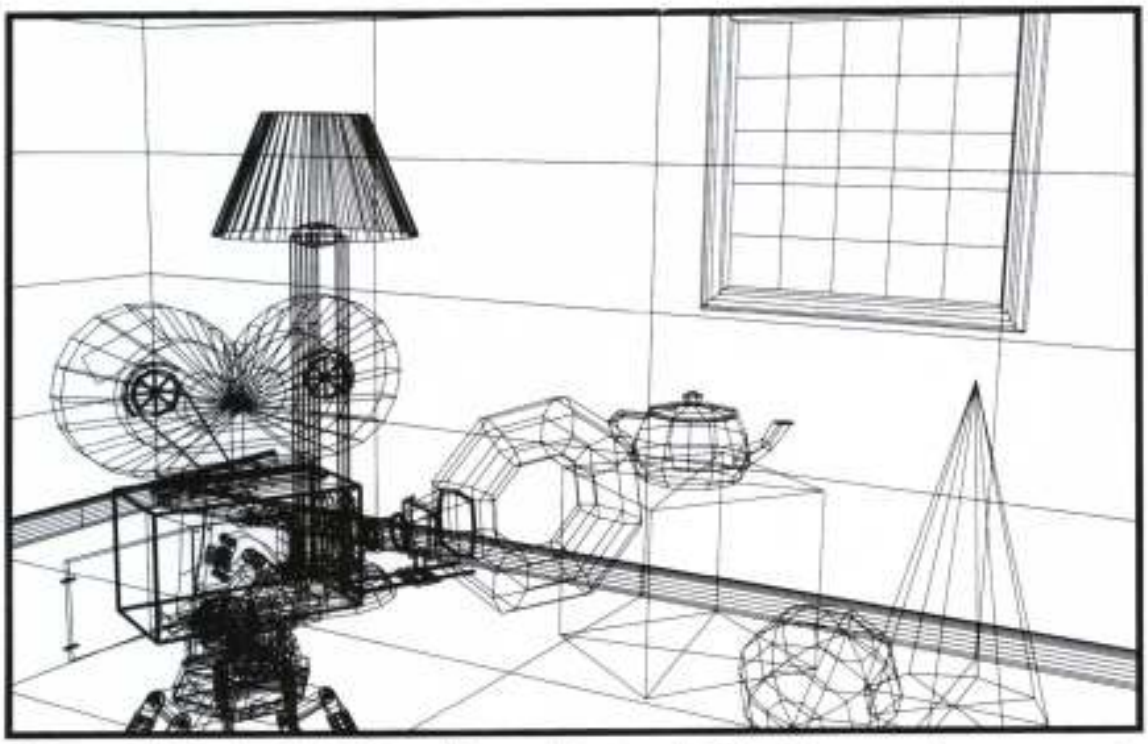


c)

彩图 II-21 摄影爱好者。带有电影摄像机的卧室场景。正交投影（6.1.2节和14.3.1节）。a) 俯视图。b) 正视图。c) 侧视图。多边形模型由样条面片产生。请注意“样条面片”（11.3.5节）在茶壶的右前面是全部可见的。它们如何引起彩图II-30至彩图II-32的采用多边形网格插值明暗处理模型中明暗处理的不连续性。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

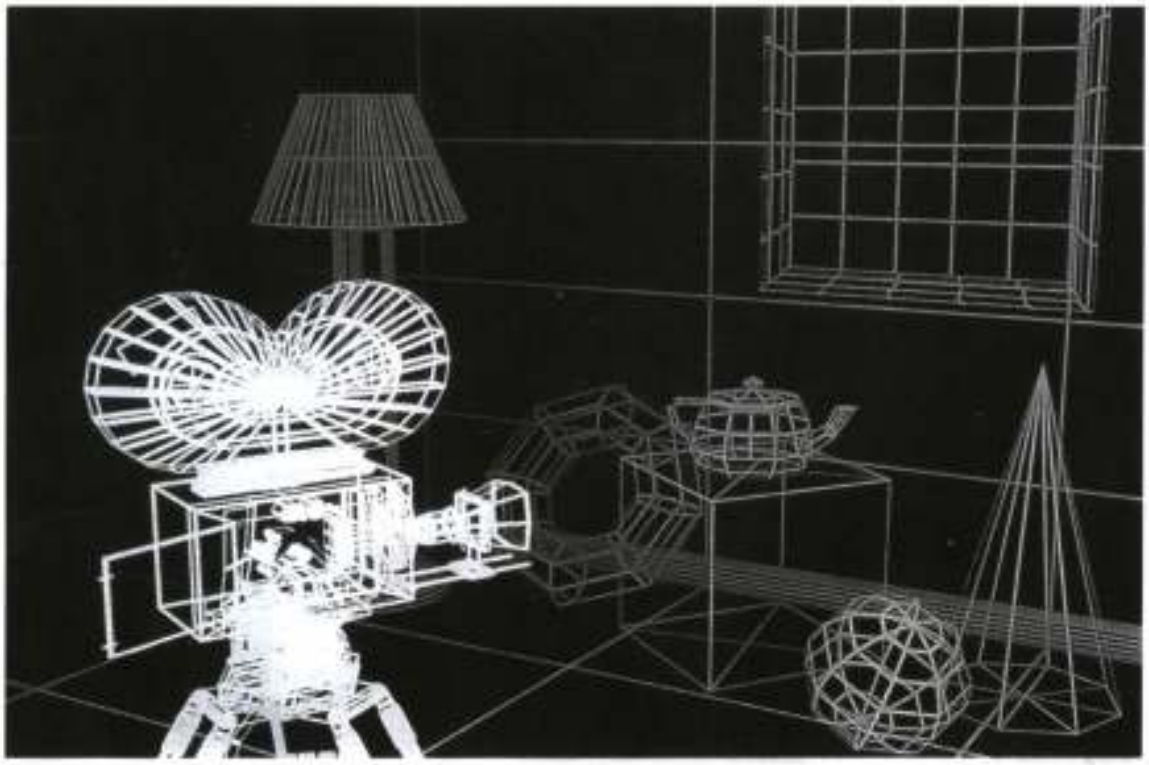
彩图 II-22 摄影爱好者。轴测投影（6.1.2节和14.3.2节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

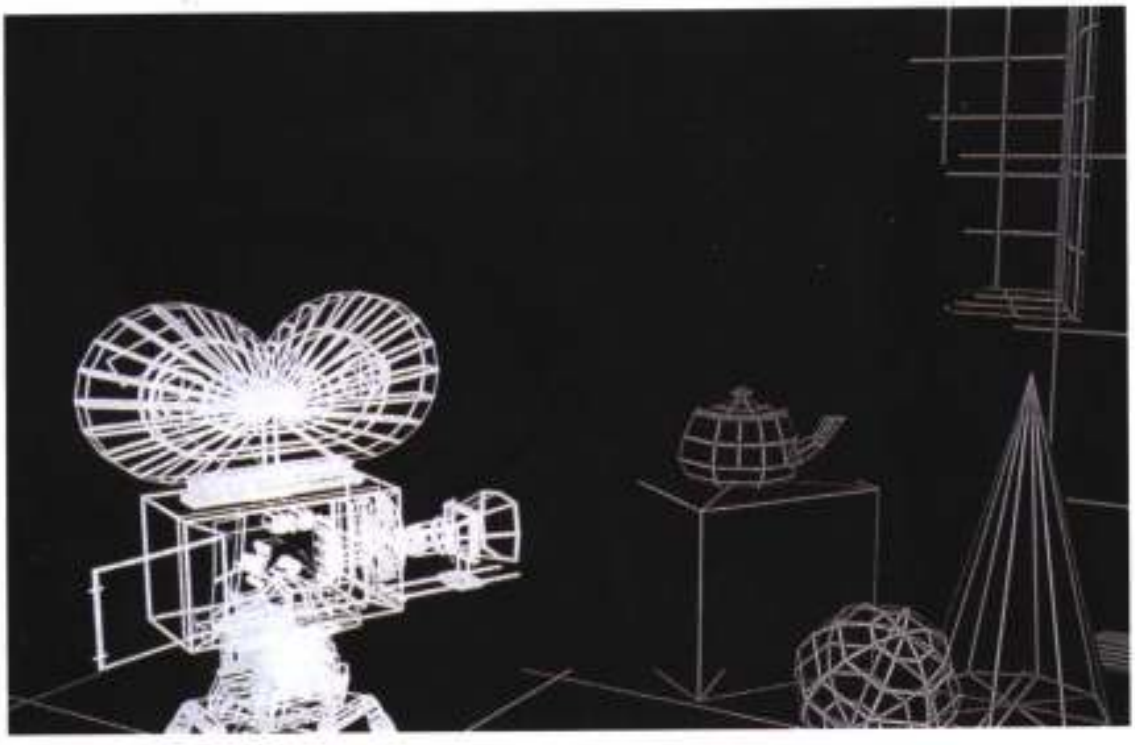




彩图 II-23 摄影爱好者。透视投影（6.1.1节和14.3.3节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

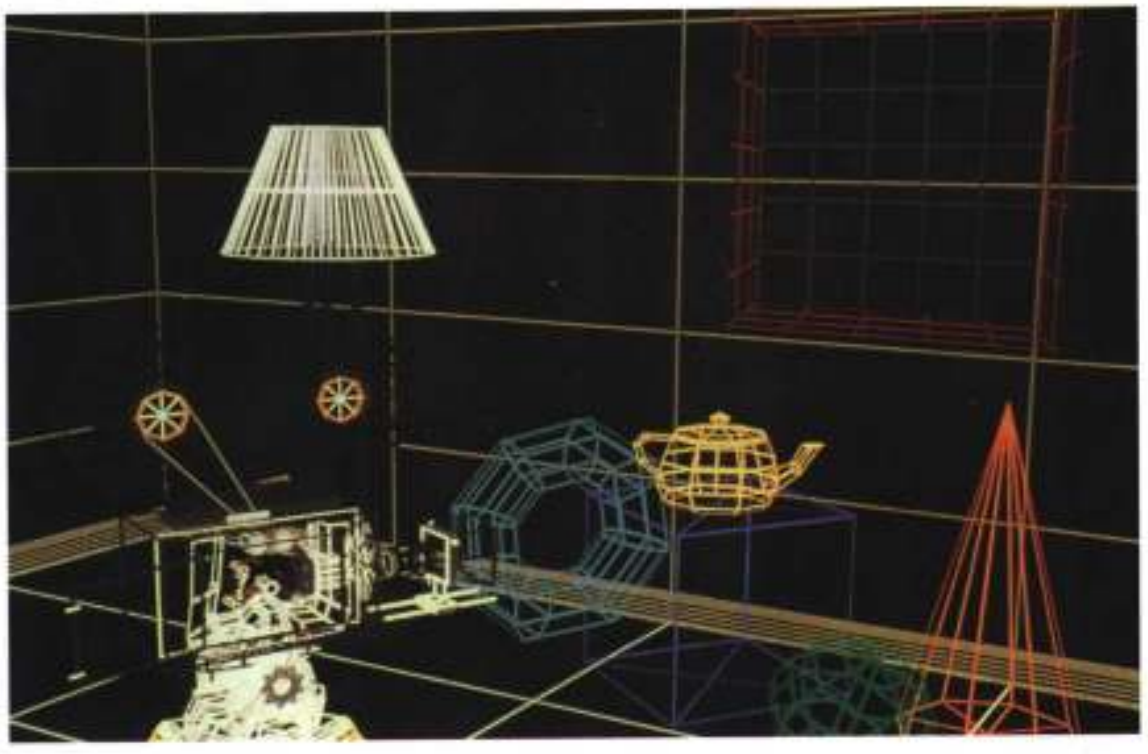
彩图 II-24 摄影爱好者。深度提示（14.3.4节和16.1.3节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

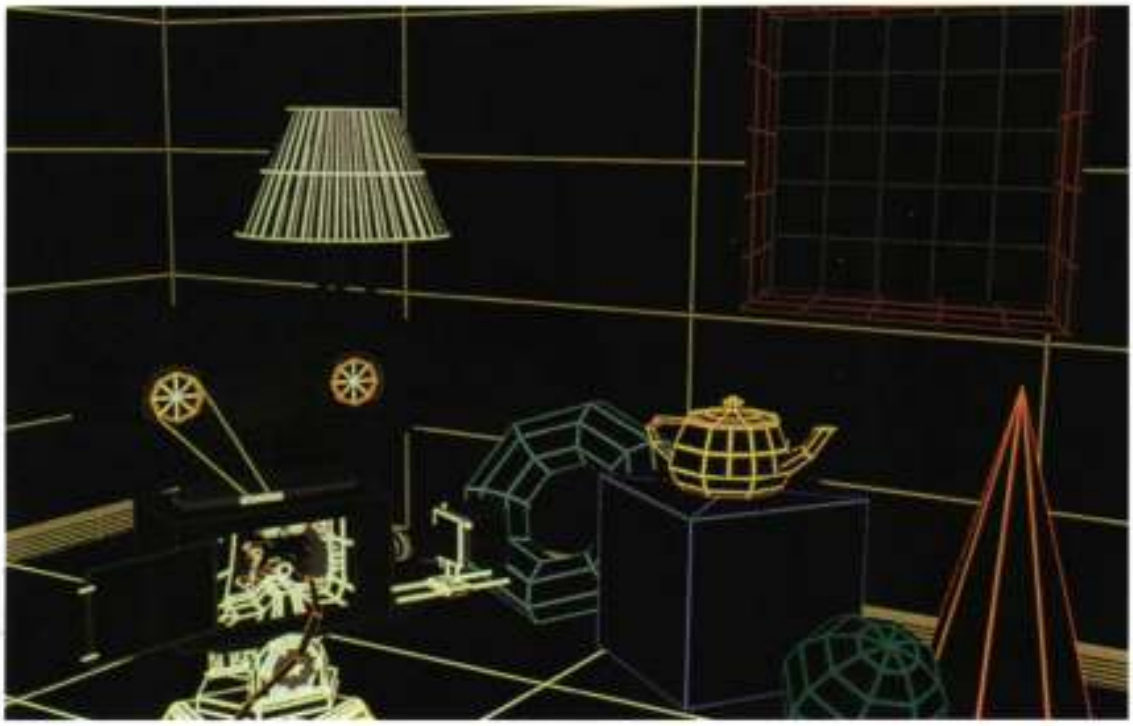




彩图 II-25 摄影爱好者。深度裁剪（14.3.5节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

彩图 II-26 摄影爱好者。彩色的向量（14.3.7节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）





彩图 II-27 摄影爱好者。可见线判定（14.3.8节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

彩图 II-28 摄影爱好者。只带有环境光照的可见面判定（14.4.1节和16.1.1节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

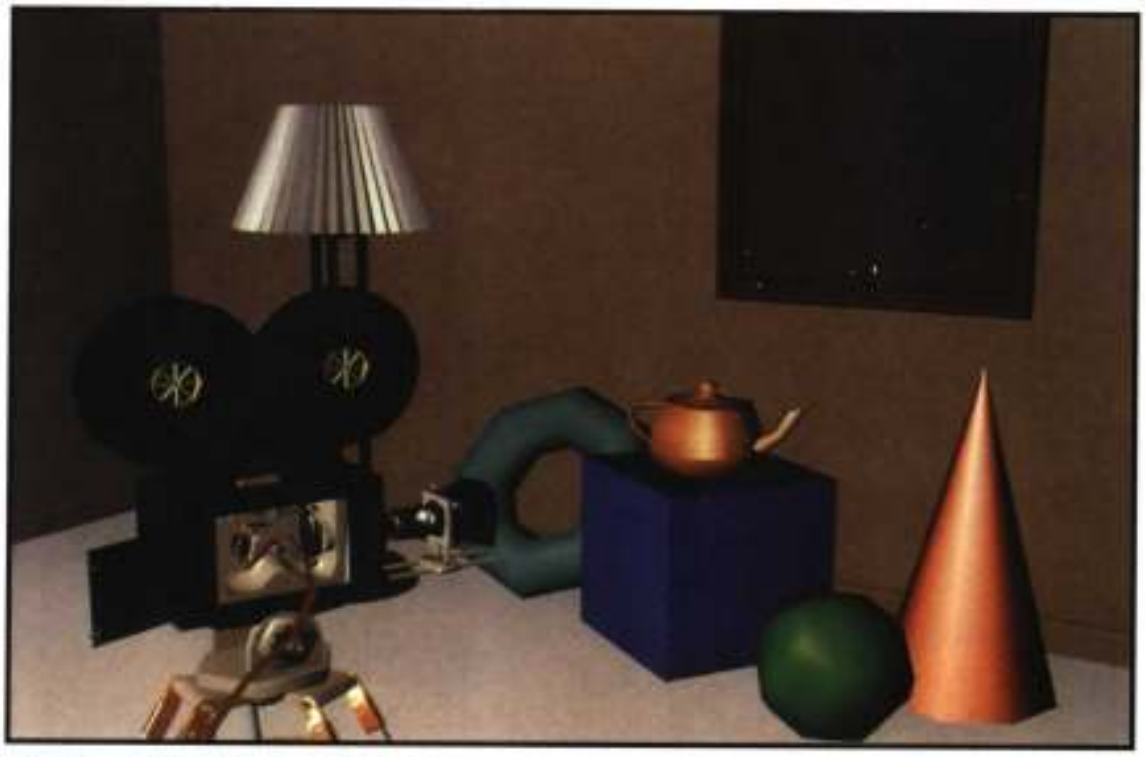




彩图 II-29 摄影爱好者。带有漫反射的独立的经明暗处理的多边形（14.4.2节和16.2.3节）。（Pixar公司版权所有，1990。由Thomas Williams 和H.B.Siegel使用Pixar的PhotoRealistic RenderMan™软件绘制。）

彩图 II-30 摄影爱好者。带有漫反射的经Gouraud明暗处理的多边形（14.4.3节和16.2.4节）。（Pixar公司版权所有，1990。由Thomas Williams 和H.B.Siegel使用Pixar的PhotoRealisticRenderMan™软件绘制。）

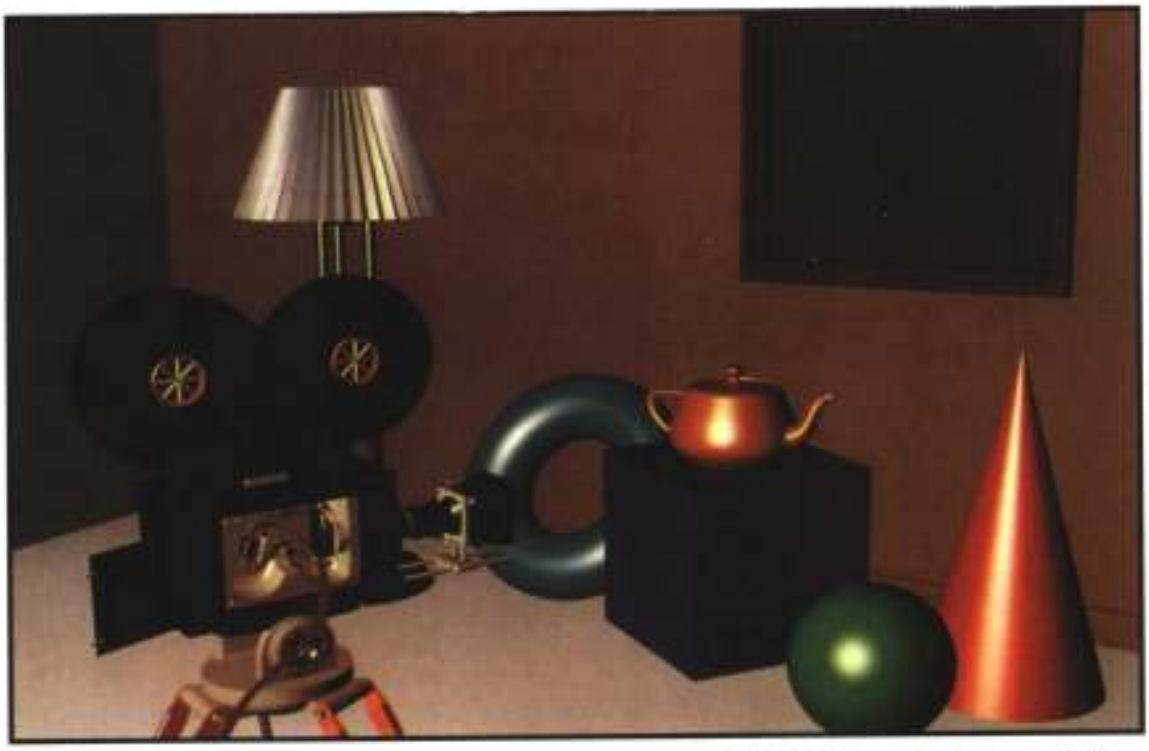




彩图 II-31　摄影爱好者。带有镜面反射的经Gouraud明暗处理的多边形（14.4.4节和16.2.5节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

彩图 II-32　摄影爱好者。带有镜面反射的经Phong明暗处理的多边形（14.4.4节和16.2.5节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）





彩图 II-33 摄影爱好者。带有镜面反射的曲面（14.4.5节）。（Pixar公司版权所有，1990。由Thomas Williams和H.B.Siegel使用Pixar的PhotoRealistic RenderMan™软件绘制。）

彩图 II-34 摄影爱好者。改进的光照模型和多个光源（14.4.6节和16.1节）。（Pixar公司版权所有，1990。由ThomasWilliams和H.B.Siegel使用Pixar的PhotoRealistic RenderMan™软件绘制。）





彩图 II-35 摄影爱好者。纹理映射（14.4.7节、16.3.2节、17.4.2节和17.4.3节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）

彩图 II-36 摄影爱好者。位移映射（14.4.7节和16.3.4节）。（Pixar公司版权所有，1990。 由ThomasWilliams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）





彩图 II-37 摄影爱好者。反射映射（14.4.9节和16.6节）。（Pixar公司版权所有，1990。由Thomas Williams 和 H.B.Siegel使用 Pixar的PhotoRealistic RenderMan™软件绘制。）



a)



b)

彩图 II-38 通过后处理实现的景深（14.4.10节和16.10节）。a) 聚焦在立方体上（550 mm），光圈是f/11。b) 聚焦在球体上（290 mm），光圈是f/11。（经RPI的Michael Potmesil 和 Indranil Chakravarty许可。）



彩图 II-39 通过分布式光线跟踪方法实现的景深（14.4.10节和16.12.4节）。（由Robert Cook、Thomas Porter和Loren Carpenter提供。Pixar公司版权所有，1984。保留所有权利。）

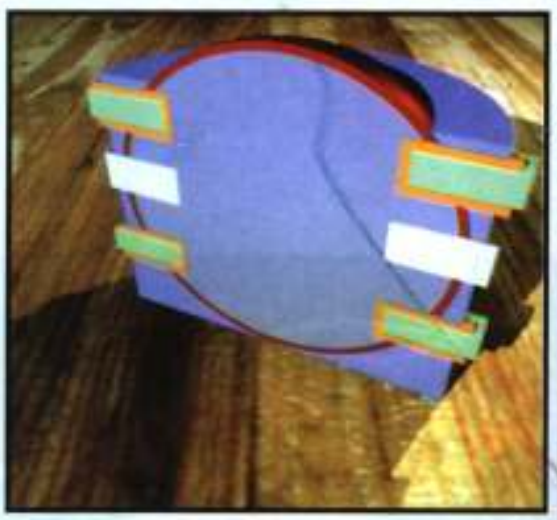


彩图 III-1 草地和树木。共有2×10^6个按照列表和栅格层次组织的对象（15.10.2节）。在一台IBM 4381/Group12的机器上花了16小时，以512×512的分辨率，每个像素16根光线进行光线跟踪。（经John Snyder 和Alan Barr许可，加利福尼亚理工学院计算机图形学组版权所有。）

彩图 III-2 采用CSG（15.10.3节）定义的光线跟踪绘制的带有星星和月亮花纹的碗。a) 在CSG操作中用到的对象；b)a) 中对象的横截面；c) 用CSG操作绘制成的碗。由于和蓝色的圆柱体相交，红色的球体被截，然后又被在b)中的内部灰色球体挖成中空。从上一步的结果减去被挤压的月亮（通过从桔黄色的圆柱中减掉绿色的圆柱得到）和白色的星星，他们在碗上就挖出洞来。运用反射映射（16.6节）和Cook-Torrance光照模型（16.7节）使碗呈现出金属外观。（经哥伦比亚大学的David Kurlander许可。）



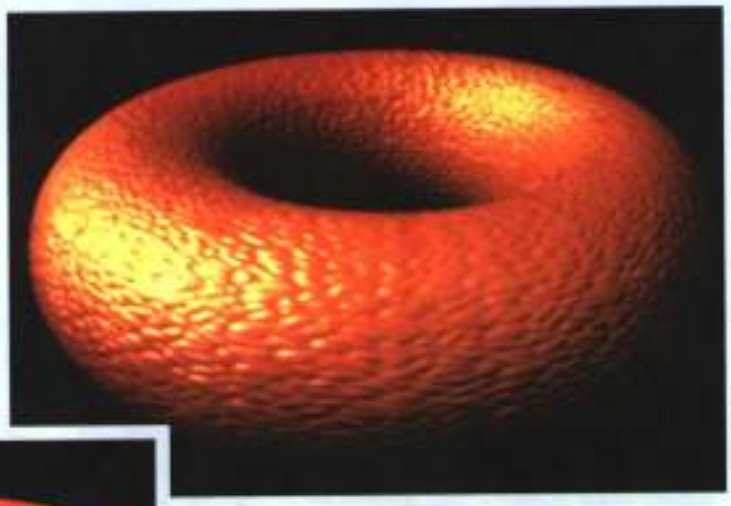
a)



b)

c)







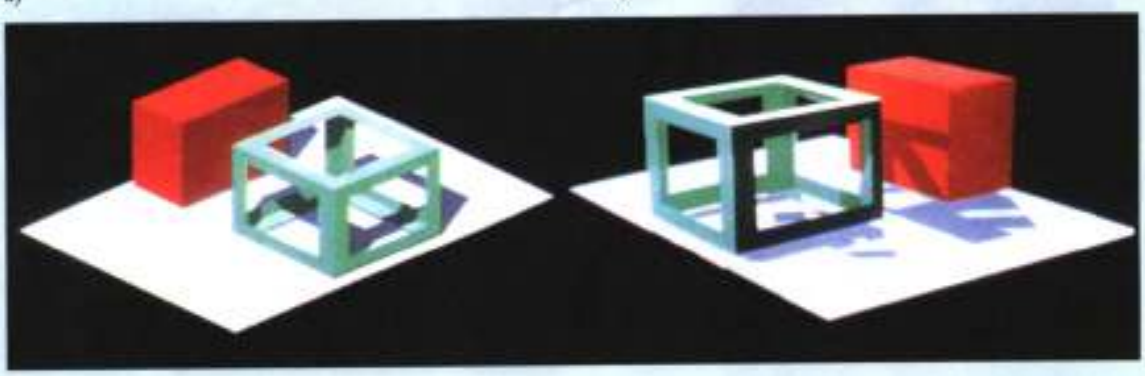
彩图 III-3 用手工生成的凹凸函数映射的一个凹凸圆环面（16.3.3节）。（经犹他大学许可由Jim Blinn提供。）

彩图 III-4 用手工生成的凹凸函数映射的一个凹凸草莓（16.3.3节）。（经犹他大学许可由Jim Blinn提供。）

彩图 III-5 由16.4.2节中两步对象精度算法生成的带有阴影效果的物体。a) 一个光源，b) 两个光源。（康奈尔大学计算机图形学组的Peter Atherton、Kevin Weiler和Donald P.Greenberg提供，1978。）

a)

b)





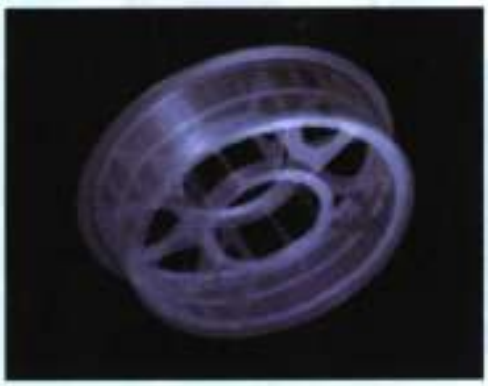
彩图 III-6 由16.4.3节中基于对象精度的阴影体BSP树算法生成的带有阴影效果的房间。a) 带有两个点光源的场景，b) 用黑线表示多边形片段的同一场景，深灰色的面片没有被任何光源照射，浅灰色的片段被一个光源照射，非灰色的部分被两个光源同时照射。（经哥伦比亚大学的Norman Chin许可。）

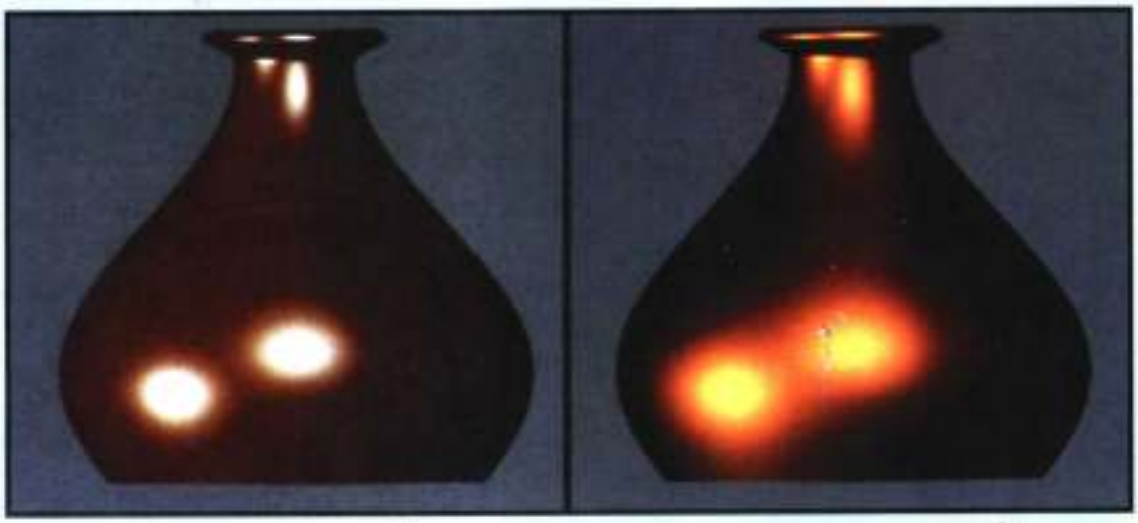
a)

b)



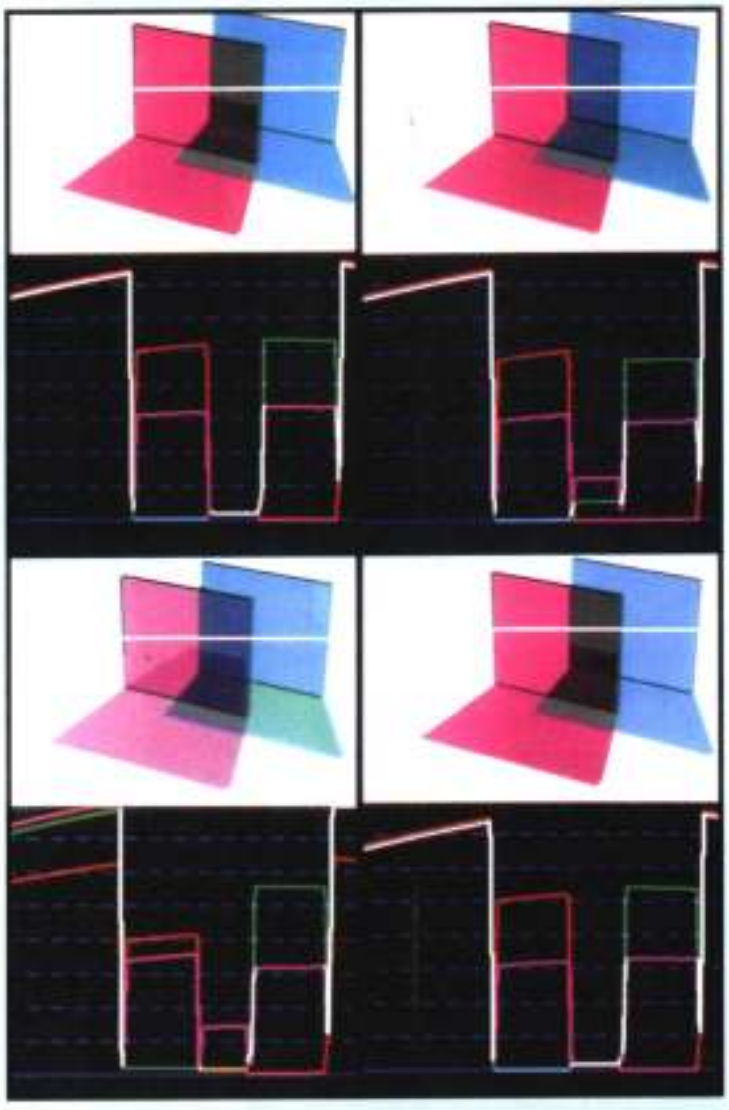
彩图 III-7 以16.5.1节中的扩展z缓存算法绘制的非折射的透明性。Unterlafette数据库经CAM-I（Computer Aided Manufacturing International, Inc., Arlington, TX）许可使用。（由Abraham Mammen在一台Stardent 1000计算机上绘制。）



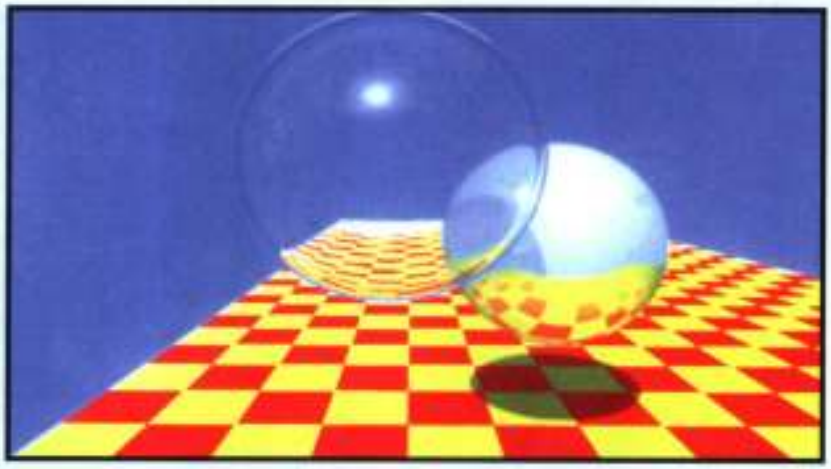


a) b)

彩图 III-8 两个采用Cook-Torrance光照模型（16.7节）绘制的花瓶。两个花瓶都是被两个$I_{l1}=I_{l2}$=CIE标准光源D6500照射，其中$d\omega_{l1}=0.0001$，$d\omega_{l2}=0.0002$；$I_a=0.01I_{l1}$；ρ_e=铜对法线入射方向的双向反射系数；$\rho_s=\pi\rho_{bi}$：a)铜色的塑料：$k_s=0.1$，F=乙烯树脂镜面的反射率；D=Beckmann函数，其中参数$m=0.15$；$k_d=0.9$；b) 铜金属：$k_s=1.0$，F=铜镜面的反射率；D=Beckmann函数，其中参数$m_1=0.4$，$w_1=0.4$，$m_2=0.2$，$w_2=0.6$；$k_d=0.0$。（由康奈尔大学计算机图形学组的Robert Cook提供。）



彩图 III-9 对于两个重叠滤波器（16.9节）的光谱采样技术的比较。这些图显示了那些标记的扫描线的RGB值，用红色(R)，绿色(G)和品红色(B)的线表示。a)在360 nm–830 nm之间每纳米采样一次，b) 3个CIE XYZ样本，c) 3个RGB样本，d) 9个光谱样本。（由康奈尔大学计算机图形学组的Roy A.Hall和Donald P.Greenberg提供，1983。）



彩图 III-10 球体和棋盘。一幅采用递归光线跟踪方法产生的早期图像（16.12节）。（经贝尔实验室的Turner Whitted许可。）



a)



b)

彩图 III-11 光线跟踪产生的图像。a) 短电影《Quest》（1985）中的一个场景。（由Michael Sciulli、James Arvo和Melissa White制作。惠普公司版权所有。）b) "时尚的航空"。用函数修改了几乎所有像素的颜色、表面的法线和透明度。（经哥伦比亚大学的David Kurlander许可，1986。）

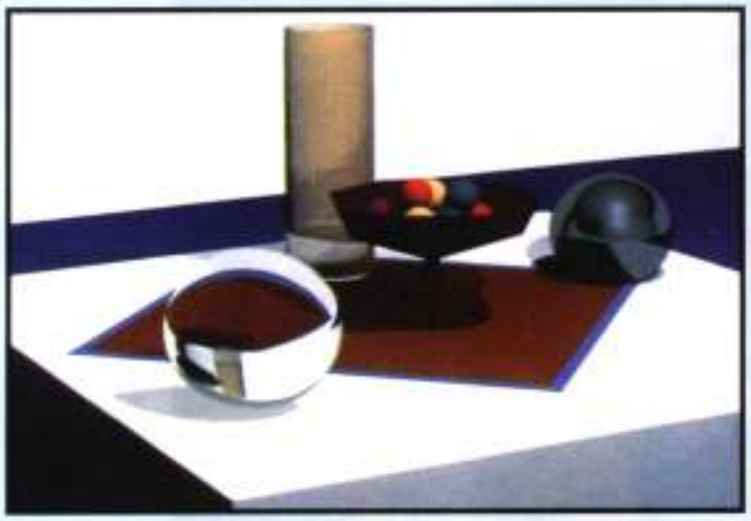
a)



彩图 III-12 用带光缓存的光线跟踪算法（16.12.1节）绘制的图像。在一台VAX 11/780机器上用50×50的光缓存绘制。a) 玻璃质地叶子的树，共有768个对象（6个多边形，256个球体，511个二次曲面）和3个光源。一个分辨率为512×480的版本采用光缓存算法的绘制时间为412分钟，而不采用光缓存算法的绘制时间为1311分钟。b)厨房。共有224个对象（1298个多边形，4个球体，76个圆柱体，35个二次曲面）和5个光源。一个分辨率为436×479的版本采用光缓存算法的绘制时间为246分钟，而不采用光缓存算法的绘制时间为602分钟。（由康奈尔大学计算机图形学组的Eric Haines 和Donald P.Greenberg提供，1986。）

b)



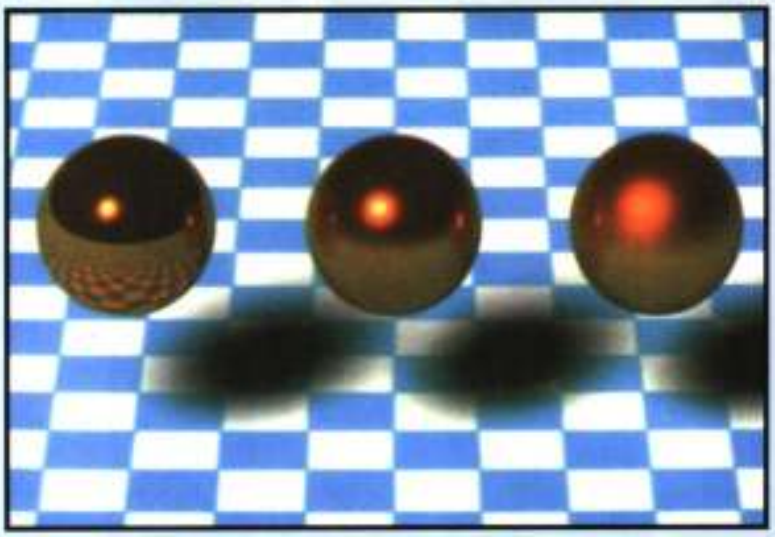


a)

b)



彩图 III-13 几种光线跟踪光照模型的比较（16.12.2节）。请注意盘底的反射率的区别和那些透明且反射的球体的颜色。a)Whitted光照模型，b)Hall光照模型。（由康奈尔大学计算机图形学组的Roy A.Hall和Donald P.Greenberg提供，1983。）



彩图 III-14 锥光线跟踪（16.12.3节）。三个球体（自左至右）通过分别以0.0、0.2、0.4倍半径增加反射光线角度扩展产生钝化的反射效果（16.12.3节），对于从左到右的三个球体分别为0.0，0.2，0.4倍半径。（经多伦多大学的John Amanatides许可。）



彩图 III-15 光束跟踪（16.12.3节）。在一个纹理映射的房间里的一个镜面立方体。（由Paul Heckbert 和Pat Hanrahan提供，NYIT版权所有，1984。）

彩图 III-16 1984。以每个像素16个采样点、用分布式光线跟踪算法（16.12.4节）绘制的4096 × 3550像素大小的图像。请注意那些运动模糊的反射效果和由额外光源引起的半影效果。（由Thomas Porter提供。Pixar公司版权所有，1984。保留所有权利。）





彩图 III-17 采用积分方程方法（16.12.4节）绘制的场景。所有不透明的物体都是朗伯反射的。请注意物体之间的反射。在一台IBM 3081计算机上，按照每个像素40条路径，512×512的分辨率进行计算，共花费了1221分钟。（由加利福尼亚理工学院的J.Kajiya提供。）



a)

b)



彩图 III-18 采用带有物体间漫反射的光线跟踪算法（16.12.4节）绘制的白天的办公室。a)只有直接光照，b)一次漫反射，c)7次漫反射。（经Lawrence Berkeley实验室的Greg Ward许可。）



c)



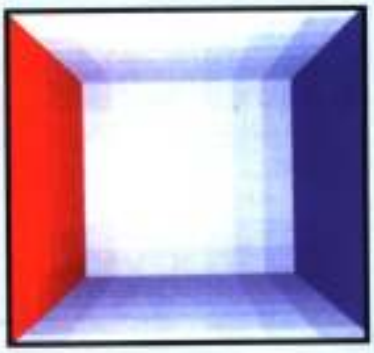
a)



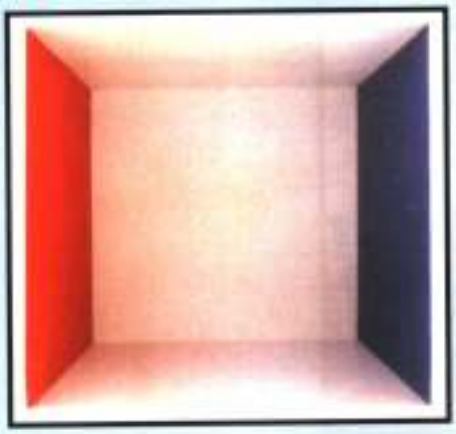
b)

彩图 III-19 会议室。a)真实的会议室照片。b)使用与彩图III-18相同的软件、用光线跟踪绘制的模型画面，但未计算物体间的反射。（经Lawrence Berkeley实验室的Greg Ward、Anat Grynberg和Francis Rubinstein许可。）

彩图 III-20 辐射度（16.13.1节）。带有六个漫射墙壁的立方体（没有显示发白光的前墙）。a)真实立方体的照片，b) 每面墙由49个面片组成、每片用恒定明暗处理来绘制的模型；c)每面墙由49个面片组成、每片用插值的明暗处理来绘制的模型。（由康奈尔大学计算机图形学组的Cindy M.Goral、Kenneth E.Torrance、Donald P.Greenberg和Bennett Battaile提供，1984。）



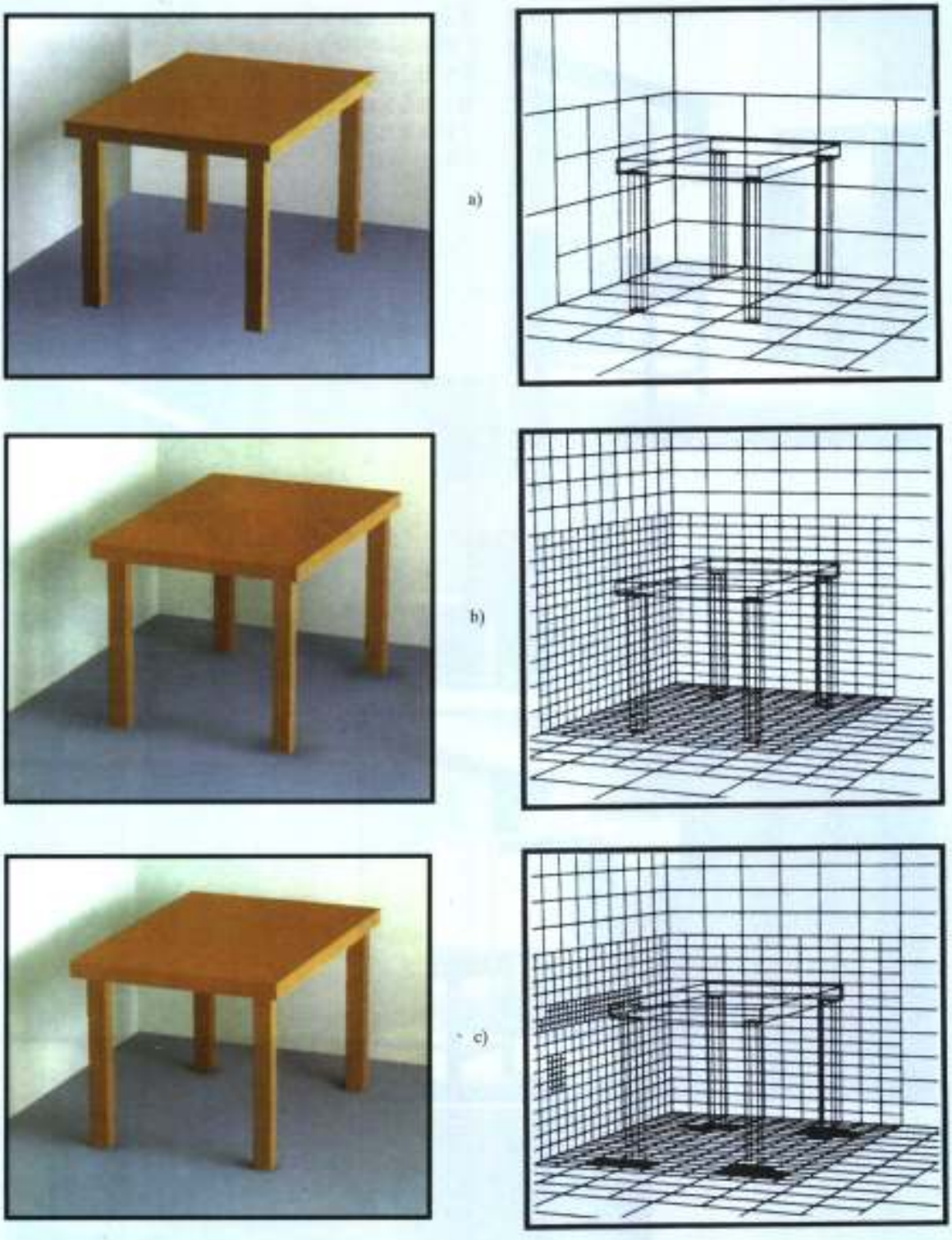
b)



a)



c)



彩图 III-21 辐射度算法，采用半立方体计算形状因子（16.13.2节至16.13.3节）。a)粗糙的面片解决方法（145个面片，10分钟）；b)改进的解决方法，采用更多的面片（1021个面片，175分钟）或者子结构（1021个子面片，62分钟）；c)用自适应划分，对b)中的子结构进一步细分（1306个子面片，多用24分钟）。（由康奈尔大学计算机图形学组的Michael F.Cohen提供，1985。）



a)



b)



c)



d)

彩图 III-22 用不同光照模型（16.13.2节和20.8.2节）和扩展光源的阴影体（16.8节）绘制的机房。a)由面光源窗照明，b)用房顶面光源照明，c)用晴空的光照模型照射，d)由包括大气散射的光照模型照射。（a–c）采用了辐射度算法。（由福山大学的T.Nishita和广岛大学的E.Nakamae提供。）



a)



b)

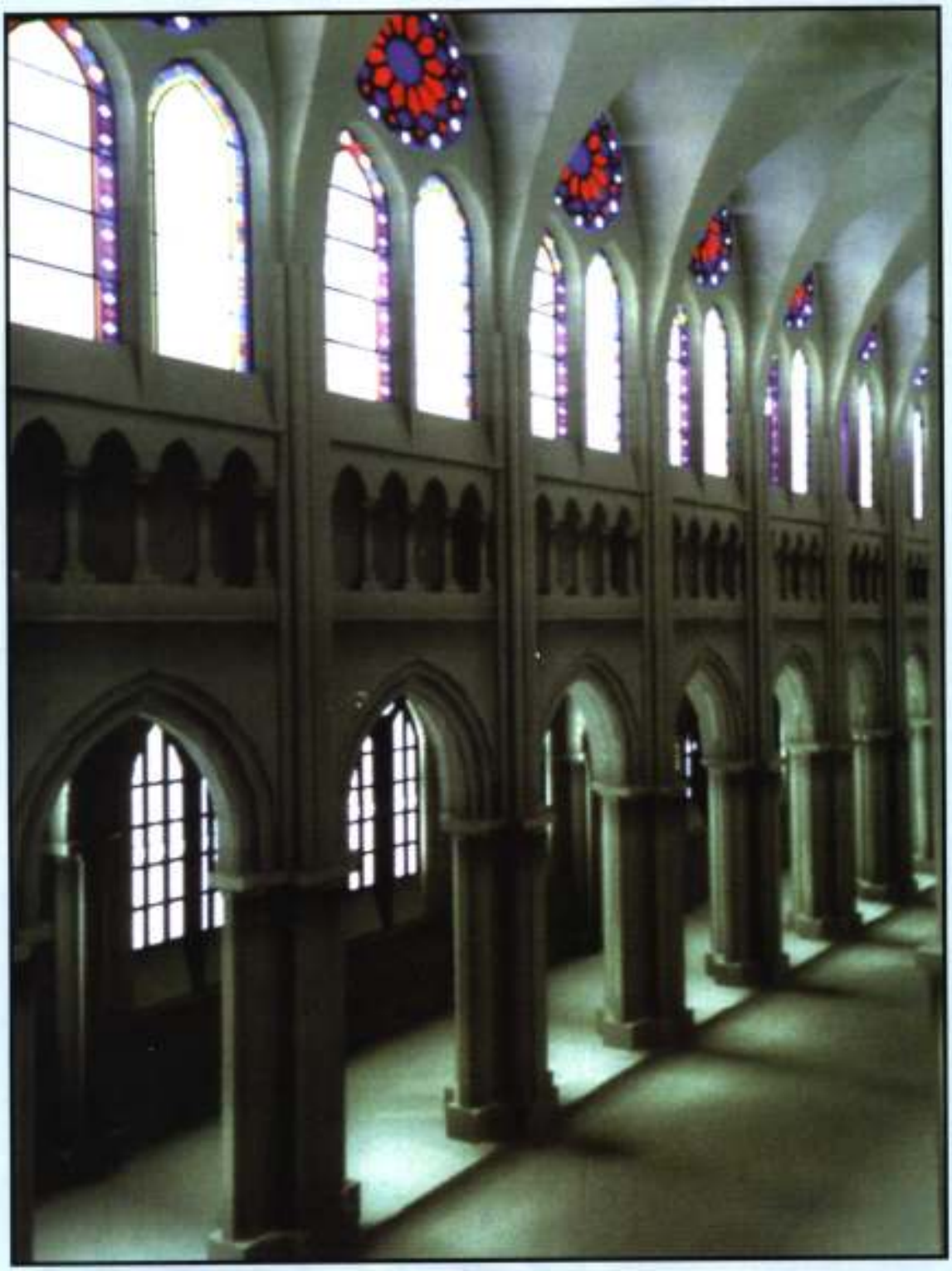


c)

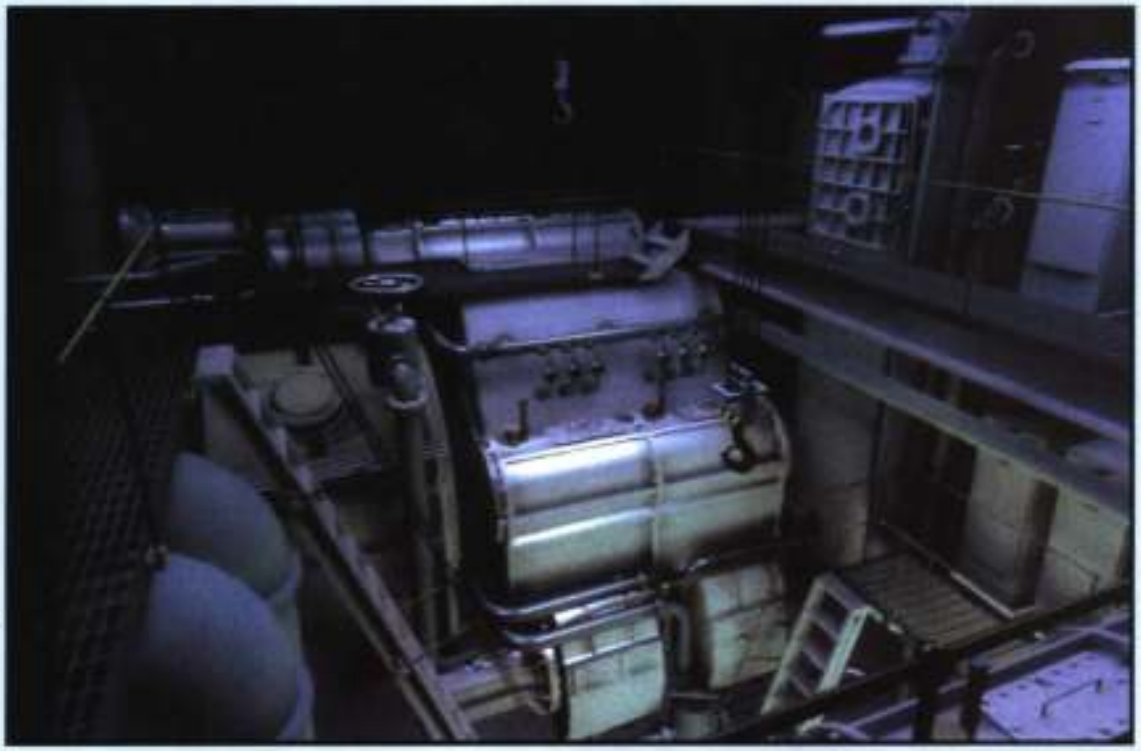


d)

彩图 III-23 采用逐步求精的半立方体辐射度算法（16.13.4节）绘制的办公室：包含500个面片，7000个子面片。加入了估计的泛光辐射度，在一台HP 9000 825 SRX型计算机上，每一次迭代的计算和显示大约需要15秒。a)1次迭代，b)2次迭代，c)24次迭代，d)100次迭代。（由康奈尔大学计算机图形学组的Shenchang Eric Chen、Michael F.Cohen、John R.Wallace和Donald P.Greenberg提供，1988。）



彩图 III-24 采用逐步求精的辐射度算法绘制的Chartres大教堂的中殿，采用光线跟踪计算其形状因子（16.13.5节）。其中的两个拱状壁凹含有9916个被处理并拷贝了三次以上的多边形。在一台HP 9000 835 TurboSRX型的计算机上进行60次迭代计算花费了59分钟。（由John Wallace和John Lin采用惠普的Starbase Radiosity and Ray Tracing软件绘制。惠普公司版权所有，1989。）



彩图 III-25 采用逐步求精的辐射度算法绘制的锅炉间，使用光线跟踪计算形状因子。（由John Wallace、John Lin和Eric Haines采用惠普的Starbase Radiosity and Ray Tracing软件绘制。惠普公司版权所有，1989。）

彩图 III-26 镜面辐射度算法（16.13.6节）；包含64个镜面面片和237个漫射面片。（由康奈尔大学计算机图形学组的David S.Immel、Michael F.Cohen和Donald P.Greenberg提供，1986。）





彩图 III-27 辐射度方法和光线跟踪方法的组合（16.13.7节）。a)漫射辐射度算法，b)第一遍用漫射方法，第二遍用光线跟踪，c)第一遍用带扩展形状因子的漫射方法，第二遍用光线跟踪。（由法国巴黎Ecole Normale Superieure的François Sillion和Liens提供。）

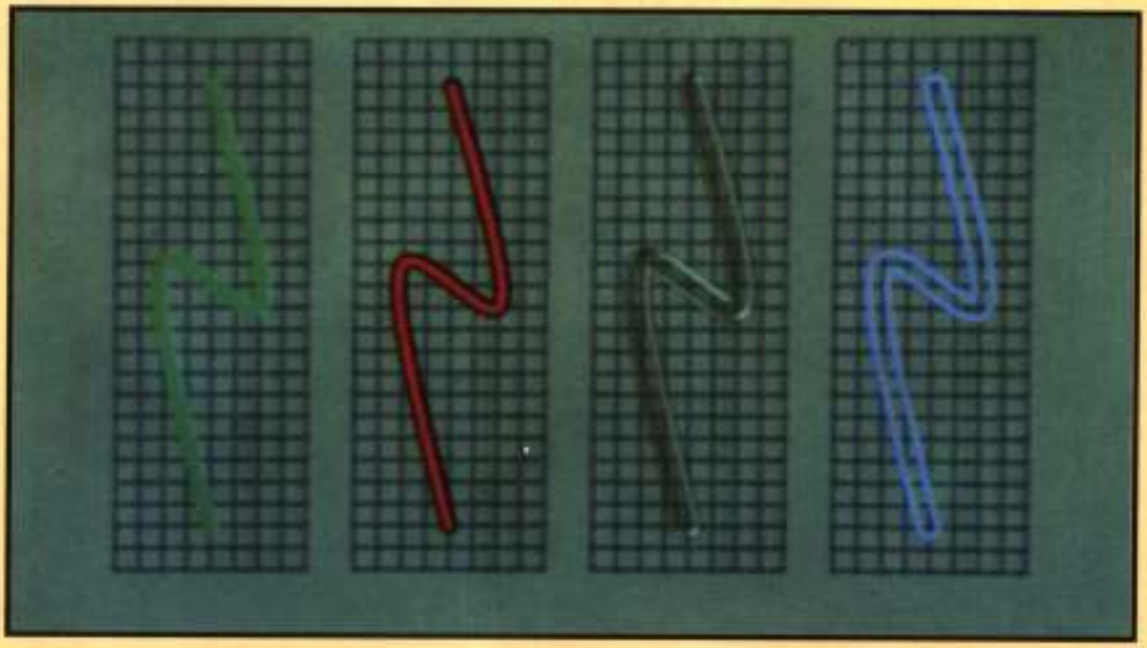


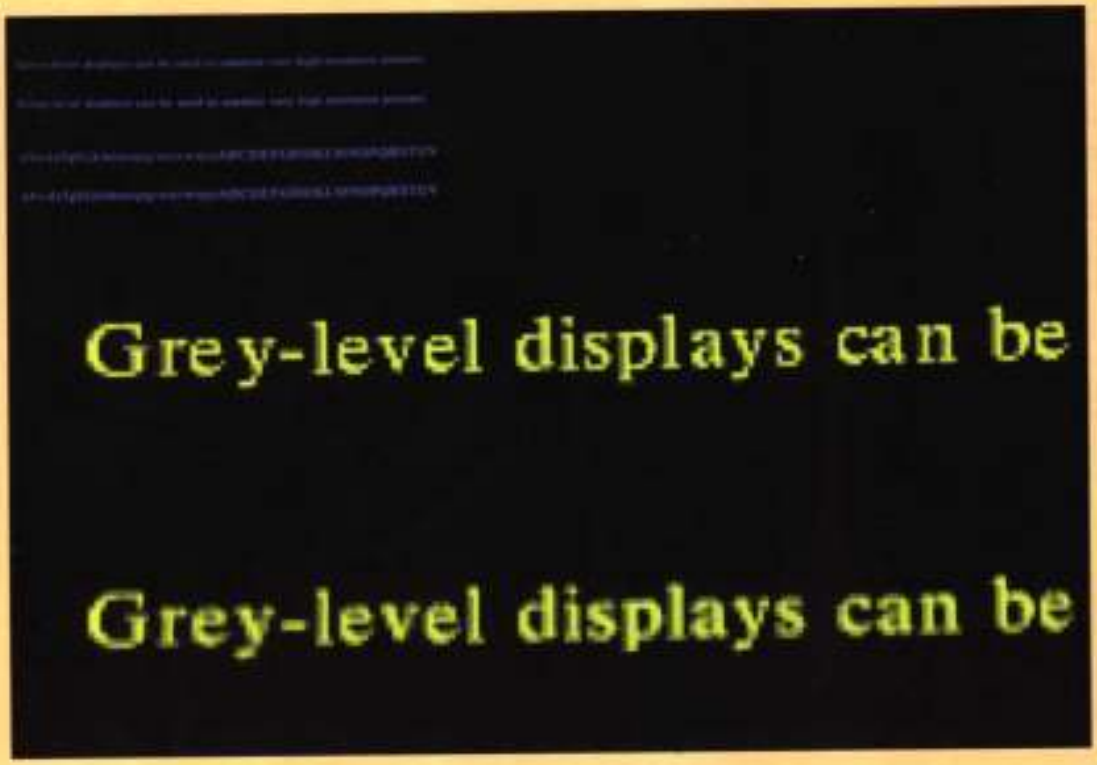
彩图 III-28 采用混合的辐射度和光线跟踪算法绘制的房间。（由法国巴黎的Ecole Normale Superieure的François Sillion和Liens提供。）



彩图 IV-1　一幅使用迭代函数系统（IFS）生成的图像。该系统包含不超过120个的仿射图。（经Michael Barnsley、Arnaud Jacquin、François Malassenet、Laurie Reuter和Alan Sloan许可。）

彩图 IV-2　使用一个反走样的画刷画出的笔划。（经贝尔实验室的Turner Whitted许可。）

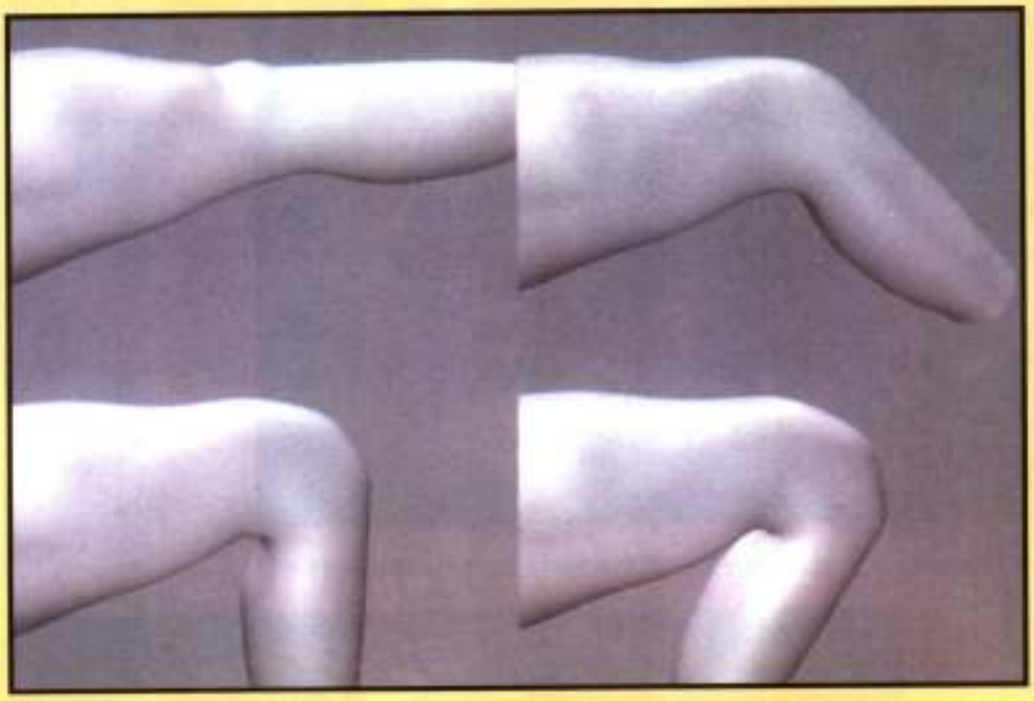




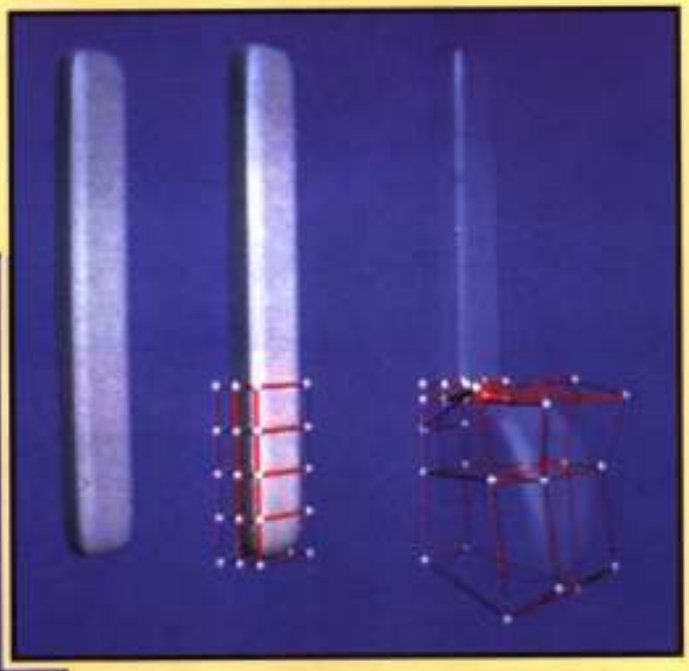
彩图 IV-3 使用YODA显示器显示的反走样的文本。（经IBM T.J.Watson研究中心的Satish Gupta许可。）

彩图 IV-4a) 采用层次样条建模的龙。（彩图a)和彩图b)经滑铁卢大学计算机图形学实验室的David Forsey许可。）





彩图 IV-4b) 通过定义层次样条相对于骨架模型的相对位移建模的皮肤。



彩图 IV-5 一个表面的底部放在一个控制点可以调节的包围盒里。在包围盒里的表面随着包围盒的拖动而形成新的形状。（经Thomas Sederberg和Scott Parry许可。）



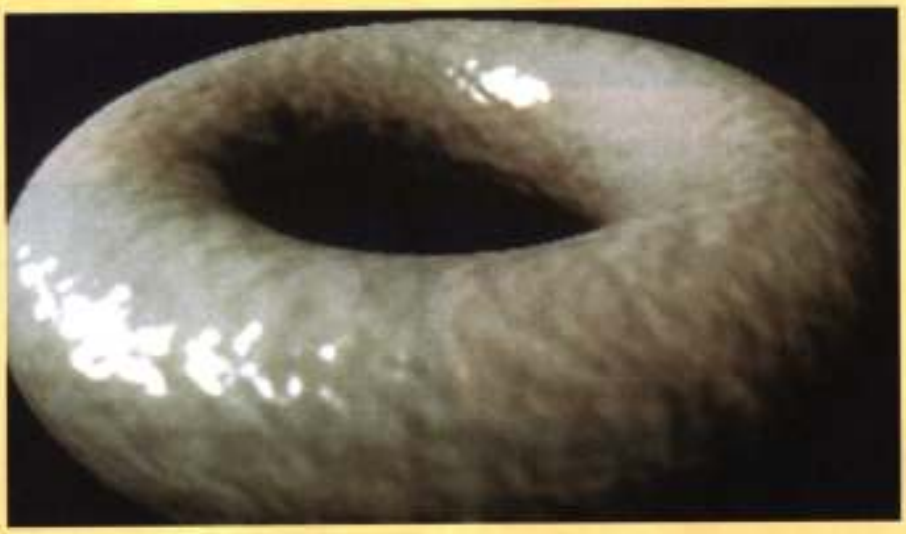
彩图 IV-6 采用任意变形建模的锤子。（经Thomas Sederberg和Alan Zundel许可。）

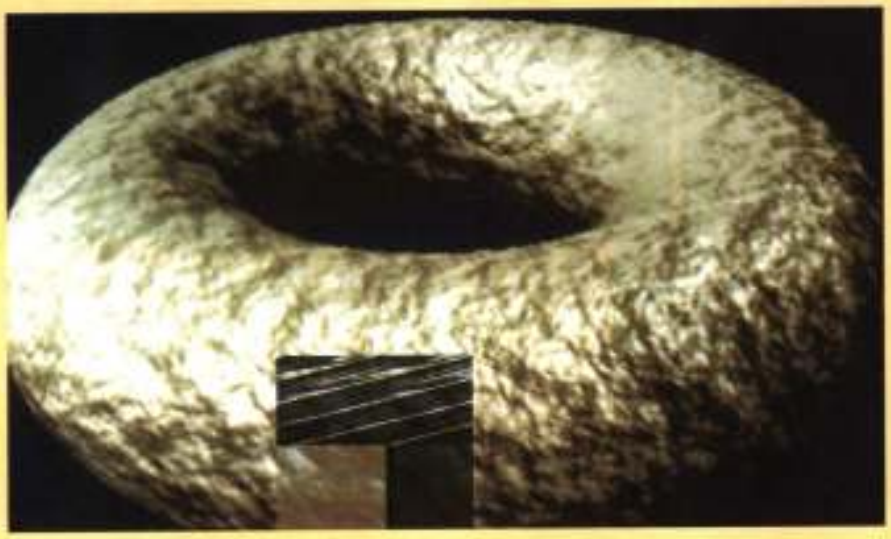


a)

彩图 IV-7 实体纹理（a–d）。灰泥材质的圆环看起来特别有效。（经Ken Perlin许可。）

b)

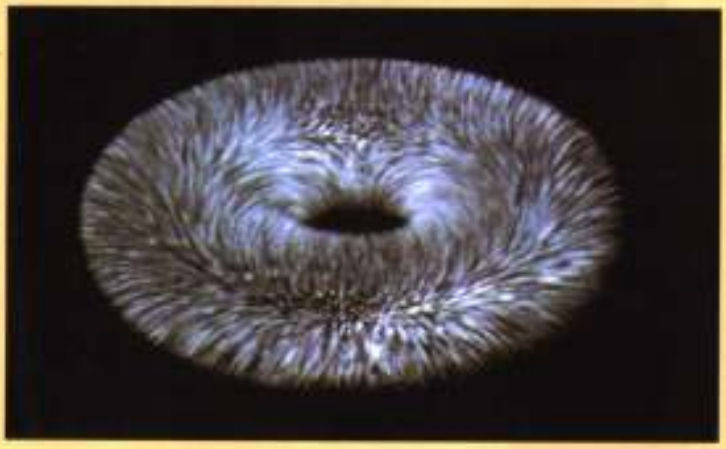




c)

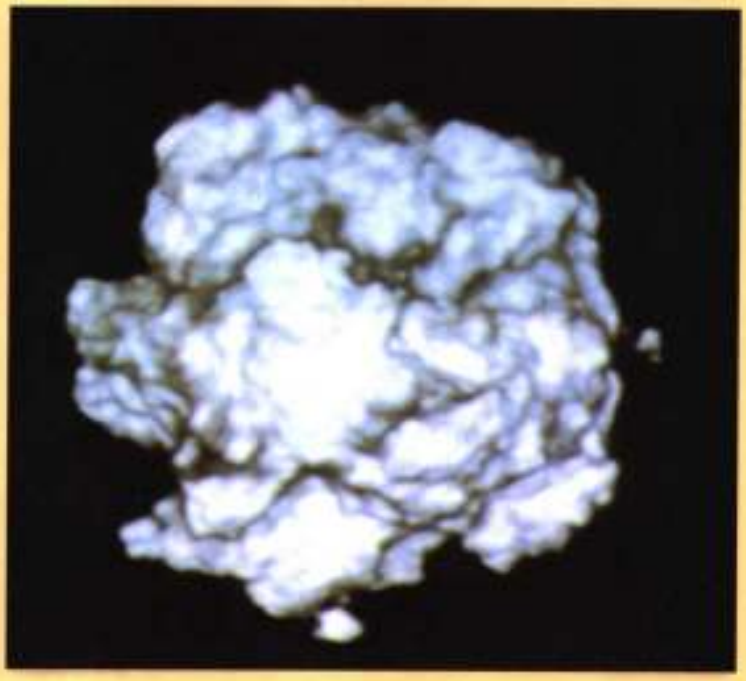
d)



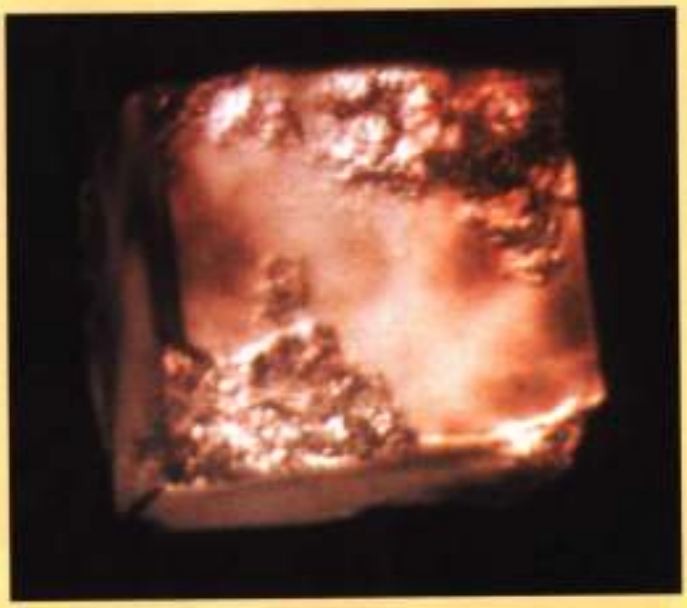


a)

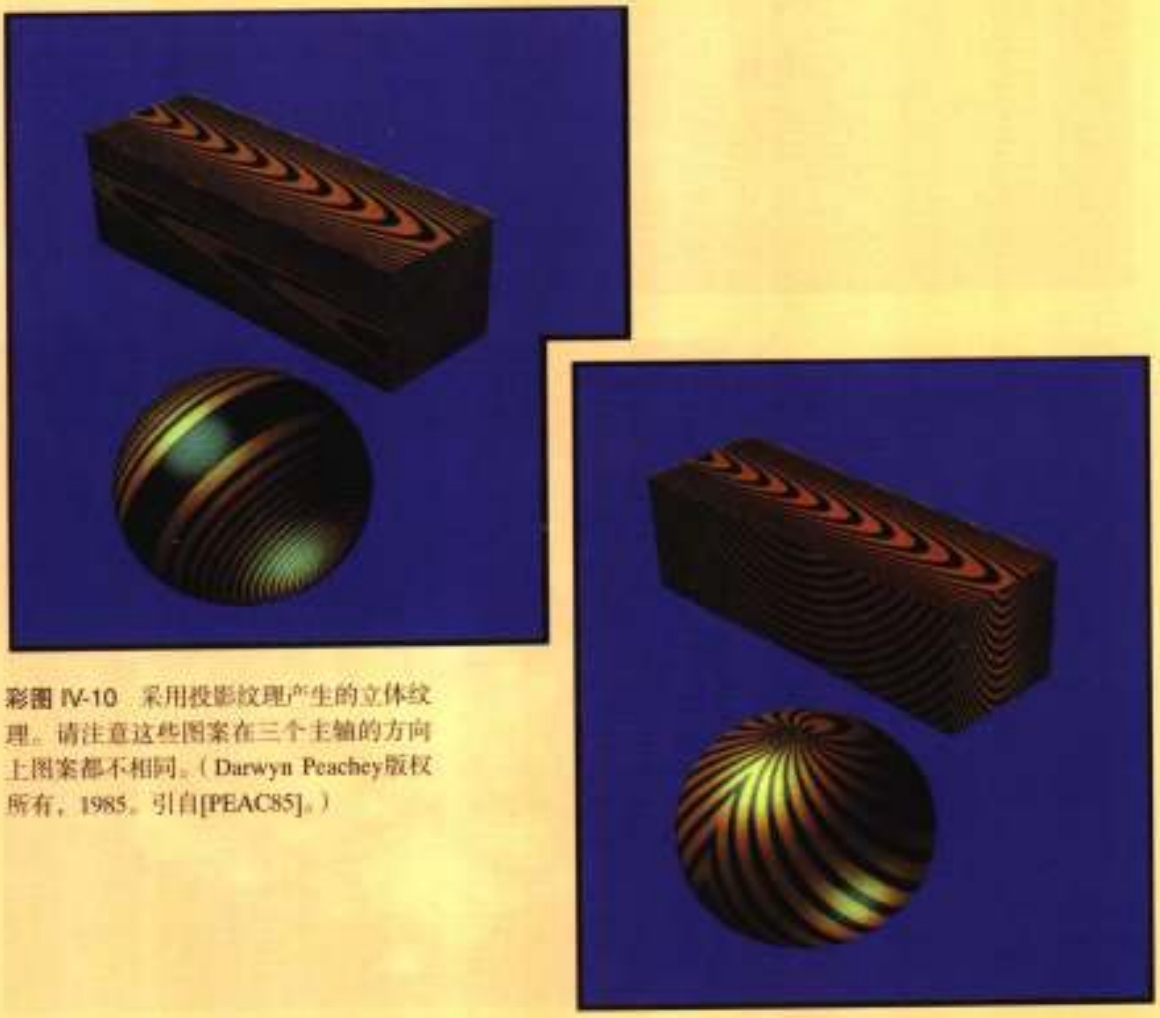
b)



彩图 IV-8　a)采用超纹理建模的一个多毛的圆环，b)一个超纹理生成的团状物。(经Ken Perlin许可。)



彩图 IV-9 一个超纹理的立方体。它表现了纹理是如何影响其表面的颜色、法向及几何形状的。（经Ken Perlin许可。）



彩图 IV-10 采用投影纹理产生的立体纹理。请注意这些图案在三个主轴的方向上图案都不相同。（Darwyn Peachey版权所有，1985。引自[PEAC85]。）



彩图 IV-11 "Vol Libre 山脊"：采用Fournier-Fussell-Carpenter算法产生分形山。（经Loren Carpenter许可。）



彩图 IV-12 采用概率文法建模的简单的树模型。a)一棵棕榈树，b)和c)冷杉树。（由AMAP公司提供。AMAP公司版权所有。）

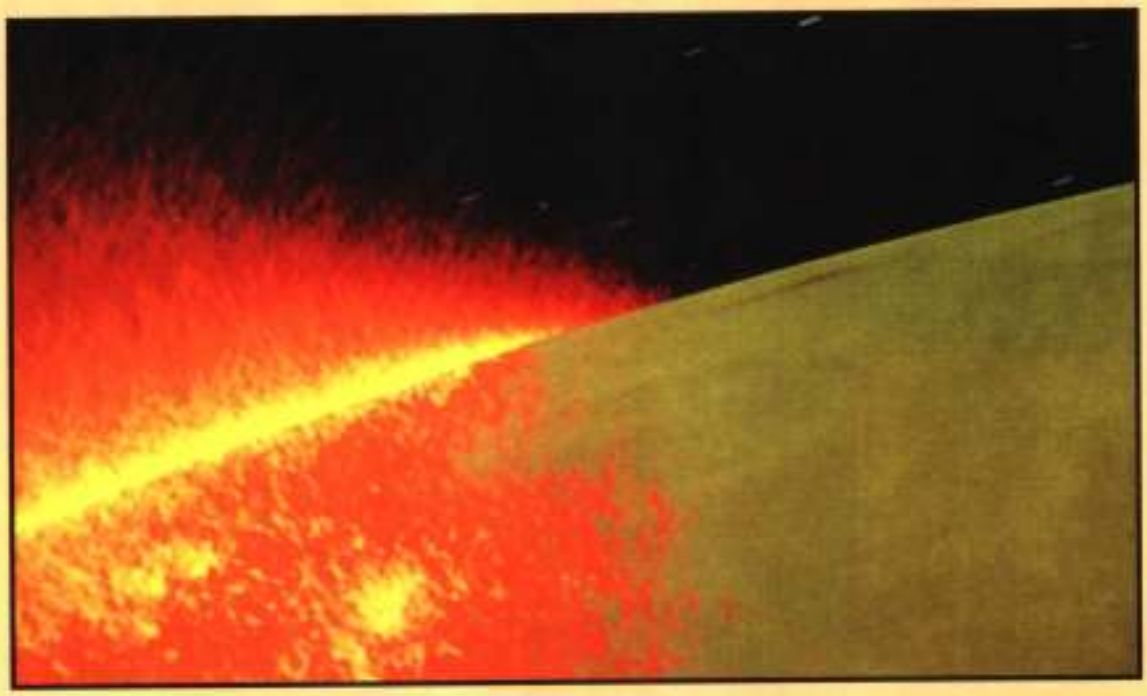


彩图 IV-13　使用与彩图IV-12相同的技术但是不同参数建模的一些更复杂的树。a)一棵柳树，b)一棵春天的果树。（经Atelier de Modilisation et d'Architecture des Plantes许可，AMAP公司版权所有。）

a)

b)

彩图 IV-14　选自“星际旅行II：*The Wrath of Khan*”。由粒子系统动画所实现的“起源”爆炸中的一帧。该爆炸的结果扩展为吞没行星的一堵火墙。（经Pixar公司的Ralph Guggenheim许可。Paramount Pictures公司版权所有，1982，保留所有权利。）

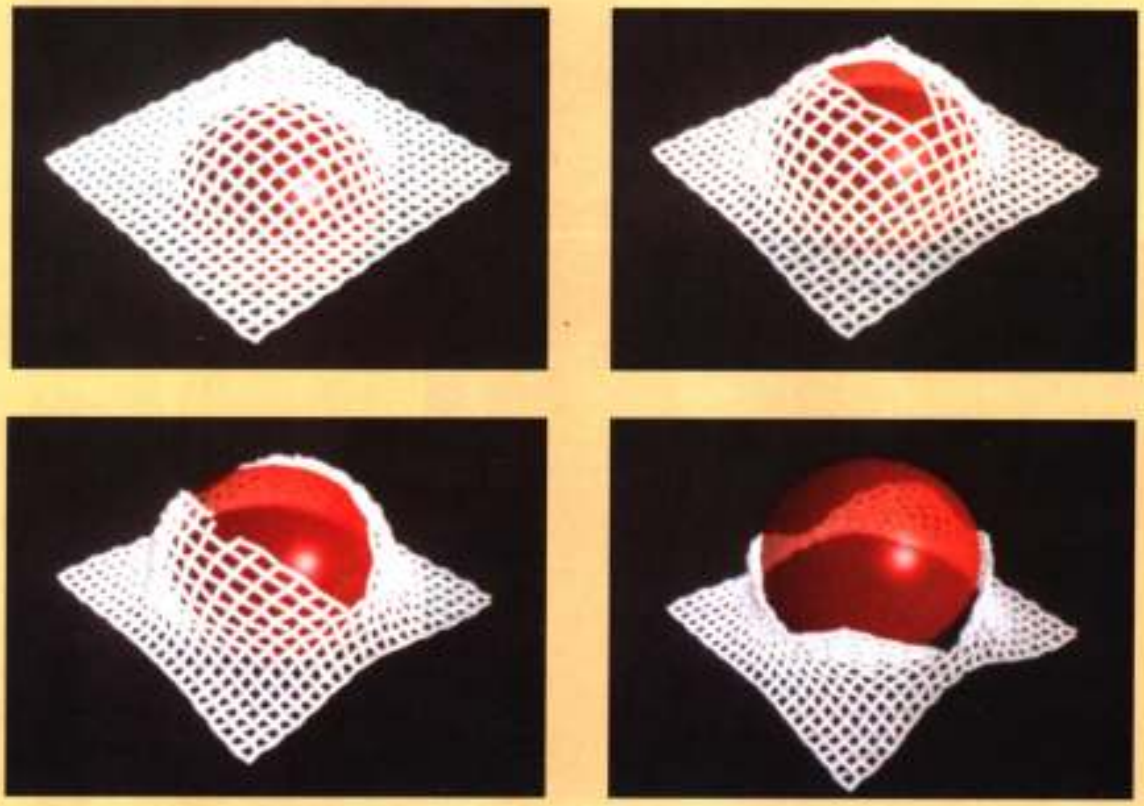


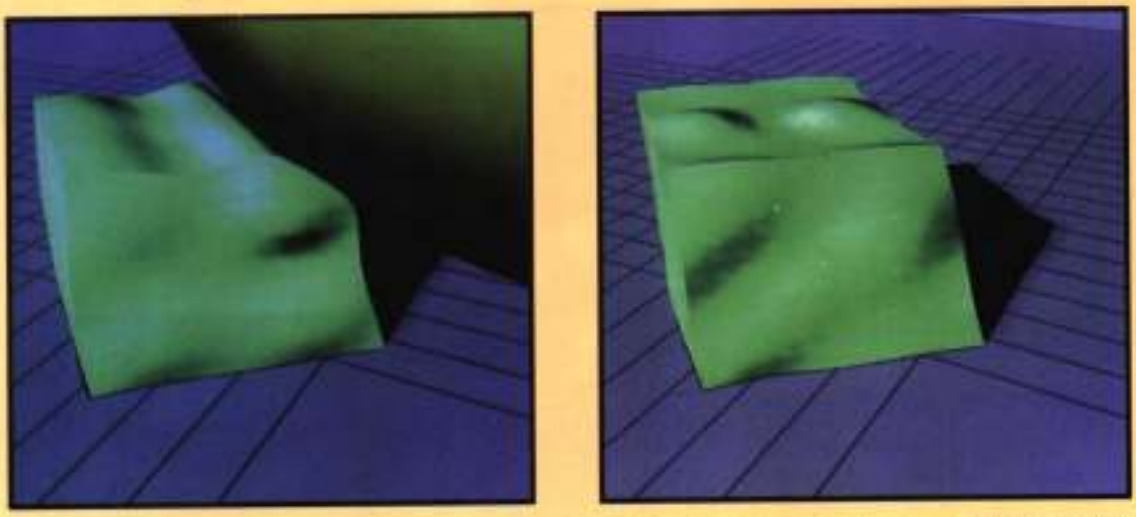




彩图 IV-15　a) 由5个点吊起的一块布，b)多块布。（经AT&T 贝尔实验室的Jerry Weil许可。）

彩图 IV-16　落到球状障碍物上的一个网，逐渐破裂。（经Schlumberger的Demetri Terzopoulos和Kurt Fleischer许可。）

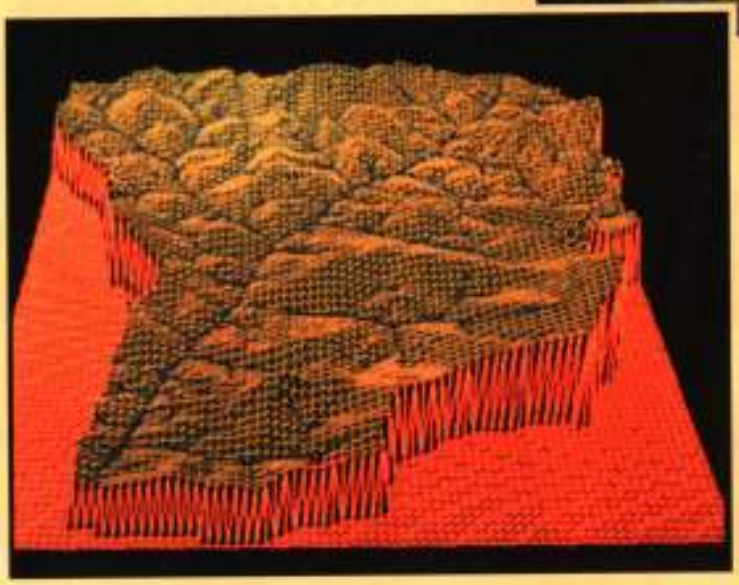




彩图 IV-17 一个弹性模型被一个巨大的球挤压，然后恢复到静止的形状。（经加州理工学院计算机图形学组的John Platt和Alan Barr许可。）

彩图 IV-18 "Shreve山谷"。这个地形是由包含两个斜墙组成的简单初始地形得到的。这些斜墙逐渐形成为倾斜的山谷（图中的那些主要河流）。（经亚利桑那州立大学的G.Nielson许可。）

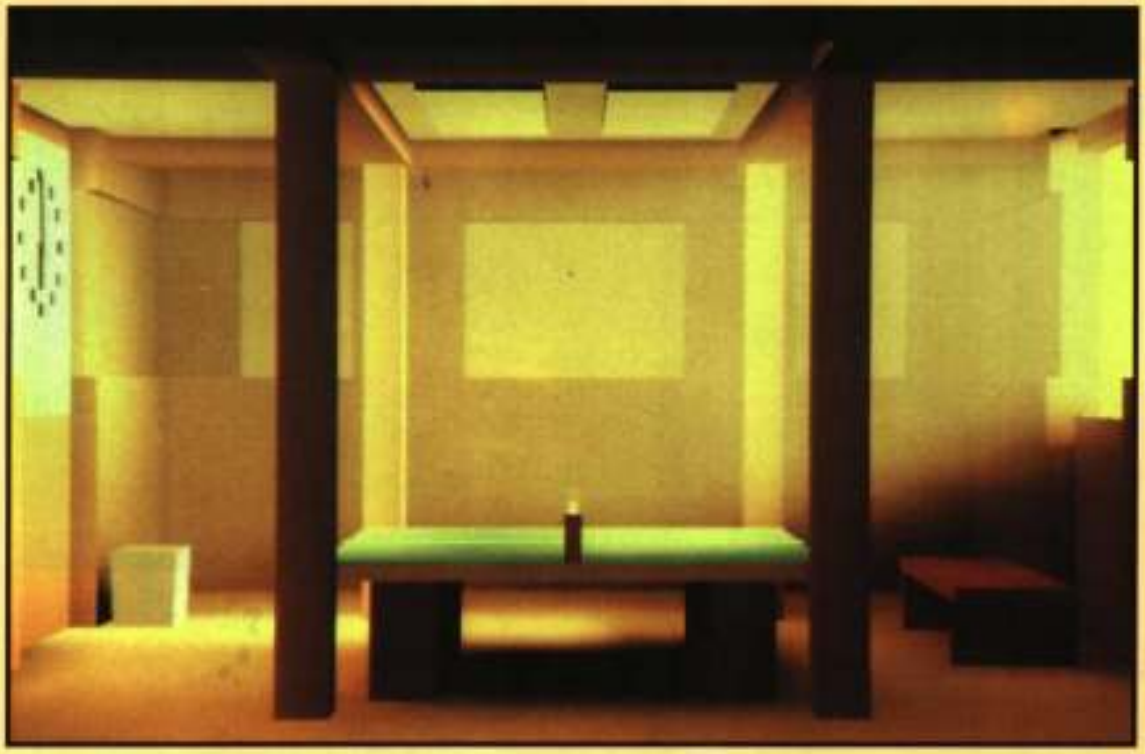






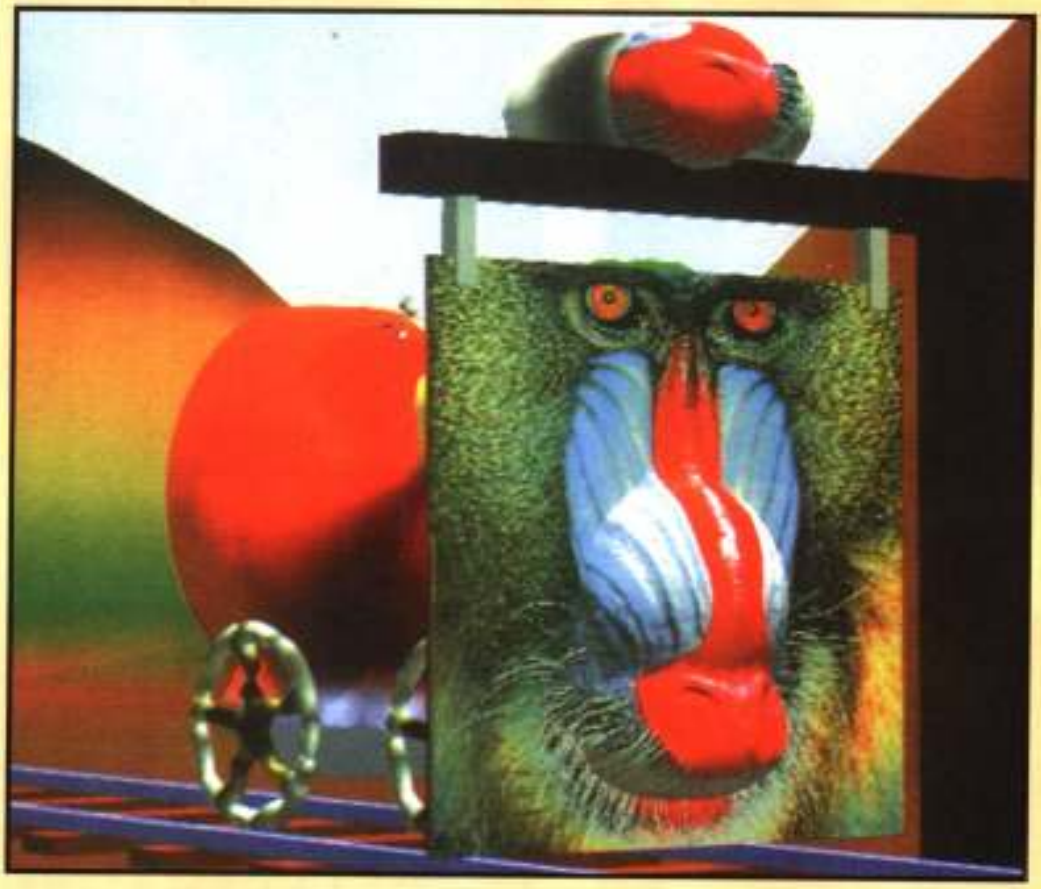
彩图 IV-19　日落的海滩。（经多伦多大学的Bill Reeves、Pixar和Alain Fournier许可。）

彩图 IV-20　有散射介质的房间的傍晚情景。（经康奈尔大学计算机图形学组的Holly Rushmeier许可。）





彩图 IV-21　以固体纹理建模的大理石花瓶。（经Ken Perlin许可。）



▲

彩图 IV-22 采用柔软物体建模的火车。（摘自卡尔加里大学的Brian Wyvill和Angus Davis编辑的《*Great Train Rubbery*》中的“Entering Mandrill Space”。）



彩图 IV-23 这条蛇由Alias Research公司为1988年9月的《*IEEE Computer Graphics and Applications*》的封面而制作。这条蛇采用ALIAS系统进行建模，使用了一幅彩色纹理图绘制蛇身上的斑纹，一幅带有凸凹形状的图绘制蛇身上的鳞。（由Alias的Gavin Miller和Robert LeBlanc制作。）



a)



b)



c)



d)



e)



f)

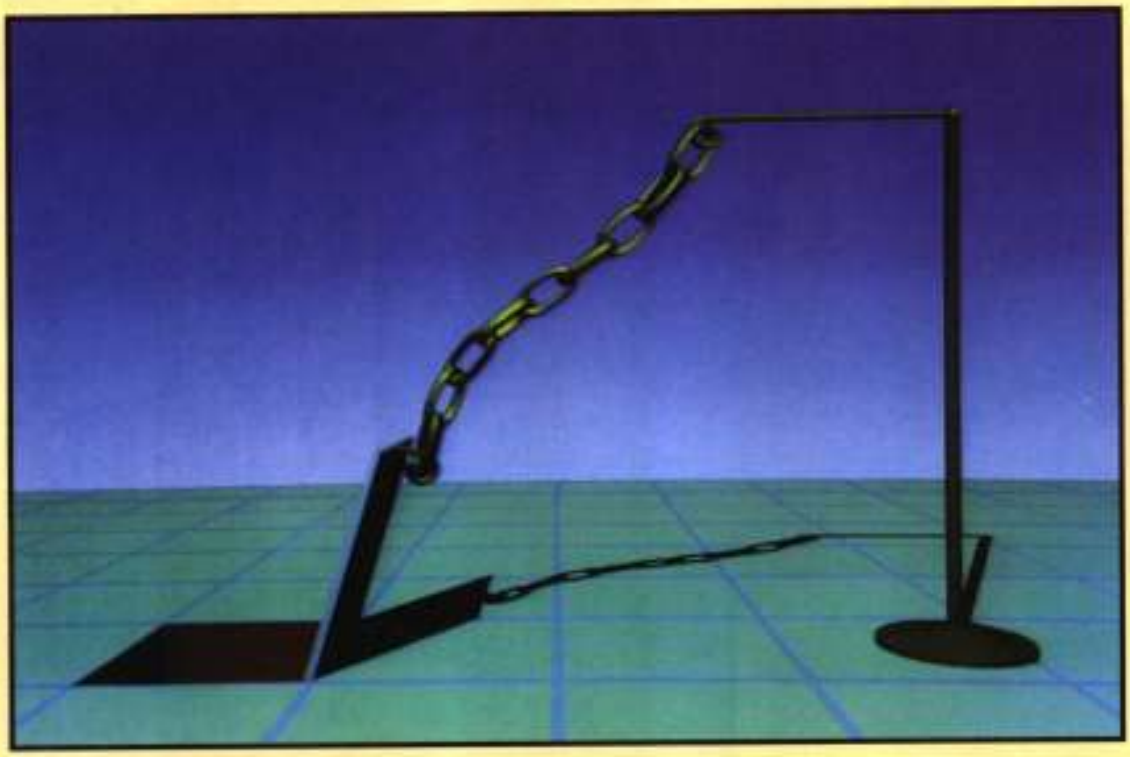
彩图 IV-24　为电影《Young Sherlock Holmes》创作的玻璃骑士。a)用颜色图绘制的护肩盔甲的基本形状，b)使用一幅环境图绘制的护肩盔甲，c)采用带有凸凹图和光照函数来对上面的环境图进行修改，d)加入了污点和小气泡，e)加入另一幅颜色图来产生接缝和铆钉，f)这块护肩盔甲的细节，g)整幅图像；护肩盔甲在图的右上侧。(Paramount Pictures公司版权所有，1989。所有权利保留。经Industrial Light & Magic公司许可。)



g)



彩图 IV-25　这些场景中的树是采用一种自动定位的方法来放置的。这些物体是用带有纹理的二次曲面和分形模型产生的。(经Grumman Data Systems的G.Y.Gardner许可。)



彩图 IV-26 采用动态约束的方法建模的一个自装配系统。（经加州理工学院计算机图形学组的Ronen Barzel和Alan Barr许可。）

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周立柱	周克定	周傲英	孟小峰	岳丽华
范　明	郑国梁	施伯乐	钟玉琢	唐世渭
袁崇义	高传善	梅　宏	程　旭	程时端
谢希仁	裘宗燕	戴　葵		

秘　书　组

武卫东　　温莉芳　　刘　江　　杨海玲

译 者 序

计算机图形学已经成为信息技术（IT）产业非常重要的一个理论基础和专业领域。一本内容更新、材料丰富、阐述深入的计算机图形学著作（中文版），对IT迅速发展的我国是十分需要的。

本书的作者Foley、van Dam等是国际图形学界的著名学者、学术带头人。本书是计算机图形学领域的经典著作。从1982年第1版出版以来，它一直是各国大学“计算机图形学”课程的主要教科书。第1版的中译本对我国计算机图形学的教学科研已起了积极的作用，而本书的第2版C语言版本对第1版做了全面的更新，增加了大量新内容，覆盖了日趋成熟的计算机图形学领域的各个方面，包括二维、三维图形学的数学基础，大量重要算法，光栅图形硬件和软件，交互技术及用户界面软件，真实感图形学，高级建模技术（分形、体绘制等），图像处理和存储，动画等。它是图形应用的广大用户、软件开发人员和硬件设计人员的优秀自学参考书。

本书包括了大量习题和经典参考文献目录，提供了大量C语言的算法实现程序，是本科生、研究生学习相关课程和开展研究工作的优秀参考书和教材。

以上所述也正是本书20多位译者在非常繁忙的工作中花很大的精力翻译这本著作贡献给大学的原因。

参加本书翻译、审校修改工作的有以下同行：许忠信译前言、序言、彩图及第4章，柴建云译第1章，胡事民译第2、7章，王文成译第3章、附录，杜威译第5章，袁晓君译第6章，倪明田译第8、13、17章，汪国平译9、10、11、12、21章并审第16章，唐龙译第14章，刘列明译第15章，刘学慧译第16章，任继成译第18章，李桂清、王洪斌译第19章，李现民、石睿译第20章，唐泽圣审第1、2、4、7、14章，董士海审8~13、16、17、21章并译审彩图，李华审第5、6、18~20章，吴恩华审第3、15章及附录，齐东旭译审索引。

衷心感谢责任编辑杨海玲所做的大量细致的编辑加工工作。没有她的辛勤劳动，是不可能有本书现在的顺利出版的。我们还要感谢为本书翻译出版作出贡献的所有同事。

由于量大、译者多及水平有限，译文中欠妥和纰漏之处恐难避免，恳请读者不吝赐教和指正。

前　言

交互式图形学的时代已经到来。就在不久以前这还是一项需要昂贵显示器硬件、大量计算机资源和独特软件的深奥的专业领域。然而在过去的几年中，随着硬件性能价格比的大幅提升（如配有标准图形终端的个人计算机）以及高端的与设备无关的图形程序包的开发，图形编程已经变得简捷、合理。交互式图形学如今已经可以为我们提供图形通信手段，并成为人机交互的主要工具。（节选自《交互式计算机图形学基础》的前言。*Fundamentals of Interactive Computer Graphics*, James Foley, Andries van Dam, 1982）

这一断言在Apple公司的Macintosh机、IBM的PC机及其他类似产品引发计算机文化革命之前就做出了。现在，就算是没上学的孩子都对交互图形技术非常熟悉，比如窗口控制、用鼠标选取菜单和图标。图形用户界面可以使新手迅速变得老练，没有图形界面的计算机已经越来越罕见了。

随着交互式图形学在用户界面和数据可视化方面的广泛应用，三维物体的绘制技术也变得越来越真实，运用这些技术生成的广告片和电影特技无所不在。20世纪80年代初期尚处于实验阶段的技术目前已变得很普通，而更加令人叹为观止的“照片真实感”技术也马上就要到来。曾经需要花上几个小时才能生成一帧的伪真实感动画，如今在个人计算机上可以以10帧/秒以上的速度生成。1981年的“实时”矢量显示器可以显示没有经过隐藏线消除的几万个矢量组成的移动线框物体；而1990年的实时光栅显示器不仅可以显示同样的线框物体，而且可以显示由十万多个三角形面片组成的、采用Gourand或者Phong明暗处理以及完全的隐藏面消除的移动物体。这些高性能系统能提供实时的纹理映射、反走样、云雾和烟尘的大气衰减以及其他特殊效果。

图形软件的标准同样较第1版时明显进步了很多。第1版的SGP软件包基于的SIGGRAPH Core'79软件包、直视存储管和刷新矢量显示器现在基本上都已经消失了。支持结构层次存储和编辑的更加强大的PHIGS软件包已经成为ANSI和ISO的标准，广泛应用于科学和工程的实时几何图形制作中，与其相伴的PHIGS + 支持光照、明暗处理、曲线和曲面。官方的图形标准中补充了许多事实标准，如Apple公司的QuickDraw，X Window的Xlib二维整型光栅图形包和Silicon Graphics公司的GL三维图形库。Pixar公司的照片真实感的RanderMan软件和硬拷贝页和屏幕图像描述的PostScript图形解释器同样也得到了广泛的应用。更好的图形软件的使用使得对用户界面的“感观”得到巨大改善，我们可以期待增加对三维效果的利用，使我们对信息的管理、表现、检索和漫游提供新的想像空间和形象，必是出于审美的考虑。

也许图形学中最重要的发展方向是对物体建模技术的愈加重视，不仅仅是生成这些物体的画面，同时人们对如何描述随时间变化的三维几何物体的几何特性和行为产生了更大的兴趣。这样，图形学在建模和绘制中就更加关注模拟、动画及“回归物理过程”，试图使创作出的物体尽可能真实。

当图形工具变得越来越复杂时，我们就需要能够有效地运用它们。绘制不再是瓶颈，所以研究人员开始尝试用人工智能技术帮助进行物体建模设计、运动规划、二维及三维物体的有效图形表示的布局。

当今图形学技术发展迅猛，任何一本参考书都必须不断进行更新和扩充才能跟上这样的发展。本书基本上对《交互式计算机图形学基础》做了总体重写，尽管页数几乎翻番，我们仍然

不得不省略大量内容。

本书和第1版的主要区别如下：

- 矢量图形学的定位由光栅图形学所代替。
- 原来简单的二维浮点图形包（SGP）被SRGP和SPHIGS代替，体现了交互图形程序设计的两大主要流派。SRGP结合了QuickDraw和Xlib二维整型光栅图形包的特征；基于PHIGS的SPHIGS提供了具有层次显示列表的三维浮点包的基本特性。本书描述了如何应用这些标准图形包进行编程，并且同时讲述了这些图形包对于基本裁剪算法、扫描转换算法、观察算法和显示列表遍历算法等基本功能的实现细节。
- 在更深的层次上讨论了用户界面问题，包括二维桌面比喻和三维交互设备。
- 对建模的讨论包括了NURB（非均匀有理B样条）曲线和曲面，实体造型以及高级建模技术，如基于物理的建模、过程化模型、分形、L文法系统和粒子系统等。
- 对绘制技术的讨论增加了对反走样的详细讨论，以及可见面判定、光照和明暗处理，包括基于物理的光照模型、光线跟踪、辐射度等。
- 增加了高级光栅图形体系结构和算法的内容，包括裁剪、复杂图元的扫描转换、简单的图像处理操作等。
- 增加了对动画的简单讨论。

阅读本书无须具备图形学的基础知识，只需要了解一定的C语言编程技术、基本的数据结构和算法、计算机体系结构和简单的线性代数。附录中列出了阅读本书必需的数学基础。本书的全部内容可在两个学期内讲授，但是书中内容划分为几个部分，也可以有选择地进行讲授。因此，读者可以根据自己的需要选择学习，由浅入深。本书可以大致分为下面几个部分：

基本知识

第1章对历史进行了简要回顾，并对硬件、软件、应用程序的最基本问题作了讨论。第2、3章讲述了简单的二维整型图形包SRGP的应用和实现方法。第4章介绍了图形硬件，包括如何应用硬件完成前面几章中提到的操作。第5、6章通过矩阵的方法介绍了在平面和三维空间中进行变换的思想，如何应用齐次坐标来统一线性变换和仿射变换，三维视图的描述，包括从任意视见体到标准视见体的变换。最后，第7章介绍了三维浮点层次图形包SPHIGS，通过一些基本的建模操作讲述了它的用法，SPHIGS是PHIGS标准的简化版本。这一章还讨论了PHIGS中可用的层次结构的优缺点以及使用这个图形软件包的应用程序的结构。

用户界面

第8~10章讲述了当前交互设备并讨论了用户界面设计的一些高层次问题，对当前流行的各种用户界面设计范型进行了介绍和比较。最后的用户界面软件一章讨论了窗口管理器、交互技术库和用户界面管理系统。

模型定义

第11、12章讲述了当前的几何建模技术：曲线和曲面的参数函数表示（尤其是三次样条函数）以及各种技术的实体表示，包括边界表示和CSG模型。第13章介绍了人类的颜色视觉系统以及各种颜色描述系统及它们之间的转换，同时也讨论了如何有效运用颜色的规则。

图像合成

在连续四章中的第一章，即第14章，讲述了从最早的矢量绘图到最新的光照图形技术，人们对真实感的探索过程。走样所引起的人为痕迹是光栅图形学中首要考虑的问题，这一章讨论了产生这些人为痕迹的原因和利用傅里叶变换的解决办法。第15章详细讨论了可见面判定的不同方法。第

16章介绍了光照和明暗处理算法，本章的前半部分讨论了当前在流行的图形硬件中使用的算法而其余部分讨论了纹理、阴影、透明效果、反射、基于物理的光照模型、光线跟踪、辐射度，等等。第17章讲述了图像操纵（如像素图的缩放、错切、旋转）和图像存储技术（包括各种图像压缩技术）。

高级技术

最后四章对最新的图形学技术做了简介。第18章描述了当前的高端商用和研究用的图形硬件，本章由高性能图形体系结构的权威Steven Molnar和Henry Fuchs提供。第19章讲述了应用于如任意圆锥曲线的扫描转换、反走样文字生成、页描述语言实现（如PostScript）等任务的复杂光栅算法。最后两章对在高级建模和计算机动画领域中最重要的技术做了概述。

前两部分内容相对基础，可用于本科生的基础课程，随后的课程可使用其余各章的高级内容。当然也可以从各部分中抽取章节定制课程内容。

比如，以二维图形学为主的课程可以包括第1、2章，第3章中的简单扫描转换和裁剪，第4章中的概述、光栅体系结构和交互设备，第5章的齐次数学，6.1~6.3节从“如何使用三维观察”的角度讲解了三维观察，由第8、9、10章组成的用户界面部分，以及由第14、15、16章组成的图像合成部分的引言和简单算法。

一门图形学概论课程可以包括第1、2章，第3章的简单算法，第4章中的光栅体系结构和交互设备，第5章，第6、7章的大部分内容以及SPHIGS。课程的后半部分包括第11、13章中的建模技术，第14、15、16章的图像合成，以及第20章中的部分高级内容。

以三维建模和绘制为重点的课程可以从第3章讲述扫描转换、线和多边形的裁剪、反走样的章节开始，然后进行到讲述变换和观察的数学基础的第5、6章，以及关于颜色的第13章，第14~16章的主要内容。也可以加入曲面和实体建模、第20章的高级建模技术、第21章的动画等高级内容。

图形包

由David Sklar设计的SRGP和SPHIGS图形包可以从出版商获取，可用于IBM PC (ISBN 0-201-54700-7)、Macintosh (ISBN 0-201-54701-5)和运行X11的UNIX 工作站，同时包括扫描转换、裁剪、观察等多种算法。

致谢

本书的完成离不开很多朋友和同事的辛勤工作。我们特别感谢那些为本书提供大量材料的人们，同样感谢对本书提出建议的很多同事。如有遗漏，敬请原谅。Katrina Avery和Lyn Dupre为本书的编辑做了大量工作，同时还有Debbie van Dam、Melissa Gold和Clare Campbell。我们特别感谢产品监督Bette Aaronson，艺术指导Joe Vetere和编辑Keith Wollman，他们不仅为本书做出了不懈努力，同时他们在过去五年所处的困境下所表现的耐心和幽默也为本书的完成做出了很大的贡献。

计算机图形学已经不是一个由四个主要作者和三位辅助作者可以完全掌握的领域，我们的同事、学生为本书提供了大量有价值的意见和建议，并发现了很多错误。下列人士对本书的一章或几章进行了仔细的技术性审读：John Airey, Kurt Akeley, Tom Banchoff, Brian Barsky, David Bates, Cliff Beshers, Gary Bishop, Peter Bono, Marvin Bunker, Bill Buxton, Edward Chang, Norman Chin, Michael F. Cohen, William Cowan, John Dennis, Tom Dewald, Scott Draves, Steve Drucker, Tom Duff, Richard Economy, David Ellsworth, Nick England, Jerry Farrell, Robin Forrest, Alain Fournier, Alan Freiden, Christina Gibbs, Melissa Gold, Mark Green, Cathleen Greenberg, Margaret Hagen, Griff Hamlin, Pat Hanrahan, John Heidema, Rob Jacob, Abid Kamran, Mike Kappel, Henry Kaufman, Karen Kendler, David Kurlander, David Laidlaw, Keith Lantz,

Hsien-Che Lee, Aaron Marcus, Nelson Max, Deborah Mayhew, Barbara Meier, Gary Meyer, Jim Michener Iakob Nielsen, Mark Nodine, Randy Pausch, Ari Requicha, David Rosenthal, David Salesin. Hanan Samet, James Sanford, James Sargent, Robin Schaufler, Robert Scheifler, John Schnizlein, MichaeI Shantzis, Ben Shneiderman, Ken Shoemake, Judith Schrier, John Sibert, Dave Simons, Jonathan Steinhart, Maureen Stone, Paul Strauss, Seth Tager, Peter Tanner, Brice Tebbs, Ben Trumbore, Yi Tso, Greg Turk, Jeff Vroom, Colin Ware, Gary Watkins, Chuck Weger, Kevin Weiler, Turner Whitted, George Wolbetg和Larry Wolff。

我们的一些同事，包括Jack Bresenham, Brian Barsky, Jerry Van Aken, Dilip Da Silva（建议对第3章采取统一的中点方法）和Don Hatfield，不仅对章节做了仔细阅读，同时对其中的算法提出了详细的建议。

Katrina Avery, Barbara Britten, Clare Campbell, Tina Cantor, Joyce Cavatoni, Louisa Hogan, Jenni Rodda和Debbie van Dam做了大量的文字处理工作。Dan Robbins, Scott Snibbe, Tina Cantor和Clare Campbell绘制了第1至3章的插图。其他插图由Beth Cobb, David Kurlander Allen Paeth和George Wolberg（在Peter Karp的帮助下）提供。彩图II-21至彩图II-37展示了绘制技术的进步，由Pixar的Thomas Williams和H.B. Siegel在M.W. Mantle的指导下应用Pixar的PhotoRealistic RenderMan软件完成。感谢Industrial Light & Magic提供激光扫描仪创建了彩图II-24至彩图II-37，感谢Norman Chin 为彩图II-30至彩图II-32计算了顶点法向量。L. Lu和Carles Castellsagué 为制作插图编写了程序。

Jeff Vogel实现了第3章的算法，他和Atul Butte检查了第2和7章的程序。在Balsys, Scott Boyajian, Atul Butte, Alex Contovounesios和Scott Draves 的帮助下，David Sklar编写了Mac和X11上的SRGP 和SPHIGS。Randy Pausch和他的学生将这些包移植到了PC平台。

为了能够让读者获取多种算法的电子拷贝、提出习题、报告本书和SRGP/SPHIGS中的错误以及得到本书和软件的勘误表，我们已经安装了一个自动的电子邮件服务器。发一封主题为“Help”的电子邮件到graphtext@cs.brown.edu，就可收到当前可用的服务列表。

C语言版本序言

原先书中的例子是用Pascal语言编写的，此版本为C语言版本。C语言版本包含了Pascal版本第2版第9次印刷以来的所有改动，同时还包含对一些算法的小改动；另外，书中所有的Pascal语言的代码都已用ANSI C重写。SRGP和SPHIGS图形包的接口现在都采用C语言定义，相应的这些图形包也都采用了新的C语言实现，不再使用Pascal。

我们衷心地感谢Norman Chin，他把原书第2版中的Pascal语言代码转换成了C语言代码，校正这些代码并且按照原书中的排版习惯对代码进行编排。感谢Matt Ayers，他对书的第2、3和7章进行了认真的校对，并且对于转换代码过程中的问题提出了许多有用的建议。

华盛顿州 J.D.F.
罗德岛州 A.v.D
纽约州 S.K.F.
罗德岛州 J.F.H.

目　录

出版者的话
专家指导委员会
译者序
前言
第1章　导言 ……1
1.1　作为图像分析的图像处理 ……1
1.2　交互式图形学的优点 ……2
1.3　计算机图形学的典型用途 ……3
1.4　应用分类 ……4
1.5　计算机图形学硬件与软件的发展 ……5
1.5.1　输出技术 ……6
1.5.2　输入技术 ……10
1.5.3　软件的可移植性与图形标准 ……11
1.6　交互式图形学的概念框架 ……12
1.6.1　概述 ……12
1.6.2　应用建模 ……12
1.6.3　对图形系统描绘观察的内容 ……13
1.6.4　交互处理 ……14
1.7　小结 ……15
习题 ……15
第2章　简单光栅图形软件包(SRGP)的编程 ……17
2.1　用SRGP画图 ……17
2.1.1　图形图元的规格 ……17
2.1.2　属性 ……21
2.1.3　填充图元及其属性 ……22
2.1.4　存储和恢复属性 ……25
2.1.5　文本 ……26
2.2　基本交互处理 ……27
2.2.1　人的因素 ……27
2.2.2　逻辑输入设备 ……28
2.2.3　采样与事件驱动处理 ……28
2.2.4　采样模式 ……30
2.2.5　事件模式 ……31
2.2.6　交互处理中的关联拾取 ……34
2.2.7　设置设备度量和属性 ……35
2.3　光栅图形特性 ……37
2.3.1　画布 ……37
2.3.2　矩形框的裁剪 ……39
2.3.3　SRGP_copyPixel操作 ……39
2.3.4　写模式或RasterOp ……41
2.4　SRGP的局限性 ……42
2.4.1　应用程序坐标系统 ……42
2.4.2　为了重新定义存储图元 ……43
2.5　小结 ……45
习题 ……45
程序设计项目 ……47
第3章　二维图元的基本光栅图形学算法 ……49
3.1　概述 ……49
3.1.1　显示系统体系结构的含义 ……49
3.1.2　软件中的输出流水线 ……52
3.2　直线的扫描转换 ……52
3.2.1　基本增量算法 ……53
3.2.2　中点线算法 ……54
3.2.3　补充要点 ……57
3.3　圆的扫描转换 ……59
3.3.1　八方向对称性 ……59
3.3.2　中点圆算法 ……60
3.4　椭圆的扫描转换 ……62
3.5　填充矩形 ……65
3.6　填充多边形 ……66
3.6.1　水平边 ……68
3.6.2　狭长条 ……68
3.6.3　边相关性和扫描线算法 ……68
3.7　填充椭圆弧区域 ……71
3.8　图案填充 ……72
3.9　宽图元 ……75
3.9.1　复制像素 ……75

3.9.2 移动画笔 ……76
3.9.3 填充边界之间的区域 ……77
3.9.4 用宽折线进行逼近 ……78
3.10 线型和笔型 ……78
3.11 光栅空间的裁剪操作 ……79
3.12 线段裁剪 ……80
3.12.1 裁剪端点 ……80
3.12.2 利用求解联立方程组的线段裁剪 …81
3.12.3 Cohen-Sutherland 线裁剪算法 ……81
3.12.4 参数化的线裁剪算法 ……84
3.13 圆和椭圆的裁剪 ……90
3.14 多边形裁剪 ……90
3.15 生成字符 ……93
3.15.1 定义和裁剪字符 ……93
3.15.2 一种文本输出图元的实现 ……95
3.16 SRGP_copyPixel ……96
3.17 反走样 ……96
3.17.1 增加分辨率 ……96
3.17.2 未加权的区域采样 ……97
3.17.3 加权区域采样 ……98
3.17.4 Gupta-Sproull反走样线扫描算法 …101
3.18 小结 ……103
习题 ……104
第4章 图形硬件 ……107
4.1 硬拷贝技术 ……108
4.2 显示技术 ……113
4.3 光栅扫描显示系统 ……120
4.3.1 简单的光栅显示系统 ……120
4.3.2 具有外围显示处理器的光栅显示系统 ……124
4.3.3 显示处理器的附加功能 ……127
4.3.4 具有集成显示处理器的光栅显示系统 ……129
4.4 视频控制器 ……130
4.4.1 查找表动画 ……131
4.4.2 位图变换和窗口技术 ……132
4.4.3 视频混合 ……133
4.5 随机扫描显示处理器 ……134
4.6 用于操作者交互的输入设备 ……136
4.6.1 定位设备 ……136
4.6.2 键盘设备 ……140
4.6.3 定值设备 ……140
4.6.4 选择设备 ……140
4.7 图像扫描仪 ……141
习题 ……142
第5章 几何变换 ……145
5.1 二维变换 ……145
5.2 齐次坐标和二维变换的矩阵表示 ……147
5.3 二维变换的合成 ……150
5.4 窗口到视口的变换 ……152
5.5 效率 ……153
5.6 三维变换的矩阵表示 ……154
5.7 三维变换的合成 ……157
5.8 坐标系的变换 ……160
习题 ……163
第6章 三维空间的观察 ……165
6.1 投影 ……165
6.1.1 透视投影 ……166
6.1.2 平行投影 ……167
6.2 指定一个任意的三维视图 ……170
6.3 三维观察的例子 ……173
6.3.1 透视投影 ……175
6.3.2 平行投影 ……177
6.3.3 有限的视见体 ……179
6.4 平面几何投影的数学 ……180
6.5 实现平面几何投影 ……183
6.5.1 平行投影 ……185
6.5.2 透视投影 ……188
6.5.3 用三维规范视见体进行裁剪 ……192
6.5.4 在齐次坐标中裁剪 ……194
6.5.5 映射到一个视口 ……197
6.5.6 实现小结 ……197
6.6 坐标系统 ……198
习题 ……199
第7章 对象的层次结构和简单的PHIGS系统 ……203
7.1 几何造型 ……204
7.1.1 什么是模型 ……204
7.1.2 几何模型 ……205
7.1.3 几何模型中的层次 ……205

7.1.4 模型、应用程序和图形系统间的关系 ……207
7.2 保留模式图形包的特点 ……208
7.2.1 中央结构存储库及其优点 ……208
7.2.2 保留模式软件包的局限性 ……209
7.3 定义和显示结构 ……209
7.3.1 打开和关闭结构 ……209
7.3.2 定义输出图元及其属性 ……210
7.3.3 提交结构进行显示遍历 ……212
7.3.4 观察 ……213
7.3.5 通过窗口管理共享屏幕的图像应用 …215
7.4 模型变换 ……216
7.5 层次式结构网络 ……219
7.5.1 两层层次结构 ……219
7.5.2 简单的三层层次结构 ……220
7.5.3 自底向上构造的机器人 ……221
7.5.4 交互式造型程序 ……223
7.6 显示遍历中的矩阵合成 ……223
7.7 层次结构中外观属性的处理 ……226
7.7.1 继承法则 ……226
7.7.2 SPHIGS的属性及文字不受变换影响 ……227
7.8 屏幕的更新和绘制模式 ……228
7.9 用于动态效果的结构网络编辑 ……229
7.9.1 利用索引和标记访问元素 ……229
7.9.2 内部结构的编辑操作 ……230
7.9.3 改进编辑方法的一些实例块 ……230
7.9.4 如何控制屏幕图像的自动再生 ……232
7.10 交互 ……232
7.10.1 定位器 ……233
7.10.2 关联拾取 ……233
7.11 其他输出特性 ……235
7.11.1 属性包 ……235
7.11.2 高亮度与不可见性的名字集 ……236
7.11.3 图像交换与元文件 ……236
7.12 实现问题 ……237
7.12.1 绘制 ……237
7.12.2 关联拾取 ……240
7.13 层次模型的优化显示 ……241
7.13.1 省略 ……241
7.13.2 参考结构 ……242
7.14 PHIGS中层次模型的局限性 ……242
7.14.1 简单层次结构的局限性 ……242
7.14.2 SPHIGS“参数传递”的局限性 …242
7.15 层次建模的其他形式 ……243
7.15.1 过程层次 ……243
7.15.2 数据层次 ……244
7.15.3 利用数据库系统 ……244
7.16 小结 ……245
习题 ……245
第8章 输入设备、交互技术与交互任务 …247
8.1 交互硬件 ……248
8.1.1 定位设备 ……249
8.1.2 键盘设备 ……250
8.1.3 定值设备 ……250
8.1.4 选择设备 ……250
8.1.5 其他设备 ……251
8.1.6 三维交互设备 ……252
8.1.7 设备级人的因素 ……254
8.2 基本交互任务 ……254
8.2.1 定位交互任务 ……254
8.2.2 选择交互任务——大小可变的选项集合 ……256
8.2.3 选择交互任务——相对固定大小的选项集合 ……259
8.2.4 文本交互任务 ……264
8.2.5 定量交互任务 ……265
8.2.6 三维交互任务 ……266
8.3 复合交互任务 ……270
8.3.1 对话框 ……270
8.3.2 构造技术 ……270
8.3.3 动态操纵 ……273
习题 ……275
第9章 对话设计 ……277
9.1 人机对话的形式和内容 ……278
9.2 用户界面风格 ……280
9.2.1 所见即所得 ……280
9.2.2 直接操纵 ……281
9.2.3 图标化用户界面 ……282
9.2.4 其他对话形式 ……284

9.3 一些重要的设计问题……285
9.3.1 一致性……285
9.3.2 提供反馈……287
9.3.3 减少错误概率……288
9.3.4 提供错误恢复……289
9.3.5 容许多种熟练级别……290
9.3.6 减少记忆……292
9.4 模式和语法……292
9.5 视觉设计……295
9.5.1 视觉清晰性……295
9.5.2 视觉编码……298
9.5.3 视觉的一致性……300
9.5.4 布局原则……301
9.6 设计方法学……303
习题……304
第10章 用户界面软件……307
10.1 基本的交互处理模型……307
10.2 窗口管理系统……310
10.3 窗口系统中的输出处理……312
10.4 窗口系统中的输入处理……315
10.5 交互技术工具箱……318
10.6 用户界面管理系统……322
10.6.1 对话序列……322
10.6.2 高级UIMS概念……328
习题……331
第11章 曲线与曲面的表示……333
11.1 多边形网格……334
11.1.1 多边形网格的表示……334
11.1.2 多边形网格表示法的一致性……336
11.1.3 平面方程……336
11.2 三次参数曲线……337
11.2.1 Hermite曲线……341
11.2.2 Bézier曲线……344
11.2.3 均匀非有理B样条曲线……346
11.2.4 非均匀非有理B样条曲线……349
11.2.5 非均匀有理三次多项式曲线段……355
11.2.6 其他样条曲线……356
11.2.7 曲线分割……358
11.2.8 各种表示法之间的转换……360
11.2.9 曲线绘制……360
11.2.10 三次曲线的比较……363
11.3 双三次参数曲面……364
11.3.1 Hermite曲面……365
11.3.2 Bézier曲面……367
11.3.3 B样条曲面……368
11.3.4 曲面的法线……368
11.3.5 双三次曲面的显示……369
11.4 二次曲面……372
11.5 小结……373
习题……373
第12章 实体造型……377
12.1 实体表示……377
12.2 正则布尔集合运算……378
12.3 基本实体举例法……381
12.4 扫掠表示法……381
12.5 边界表示法……383
12.5.1 多面体和欧拉公式……383
12.5.2 翼边表示法……385
12.5.3 布尔集合运算……385
12.5.4 非多边形的边界表示法……386
12.6 空间划分表示法……387
12.6.1 单元分解法……387
12.6.2 空间位置枚举法……387
12.6.3 八叉树表示法……388
12.6.4 二元空间划分树……392
12.7 构造实体几何……393
12.8 各种表示法的比较……394
12.9 实体造型的用户界面……396
12.10 小结……396
习题……397
第13章 消色差光与彩色光……399
13.1 消色差光……399
13.1.1 选择亮度值——gamma校正……399
13.1.2 半色调逼近……402
13.2 彩色……406
13.2.1 心理物理学……407
13.2.2 CIE色度图……409
13.3 用于光栅图形的颜色模型……412
13.3.1 RGB颜色模型……413
13.3.2 CMY颜色模型……414

13.3.3 YIQ颜色模型 ……415
13.3.4 HSV颜色模型 ……416
13.3.5 HLS颜色模型 ……418
13.3.6 颜色的交互指定 ……421
13.3.7 在颜色空间中进行插值 ……422
13.4 颜色再现 ……422
13.5 在计算机图形学中应用颜色 ……424
13.6 小结 ……425
习题 ……426
第14章 可视图像真实感的探讨 ……429
14.1 为什么讨论真实感 ……429
14.2 基本的困难 ……430
14.3 线条图的绘制技术 ……431
14.3.1 多正交视图 ……431
14.3.2 轴测投影和斜投影 ……432
14.3.3 透视投影 ……432
14.3.4 深度提示 ……432
14.3.5 深度裁剪 ……432
14.3.6 纹理 ……433
14.3.7 颜色 ……433
14.3.8 可见线的判定 ……433
14.4 明暗图像的绘制技术 ……433
14.4.1 可见面的判定 ……433
14.4.2 光照和明暗处理 ……433
14.4.3 插值明暗处理 ……434
14.4.4 材质属性 ……434
14.4.5 曲面造型 ……434
14.4.6 改进光照和明暗效果 ……434
14.4.7 纹理 ……434
14.4.8 阴影 ……434
14.4.9 透明性和反射 ……435
14.4.10 改进的相机模型 ……435
14.5 改进的物体模型 ……435
14.6 动力学 ……435
14.7 立体观测 ……436
14.8 改进的显示技术 ……436
14.9 与其他感官的交互 ……437
14.10 走样与反走样 ……437
14.10.1 点采样 ……439
14.10.2 区域采样 ……439
14.10.3 采样理论 ……441
14.10.4 滤波 ……445
14.10.5 重构 ……450
14.10.6 实际的反走样 ……454
14.11 小结 ……456
习题 ……456
第15章 可见面的判定 ……459
15.1 双变量函数 ……460
15.2 可见面判定算法中的常用技术 ……463
15.2.1 相关性 ……464
15.2.2 透视变换 ……464
15.2.3 范围与包围体 ……466
15.2.4 背面消除 ……467
15.2.5 空间划分 ……468
15.2.6 层次结构 ……469
15.3 可见线判定算法 ……469
15.3.1 Roberts算法 ……469
15.3.2 Appel算法 ……470
15.3.3 光环线算法 ……470
15.4 z缓存算法 ……471
15.5 列表优先级算法 ……474
15.5.1 深度排序算法 ……474
15.5.2 二元空间划分树 ……476
15.6 扫描线算法 ……480
15.7 区域细分算法 ……484
15.7.1 Warnock算法 ……484
15.7.2 Weiler-Atherton算法 ……486
15.7.3 子像素区域细分算法 ……489
15.8 八叉树算法 ……490
15.9 曲面算法 ……492
15.10 可见面光线跟踪 ……494
15.10.1 相交计算 ……495
15.10.2 可见面光线跟踪算法的效率 ……496
15.10.3 计算布尔集合运算 ……501
15.10.4 反走样光线跟踪 ……503
15.11 小结 ……504
习题 ……505
第16章 光照和明暗处理 ……509
16.1 光照模型 ……509

16.1.1 环境光 ……509
16.1.2 漫反射 ……510
16.1.3 大气衰减 ……513
16.1.4 镜面反射 ……514
16.1.5 点光源模型的改进 ……516
16.1.6 多光源 ……517
16.2 多边形的明暗处理模型 ……518
16.2.1 恒定明暗处理 ……518
16.2.2 插值明暗处理 ……518
16.2.3 多边形网格的明暗处理 ……518
16.2.4 Gouraud 明暗处理技术 ……519
16.2.5 Phong明暗处理技术 ……520
16.2.6 插值明暗处理中的问题 ……521
16.3 曲面细节 ……522
16.3.1 曲面细节多边形 ……523
16.3.2 纹理映射 ……523
16.3.3 凹凸映射 ……524
16.3.4 其他方法 ……525
16.4 阴影 ……525
16.4.1 扫描线生成阴影算法 ……526
16.4.2 对象精确的两步法阴影算法 ……526
16.4.3 阴影体 ……528
16.4.4 两遍z缓存阴影算法 ……529
16.4.5 全局光照阴影算法 ……531
16.5 透明性 ……531
16.5.1 无折射的透明性 ……531
16.5.2 折射透明性 ……533
16.6 物体间的反射 ……534
16.7 基于物理的光照模型 ……536
16.7.1 表面模型的改进 ……538
16.7.2 微面元分布函数 ……538
16.7.3 几何衰减因子 ……539
16.7.4 菲涅耳项 ……539
16.8 扩展光源模型 ……543
16.9 光谱采样 ……543
16.10 相机模型的改进 ……545
16.11 全局光照算法 ……545
16.12 递归光线跟踪 ……546
16.12.1 递归光线跟踪算法的效率考虑 ……550
16.12.2 一个更佳的照明模型 ……553
16.12.3 区域采样的不同方法 ……553
16.12.4 分布式光线跟踪 ……554
16.12.5 从光源出发的光线跟踪 ……557
16.13 辐射度方法 ……558
16.13.1 辐射度方程 ……558
16.13.2 计算形状因子 ……560
16.13.3 子结构技术 ……562
16.13.4 逐步求精算法 ……562
16.13.5 更加精确的形状因子的计算 ……565
16.13.6 镜面反射 ……565
16.13.7 辐射度和光线跟踪的结合 ……565
16.14 绘制流水线 ……567
16.14.1 局部光照绘制流水线 ……567
16.14.2 全局光照绘制流水线 ……569
16.14.3 设计灵活的绘制法 ……569
16.14.4 逐步求精方法 ……571
16.15 小结 ……571
习题 ……572
第17章 图像处理和存储 ……575
17.1 什么是图像 ……576
17.2 滤波 ……576
17.3 图像处理 ……578
17.4 图像的几何变换 ……578
17.4.1 基本几何变换 ……579
17.4.2 带滤波的几何变换 ……581
17.4.3 其他图案映射技术 ……583
17.5 多重变换 ……584
17.5.1 多重变换的代数学 ……585
17.5.2 利用滤波生成变换后的图像 ……587
17.5.3 评价变换方法 ……589
17.6 图像合成 ……589
17.6.1 α通道合成 ……589
17.6.2 其他合成方法 ……593
17.6.3 通过填充机制生成α值 ……595
17.6.4 用于图像组装的一个界面 ……595
17.7 图像存储机制 ……596
17.7.1 存储图像数据 ……597
17.7.2 用于图像压缩的迭代函数系统 ……598
17.7.3 图像属性 ……600
17.8 图像的特殊效果 ……601

17.9 小结 ……601
习题 ……601
第18章 高级光栅图形体系结构 ……605
18.1 简单光栅显示系统 ……605
18.1.1 帧缓冲内存访问问题 ……605
18.1.2 动态存储器 ……606
18.1.3 提高帧缓冲内存带宽 ……607
18.1.4 视频RAM ……607
18.1.5 高分辨率显示器的帧缓存 ……608
18.2 显示处理器系统 ……609
18.2.1 外部显示处理器 ……609
18.2.2 德克萨斯仪器公司的TMS34020——单芯片外部显示处理器 ……610
18.2.3 集成的图形处理器 ……611
18.2.4 Intel的i860——一个具有集成3D图形支持的单芯片微处理器 ……611
18.2.5 三个性能障碍 ……612
18.3 标准图形流水线 ……613
18.3.1 显示遍历 ……613
18.3.2 模型变换 ……614
18.3.3 简单接受/简单拒绝的区分 ……614
18.3.4 光照处理 ……614
18.3.5 观察变换 ……615
18.3.6 裁剪 ……615
18.3.7 除以w并映射到3D视口 ……615
18.3.8 光栅化 ……615
18.3.9 一个样板应用的性能要求 ……616
18.4 多处理简介 ……618
18.4.1 流水线 ……618
18.4.2 并行性 ……619
18.4.3 多处理器图形系统 ……620
18.5 流水线前端体系结构 ……620
18.5.1 应用程序和显示遍历 ……620
18.5.2 几何变换 ……621
18.5.3 简单接受/简单拒绝的区分 ……621
18.5.4 光照处理 ……621
18.5.5 裁剪 ……621
18.5.6 除w并映射到3D视口 ……622
18.5.7 前端流水线的限制 ……622
18.6 并行前端体系结构 ……622
18.6.1 显示遍历 ……622
18.6.2 重组并行数据流 ……623
18.6.3 流水线同并行性的比较 ……624
18.7 多处理器光栅化体系机构 ……624
18.7.1 以物体为序的流水线体系结构 ……624
18.7.2 以图像为序的流水线体系结构 ……627
18.7.3 流水线光栅化的限制和对并行性的需求 ……627
18.8 图像并行光栅化 ……627
18.8.1 内存划分体系结构 ……628
18.8.2 Silicon Graphics公司的 POWER IRIS 4D/240GTX——一个交叉划分帧缓冲内存体系结构 ……630
18.8.3 逻辑增强的内存 ……633
18.9 物体并行光栅化 ……637
18.9.1 每图元一个处理器的流水线 ……638
18.9.2 基于树结构的每图元一个处理器的体系结构 ……638
18.9.3 物体并行性和图像并行性的比较 ……639
18.10 混合并行光栅化 ……639
18.10.1 虚拟缓冲区和虚拟处理器 ……639
18.10.2 并行虚拟缓冲区体系结构 ……641
18.10.3 图像合成体系结构 ……642
18.11 增强的显示能力 ……643
18.11.1 对多窗口的支持 ……643
18.11.2 对增加的真实感的支持 ……644
18.11.3 Stellar GS2000——促进真实感绘制的紧密集成的体系结构 ……646
18.11.4 对高级图元的支持 ……647
18.11.5 对增强的3D感知的支持 ……649
18.11.6 实时飞行模拟器 ……651
18.12 小结 ……652
习题 ……652
第19章 高级几何与光栅算法 ……655
19.1 裁剪 ……655
19.1.1 矩形区域对直线的裁剪 ……656
19.1.2 矩形和其他多边形区域对多边形的裁剪 ……659
19.1.3 矩形区域裁剪：梁友栋-Barsky多边形算法 ……659
19.1.4 Weiler多边形算法 ……665

19.2 图元的扫描转换 ……671
19.2.1 属性 ……671
19.2.2 评价扫描转换算法的准则 ……671
19.2.3 直线的其他考察方式 ……673
19.2.4 高级折线算法 ……674
19.2.5 画圆算法的改进 ……674
19.2.6 一般圆锥曲线算法 ……676
19.2.7 宽图元 ……683
19.2.8 填充图元 ……685
19.3 反走样 ……686
19.3.1 直线的反走样 ……687
19.3.2 圆的反走样 ……688
19.3.3 圆锥曲线的反走样 ……690
19.3.4 一般曲线的反走样 ……692
19.3.5 矩形、多边形和直线端点的反走样 …692
19.4 文字的特殊问题 ……693
19.5 填充算法 ……695
19.5.1 区域类型、连通性和填充 ……695
19.5.2 基本填充算法 ……696
19.5.3 软填充算法 ……698
19.6 加速copyPixel ……700
19.7 形状数据结构和形状代数 ……704
19.8 用bitBlt管理窗口 ……706
19.9 页面描述语言 ……708
19.10 小结 ……713
习题 ……713
第20章 高级建模技术 ……719
20.1 前述技术的扩展 ……720
20.1.1 采用样条的高级建模技术 ……720
20.1.2 基于噪声的纹理映射 ……722
20.2 过程模型 ……724
20.3 分形模型 ……725
20.4 基于文法的模型 ……729
20.5 粒子系统 ……732
20.6 体绘制 ……735
20.7 基于物理的建模 ……738
20.7.1 基于约束的建模 ……738
20.7.2 布面和柔软表面的建模 ……739
20.7.3 实体建模 ……740
20.7.4 地形建模 ……740
20.8 模拟自然物体和合成物体的特殊模型 ……740
20.8.1 波浪 ……741
20.8.2 云层和气象 ……741
20.8.3 湍流 ……743
20.8.4 滴状物体 ……743
20.8.5 生物 ……744
20.8.6 人 ……745
20.8.7 来自于娱乐业的一个例子 ……745
20.9 自动放置物体 ……746
20.10 小结 ……748
习题 ……749
第21章 动画 ……751
21.1 传统动画和计算机辅助动画 ……751
21.1.1 传统动画 ……751
21.1.2 计算机辅助动画 ……752
21.1.3 插值 ……752
21.1.4 简单的动画效果 ……755
21.2 计算机动画语言 ……756
21.2.1 线性表表示法 ……756
21.2.2 通用计算机语言 ……757
21.2.3 图形语言 ……757
21.3 动画控制方法 ……760
21.3.1 完全显式的控制 ……760
21.3.2 过程化控制 ……760
21.3.3 基于约束的系统 ……760
21.3.4 真实动作跟踪 ……762
21.3.5 演员 ……762
21.3.6 运动学和动力学 ……762
21.3.7 基于物理的动画 ……764
21.4 动画的基本规则 ……765
21.5 动画特有的一些问题 ……765
21.6 小结 ……767
习题 ……767
附录 计算机图形学的数学基础 ……769
参考文献 ……793
索引 ……833

第1章 导 言

计算机图形学始于计算机出现后不久，在硬拷贝绘图仪和阴极射线管荧光屏上显示数据。它现在已发展到包括产生、存储和操作对象的模型与图像。这些模型来自于广泛的，并且还在不断扩展的领域，包括物理、数学、工程、建筑，甚至概念（抽象）结构及自然现象等。当今的计算机图形主要是交互的，即用户可以通过使用输入设备（如键盘、鼠标器或荧光屏上的触敏膜）控制对象及其显示图像的内容、构造和外观。由于输入设备与显示之间的密切联系，如何运用这些设备亦成为计算机图形学的研究内容。

直到20世纪80年代早期，计算机图形学还是一个狭窄而特殊的领域，这主要是因为当时的硬件十分昂贵，而且缺乏易用和廉价的基于图形的应用程序。此后，带有内置式光栅图形显示器的个人计算机（例如施乐Star，以及后来投入批量生产、价格较低的苹果Macintosh和IBM PC及其兼容机）使得人机交互中使用位图图形广为流行。位图是一个在屏幕上由0和1表述的矩形点阵，每一个点称为一个像素（pixel）或图像单元（pel）。位图图形一经出台，便很快导致了廉价易用的基于图形应用程序的大量涌现。基于图形的用户界面使数百万新用户得以驾驭简单、便宜的应用程序，例如，电子表格、字处理和制图程序。

“桌面”概念现在已成为一个为人熟知的组织屏幕空间的方式。借助窗口管理器，用户可以产生、定位并调整矩形屏幕的面积，称为窗口。其作用像一个虚拟图形终端，每一个窗口运行一个应用程序。这使得用户只要用鼠标或其他设备指向希望的窗口，就可以在多个活动间互相切换。像在一张杂乱书桌上的纸张一样，窗口可以任意地重叠。图标显示也是桌面隐喻的一部分，它不仅可表示数据文件和应用程序，还可表示公共的办公对象，例如文件夹、邮箱、打印机和垃圾箱，它们在计算机上的操作与其实际对应物的使用完全类似。通过“点击”直接操纵对象取代了早期操作系统和计算机应用中许多神秘的键入命令。因此，用户可以选择图标来激活对应的程序或对象，或者选择下拉式或弹出式屏幕菜单中的按钮来做出选择。今天，几乎所有的交互式应用程序，甚至连那些操作文本（如字处理）或数值数据（如电子表格）的程序，在用户界面和显示及操纵特定应用对象时都广泛地使用了图形。光栅显示器（使用位图显示）的图形交互已取代了大多数文字终端上的文本交互。

即使是那些在日常工作中不使用计算机的人也会在电视广告和电影特技当中遇到计算机图形。计算机图形已不再是稀罕物。它是所有计算机用户界面的一个组成部分，并且对于二维（2D）、三维（3D）和更高维对象的可视化是不可或缺的。诸如教育、科学、工程、医学、商业、军事、广告和娱乐等各种各样的领域都离不开计算机图形。现在，学习编程和使用计算机，通常也包括学习如何使用简单的二维图形。

1.1 作为图像分析的图像处理

计算机图形学研究从真实或虚构物体的计算机模型合成画像；而图像处理（也称为画像处理）则研究与此相反的过程：场景分析，即由物体的画像重构其二维或三维模型。画像分析在许多领域中尤显重要。例如，航空侦察摄影，由空间探测器收集的月球或行星的慢扫描电视图像，工业机器人“眼”里的电视图像、染色体扫描、X射线图像，计算机化轴向断层摄影法

（CAT）扫描和指纹分析都利用了图像处理技术（见彩图I-1）。图像处理包括图像增强、模式检测与识别、场景分析和计算机视觉几个子领域。图像增强是指通过消除噪声（多余或缺少的像素数据）或增加对比度以改善图像的质量。模式检测与识别是指检测和分类标准模式，并找出（实际图像）与这些标准模式的偏差（变形）。一个特别重要的例子是光学字符识别（OCR）技术，它把大量排版的、打字的甚至是手写的纸页经济地输入计算机。场景分析和计算机视觉让科学家从若干二维图像来识别并重构场景的三维模型。一个例子是工业机器人检测传送带上零
2 部件的相对尺寸、形状、位置和颜色等。

尽管计算机图形学和图像处理都涉及图像的计算机处理，但直至最近，它们一直是完全独立的学科。然而，现在它们都在使用光栅显示器，二者的重迭部分不断增加，这在两个方面表现得极为明显。首先，在交互式图像处理中，借助菜单或其他图像交互技术的人工介入，有助于对实时显示在荧光屏上的连续色调图像变换的各个子过程进行控制。例如，扫描进来的照片在印刷之前，可以用电子方式进行润色、裁剪并与其他图像（甚至是计算机生成的合成图像）组合。其次，在计算机图形学中，简单的图像处理操作经常被用来帮助合成模型的图像。某些变换和组合合成图像的方法在很大程度上依赖于图像处理操作。

1.2 交互式图形学的优点

图形给人类与计算机的交流提供了一种最为自然的方式，因为人们的高度发达的二维和三维模式识别能力使我们可以快速、高效地理解和处理图像数据。在当今的许多设计、实现和构筑过程中，图像所能提供给我们的信息实际上是不可缺少的。科学计算可视化在20世纪80年代后期成为一个重要的领域，科学家和工程师们知道，如果不对数据进行归纳，并用各种图形表现方法去突出趋势与现象，他们就无法理解由超级计算机运算产生的大量数据。

然而，在推广应用的过程中，图像的生成与再生也遇到了一些技术问题。所以，只有在出现了便宜、简单的制造图片的技术，首先是打印印刷，然后是印像技术之后，中国的古训“百闻不如一见”才会感到过时。

交互式计算机图形是自摄影术和电视发明以来，生成图像的最重要的方法。它有许多额外的优点。利用计算机，我们不仅可以制作有形的“真实世界”物体的图像，还可以制作抽象的合成物体的图像，例如，四维数学曲面（见彩图I-3和彩图I-4）和像综述结果那样的先天没有几何形状的数据的图像。此外，我们不必局限于静态图像。虽然静态图像是交流信息的一种好方法，但动态变化的图像往往效果更好。可以这样讲，一幅动态图像抵得上上万幅静态图像。这句话特别适用于时变现象，无论是真实的过程（如超音速飞行中飞机机翼的变形或人的脸部从童年到老年的演变），还是抽象的过程（如美国核能使用量或城乡间人口迁徙的增长趋势）。因此，一部影片比一串幻灯片更能形象地表现随着时间的演变。类似地，以每秒超过15帧的速度在荧光屏上显示的画帧，比间隔数秒跳跃式变化的画帧序列，能够更好地传达平滑的运动和
3 变化。如果用户可以通过调整速度、视角内场景的多寡、显示细节的程度和场景中物体的几何关系等来控制动画的话，则更能发挥动画的效果。所以，许多交互式的图形技术都包含了用户控制的运动动力学和更新动力学的硬件和软件。

借助*运动动力学*（motion dynamics），可使物体相对于静止的观察者移动或旋转；也可保持物体不动，而使观察者围绕物体移动、摇动镜头以选择视框或者拉近或推远景物以选择细节的多寡，这就像快速移动摄像机时，通过瞄准镜所看到的情景一样。在很多情况下，物体和观察者都在运动。一个典型的例子是飞行模拟器（彩图I-5a和I-5b），它由一台支撑着一个模拟飞

机座舱的机械平台和若干窗式显示屏组成。计算机控制平台的运动及运动的规则，并模拟飞行员航行中看到的静止和运动的周围世界。这些价值数百万美元的系统通过让飞行员在一个模拟的三维景观当中和模拟的飞行器周围环境，操纵一架模拟飞机来训练飞行员。在个人计算机和工作站的流行游戏中有一些非常简单的飞行模拟器。游乐场也提供一些在陆地或超陆地景观中驰骋的“运动模拟器”。视频车道提供了基于图形的敏捷游戏（彩图I-6）和赛车驾驶模拟器（一种利用交互运动的视频游戏），在其中，当树木、建筑物和其他车辆从身边飞驰而过时，游戏者可以用“gas pedal”或“steering wheel”改变自己的速度和方向（见彩图I-7）。类似地，运动动力学使用户可以在建筑物、分子和三维或四维数学空间的内部或周围漫游。在另一类运动动力学中，“照相机”是不动的，而场景中的物体相对于“照相机”运动。例如，对于一台蒸汽机，可以适当地移动或旋转各个部件，来动态地显示其复杂的机械联接关系。

更新动力学（update dynamics）是指被观察物体的形状、颜色或其他性质实际发生了变化。例如，一个系统可以显示一架飞机的结构在飞行中的变形，或一个核反应堆流程中的状态因操作者在许多控制机构的图形界面上的操作而发生的变化。变化得越平滑，其结果就越真实、越有意义。动态的交互图形提供了大量的、用户可控的编码和交换信息的模式：在一幅图片中物体的二维或三维形状，它们的灰度或颜色，以及这些性质随着时间的变化。利用新近开发的数字信号处理器（DSP）和声音合成芯片，现在已经可以将声音反馈加到图形反馈上，使模拟环境更加逼真。

因此，交互的计算机图形允许大量的、高带宽人机交互。这会明显地增强我们理解数据、把握趋势和观看真实或假想对象的能力，实际上我们可以创造一个从任何角度都可以观察的“虚拟世界”（见彩图I-15和I-16）。通过高效率的信息交流，图形使得更优质和更精确的结果或产品、更高的生产率和更低的分析与设计成本成为可能。

1.3 计算机图形学的典型用途

今天，计算机图形已被用于工业、商业、政府、教育、娱乐等各个方面。最近，还进入了家庭。其应用不胜枚举，而且，随着图形能力成为日常商品，其应用还在不断增加。让我们看 4
一看在这些领域中有代表性的例子。

- *用户界面*：正如我们已经提到的，大多数运行于个人计算机和工作站，甚至运行于挂靠在分时式计算机和网络计算机服务器终端上的应用程序，都具有用户界面。它依靠桌上窗口系统管理多个同时进行的活动，并依靠点击功能使用户可以选择菜单项、图标和屏幕上的对象。只有在输入将要被保存和操作的文本时，才需要打字。字处理、电子表格和桌上印刷程序都是能够发挥上述用户界面技术的典型应用。本书的作者就使用了这样的程序生成文字和图表，然后，出版商及其协作商用类似的排版和绘图软件印制了本书。
- *商业与科技中的（交互）绘图*：当今图形的另一个极为广泛的应用也许是绘制二维和三维的数学、物理或经济函数的图；柱状图、条状图或饼图，任务调度图，库存和生产图等。所有这些图都是用来明确而简洁地表示由数据抽取出来的趋势和模式，从而将复杂的现象分类，做出有见识的决定。
- *办公自动化与电子出版*：自从出现了个人计算机上的桌上印刷系统，图形在产生和传播信息中的应用就大量增加。许多曾把自己的出版物交给外面的专业厂商印刷的组织现在可以在内部自己印制资料。办公自动化和电子印刷既可以输出传统的打印（硬拷贝）文件，也可以制成电子（软拷贝）文件。电子文件含有文本、表、图和其他形式的制图或

通过扫描读入的图形等。允许浏览相互链接的多媒体文档的网络超媒体系统正在急剧地扩大（见彩图I-2）。

- 计算机辅助制图和设计：在计算机辅助设计（CAD）中，交互式图形用来设计各种机械的、电气的、机电一体的和电子设备中的元件和系统，包括了诸如建筑物、汽车车体、飞机机身和轮船船体、大规模集成电路（VLSI）芯片、光学系统和电话与计算机网络的结构。有些时候，用户并不希望画出精确的零件图和装配图，而只是在线画草图或建筑蓝图。彩图I-8表现一个非专业用的三维设计程序范例，它被用来在木材堆积场定制一个有个人特色的PATIO DECK。然而，我们更常遇到的是与设计当中的元件或系统的计算机模型交互，以测试其结构、电气或热性能。这些模型通常由一个模拟器作解释，并将系统的行为反馈给用户，供下一轮交互设计与测试参考。当一个对象设计完毕之后，可以用工具程序对设计数据库进行后处理，生成零件清单，编制材料账单，确定用于切削或钻研工件的数控程序带等。

5

- 科学计算可视化和环境的模拟与动画：对于科学与工程可视化而言，计算机生成的动画影片和真实对象或模拟对象的时变行为的显示正变得日益流行（见彩图I-10）。我们可以用它们来学习抽象的数学实体，或研究许多现象的数学模型，例如，液体流动、相对论、核反应与化学反应、生理系统与组织功能以及机械结构在各种负荷下的变形等。另一个先进技术领域是交互式卡通。各种制作“平面”卡通片的简单系统已经很便宜，它们可以在两幅特定的“关键帧”之间内插生成一系列中间帧。越来越多的卡通人物将在计算机中以三维形状描述来建模，其运动将由计算机命令控制，而不是卡通艺术家的画笔来控制（见彩图D和F）。带有飞行徽标和诱人的视觉幻景的电视广告已经普及，其效果如同电影中的一流特技（见彩图I-12、I-13、II-18和G）。复杂的机制可用于物体的造型和光与阴影的表现。
- 艺术与商业：重叠以前的范畴是计算机图形在艺术和广告中的用途；这里，计算机图形用来产生能够表达信息并吸引注意力的图像（见彩图I-9、I-11和H）。个人计算机和诸如博物馆、交通港站、超级市场和旅馆，还有私人住宅中的电视图文和可视图文终端，提供了非常简单但富含信息的图像。这些图像使用户适应形势，做出选择，甚至“远程购物”，并与其他商业团体接触。最后，用于商业、科学和教学陈述的幻灯制作是图形的另一项成本有效的用途。使用传统的方法制作这样的材料，其人工成本正在急剧地上升。
- 过程控制：飞行模拟器或电子游戏使用户与真实或人造世界的模拟之间实现了交互；而许多其他应用却可以使人们与真实世界本身的某些方面相互作用。粉碎机、电厂和计算机网络中的状态显示器显示来自与关键系统元件相连传感器的数据值。例如，军事指挥员在命令与控制显示器上观看战场数据——车辆的数量与位置、作战火力、部队移动和伤亡，依此调整战术。机场的飞行管制员观看由计算机生成的有关其雷达范围内飞机标志的确认与状态信息，因而控制交通可以比使用未经解释的雷达数据的情形更加快速、准确。宇宙飞船控制员监视遥测数据，采取必要的修正措施。
- 制图学：计算机图形被用来从测量数据生成地理或其他自然现象的准确和图解的表述。其范例包括地图、地形图、用于钻井和开采的探矿图、航海图、气象图、等高线图和人口密度图。

1.4 应用分类

6

上节所列计算机图形的广泛应用，在许多方面各不相同，可以使用多种方法对这些应用进

行分类。第一种分类方法是根据被表述物体的类型（维数）和生成图像的种类。可能的组合范围如表1-1所示。

表1-1　按物体和图像划分的计算机图形的种类

物体类型	画　法	例　子
二维	线图	图2-1
	灰度图像	图1-1
	彩色图像	彩图I-2
三维	线图（或线条图）	彩图II-21~II-23
	具有各种效果的线图	彩图II-24~II-27
	标准的或具有各种效果的彩色图像	彩图II-28~II-39

有些用图形表现的对象显然是抽象的，而另外一些则是真实的。同样，图像可以是纯粹的符号（简单的二维图），也可以是纯粹的写实（静物的实况）。当然，同一个物体可以用不同的形式表达。例如，一块充满集成电路的印制电路板可以用许多不同的二维符号表达式或电路板的三维合成照片描述。

第二种分类方法是根据交互的类型。交互的类型决定了用户对物体及其图像的控制程度。这里，交互的范围包括：离线绘图，其预先定义的数据库由其他应用程序产生或物理模型数字化；交互绘图，由用户控制“提供参数，绘图，改变参数，再绘图”的循环；在用户的控制下实时地预定义或计算物体，并在其周围飞行，如同用于科学计算可视化和飞行模拟器的实时动画系统；交互设计，其中用户从一个空白屏幕开始，定义新的物体（通常用预先定义好的组件装配而成），然后移动物体得到希望的视图。

第三种分类方法是按照图像的作用，即图像本身是最终目的还是仅仅是通往最终目的的一个手段。例如，在制图、草图、光栅画图、动画和美工当中，绘画就是最终的产品；而在许多CAD应用中，绘画只是正在被设计或分析的物体的几何特性的表现形式。这里的画图或构图阶段是一个更大过程的重要却很小的一部分，它的目的是用一套集成的应用程序来创建和后处理公共数据库。

CAD中一个很好的例子是设计一块VLSI芯片。工程师使用CAD软件包进行芯片的初步设计。一旦布置好所有的门，她就让芯片经历数小时的模拟试用。例如，从第一次运行的结果，她知道芯片只能在时钟速度超过80ns（纳秒）时工作。由于机器的目标时钟速度为 50 ns，工程师调出初始布线，重新设计一部分逻辑，减少其级数。根据第二次模拟运行，她知道芯片在时钟速度低于60 ns时无法工作。她再次调出布线图，重新设计芯片的一部分。一旦芯片通过了所有的模拟测试，她会请求后道工序为制造厂生成关于设计和材料规格的信息数据库。例如，导体路径图和装配图。在这个例子中，芯片的几何表述产生的输出超出了图像自身的内容。实际上，屏幕上显示的几何形状的内容也许少于底层的数据库。

7

最后一种分类方法是依据物体和它们的图像之间的逻辑和时间关系。例如，用户在某一时刻只处理一幅图像（如绘图），或处理相关图像的一个时变序列（如运动动力学或更新动力学），或处理多个物体的构造集合（如许多含有多层装配图和分装配图的CAD应用）。

1.5　计算机图形学硬件与软件的发展

本书侧重于那些源于过去，至今仍在使用，并且将来大概还会继续使用的基本原理与技术。在本节中，我们简要地回顾计算机图形学的发展历史，说明当今系统的由来。关于该领域令人

兴趣盎然的演化过程在文献[PRIN71]、[MACH78]、[CHAS81]和[CACM84] 中已有较为完整的论述。编写硬件的发展史显然比软件的要容易一些，因为硬件的进展对该领域如何发展影响较大。因此，我们就先从硬件谈起。

早期的计算机在硬拷贝设备（如电传机和行式打印机）上粗略地画图。1950年麻省理工学院（MIT）开发的旋风式计算机（Whirlwind Computer）配置了由计算机驱动的CRT显示器作为输出，既供操作员使用，也供照相制作硬拷贝。在20世纪50年代中期开发的SAGE防空系统首先采用了命令与控制CRT显示控制台，在控制台上，操作员使用光笔（手持式可接收屏幕上目标发出亮光的点定位设备）识别目标。然而，现代意义上的交互式图形学起源于Ivan Sutherland开创性的关于Sketchpad绘图系统博士论文[SUTH63]。他引入了用于存储由标准组件简单复制构成的多层符号的数据结构，这是一种类似于使用塑料模板画电路符号的技术。他还开发了利用键盘和光笔实现选择、定位和绘制等交互技术，并提出了其他许多沿用至今的基本原理和技术。的确，许多Sketchpad中引入的特征在第7章中将要讨论的PHIGS图形软件包中仍可见到。

与此同时，计算机、汽车和飞机制造商们已经清楚地意识到，计算机辅助设计（CAD）和制造（CAM）对于自动绘图和需要大量绘图的工作具有巨大的潜力。通用汽车公司用于设计汽车的DAC系统[JACK64]和Itek公司用于设计透镜的Digitek系统，是表现工程中常见的迭代设计周期中图形交互卓有成效的先驱。到了20世纪60年代中期，已经出现了一些研究项目和商业产品。

由于那时的计算机输入/输出（I/O）主要用穿孔卡批处理完成，非常希望在交互式人机通
8 信上有所突破。交互式图形学作为“计算机的窗口”，成为大幅度缩短交互设计周期不可缺少的一部分。然而，结果不如想像的那样显著，交互式图形仍被划归于技术密集的组织所属的资源范畴。其原因如下：

- 图形硬件成本昂贵。未形成规模效益时，生产第一台商用计算机显示器成本达10万美元；而同时期一台汽车的成本为数千美元，一台计算机的成本为数百万美元。
- 需要大量昂贵的计算资源来支撑庞大的设计数据库、交互式图像操作以及通常对图形设计阶段生成的数据进行后处理的大量程序。
- 为新的分时系统编写大型交互式程序的困难。当时，对于那些习惯于批处理的FORTRAN编程员来说，图形和交互还是新事物。
- 有针对性的而且不可移植的软件。这些软件通常为特定的制造商生产的显示设备编写，编程中也未考虑现代软件工程的结构化系统和模块化原则。当软件不具有可移植性时，转移到新的显示设备就需要费时而且代价高昂的重新编程。

正是由于出现了基于图形的个人计算机，如苹果Macintosh计算机和IBM PC，才最终大幅度降低了软件和硬件的成本，使得数百万图形计算机作为“家电”进入了办公室和家庭；当计算机图形学萌生于20世纪60年代初期时，其开创者怎么也不会想到，具有图形交互功能的个人计算机竟会如此迅速地普及。

1.5.1 输出技术

20世纪60年代中期开发的一直到20世纪80年代中期还在使用的一种显示设备被称为向量显示器、笔划显示器、线条绘制器显示器或书法显示器。向量（vector）这个词，在这里被用作线条（line）的同义词；笔划（stroke）是短的线条，而字符则由一系列这样的笔划组成。我们将简略地回顾向量系统体系结构，因为许多现代光栅图形系统也使用类似的技术。一个典型的向量系统由显示处理器、显示缓冲存储器和显示屏（CRT）组成，显示处理器与中央处理器（CPU）相连，作

为其外围输入/输出（I/O）。缓冲器存储着由计算机生成的显示列表或显示程序，它包含了用端点（x, y）或（x, y, z）表述的点绘制命令和线绘制命令，以及字符绘制命令。图1-1表示一个典型的向量体系结构，内存中的显示列表体现为输出命令的符号表达式和它们的坐标（x,y）或字符值。

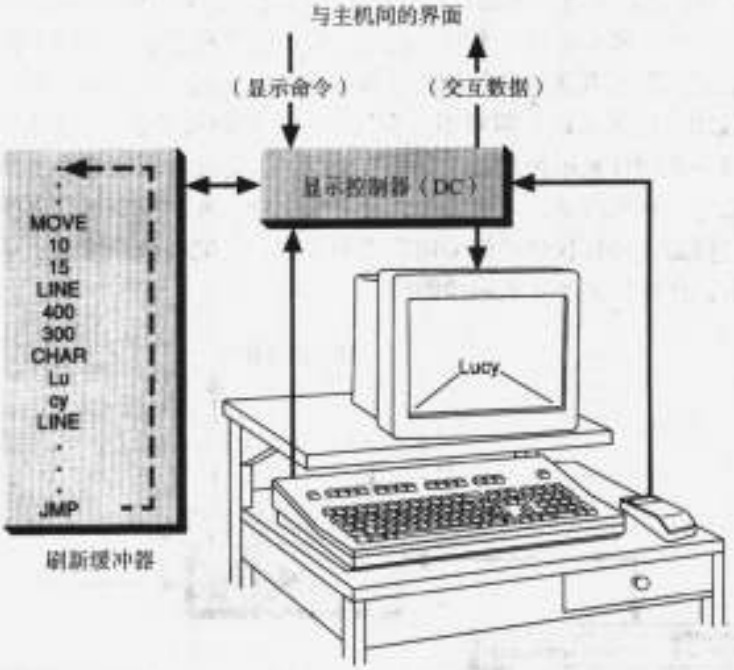


图1-1 向量显示的体系结构

绘制点、线和字符的命令由显示处理器解释。它向向量发生器发送数字和点坐标，然后向量发生器将数字坐标值转换为电子束偏转电路中的模拟电压，控制电子束落在CRT含荧光物质涂层上的位置（细节在第4章给出）。向量系统的本质是按照显示命令的任意顺序，将电子束从一个端点偏转到另一个端点，这种技术称为随机扫描（激光显示同样也利用了激光束随机扫描偏转技术）。由于荧光物质输出的衰减要持续数十或数百微秒，为了避免闪烁，显示处理器应以每秒至少30次（30 Hz）的频率循环执行显示列表，刷新显示器；因此，存放显示列表的缓冲器通常被称为刷新缓冲器。注意，在图1-1中，跳转指令使执行回到显示列表的第一行，形成周期性的刷新。

在20世纪60年代，能够达到30 Hz刷新频率的缓冲存储器和处理器价格昂贵，并且，要做到没有明显的闪烁，一个周期只能显示几千条直线。到了20世纪60年代末，直视储存管（direct-view storage tube, DVST）不再需要缓存和刷新处理，避免了闪烁。这是交互式图形实用化中关键的一步。DVST通过将图像一次写入的方式存储图像，写入图像时，电子束在涂有荧光物质的存储网格上以相对较慢的速度移动。小尺寸、自给自足的DVST终端比典型的刷新系统在价格上便宜一个数量级；此外，它非常适于低速（300~1200波特）电话与分时系统间的界面。DVST终端把许多用户和程序员带入交互式图形。

20世纪60年代末，硬件的另一个主要进展是将显示器与小型计算机捆绑在一起。在这样的配置中，中央分时计算机从刷新显示设备的繁重任务中解脱出来，特别是其中的用户交互处理和刷新屏幕上显示图像两项工作。小型计算机通常也运行应用程序，并且可以反过来连接到较

大的中央主机来运行大的分析程序。小型计算机和DVST两种配置都安装了数千台图形系统。也是在这一时期，显示处理器硬件本身变得更加完善，取代了图形软件中许多例行的耗时的工作。其中最突出的是在1968年发明的几何变换刷新显示硬件，它能够实时地对屏幕上的点和线进行缩放、旋转和平移，可以完成二维和三维剪裁，还可以生成平行投影和透视投影（见第6章）。

20世纪70年代初，基于电视技术发展起来的低价位光栅图形设备对于计算机图形学领域成长的贡献远远超过此前所有其他的技术。光栅显示器（raster display）将显示图元（如线、文字、填充涂色或图案区域）以其像素形式存储在一个刷新缓冲器中，如图1-2所示。在某些光栅显示器中，有一个硬件显示控制器（如图1-2所示），它接收和解释输出命令序列，类似于向量显示器。在简单、常用的系统（例如个人计算机）中，显示控制器仅以图形库软件包的软件组件形式出现，刷新缓冲器仅仅是CPU内存中的片段，它能够被图像显示子系统（常被称为视频控制器）读出，在屏幕上产生实际的图像。

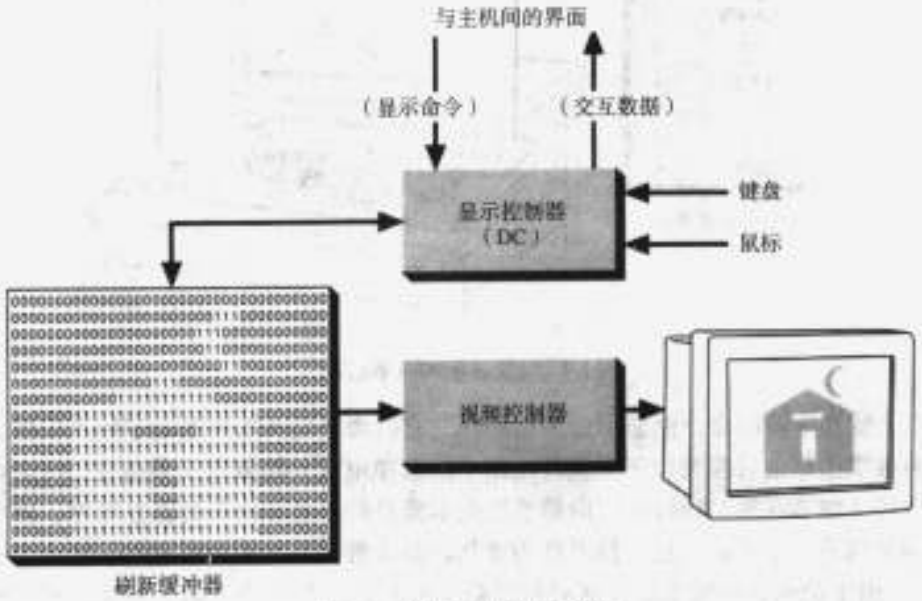


图1-2 光栅显示的体系结构

光栅显示器上的全部图像是由光栅（raster）形成的。光栅是一组互相平行的水平光栅线，
11 光栅线上每行有各自的像素；所以，光栅存储为一个代表了整个屏幕区域的像素矩阵。整幅图像由视频控制器按照从上到下然后再跳回顶部的顺序逐行扫描（如图1-3所示）。在每个像素上，电子束的亮度反映了像素的亮度；彩色系统中，三个电子束分别对应红、绿、蓝三原色，使之与每个像素值的三个颜色分量值相一致（见第4章和第13章）。图1-4说明了随机扫描和光栅扫描在显示一所房子的简单二维线条（图1-4a）时的差别。在图1-4b中，向量弧线段标上了箭头，表示电子束的随机偏转。虚线表示电子束的“空”偏转，因而偏转过程中，向量不画在屏幕上。图1-4c表示用矩形、多边形和圆弧绘制的未填充的房子，而图1-4d是一个填充

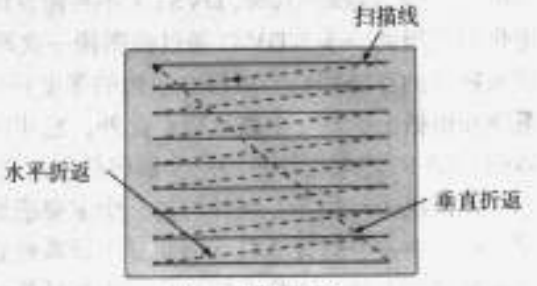
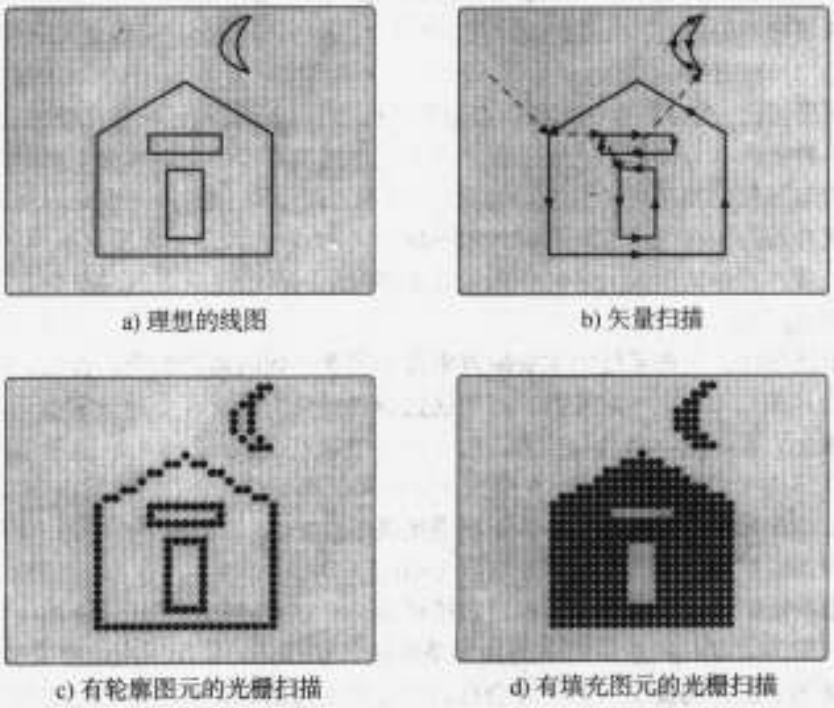


图1-3 光栅扫描

的版本。注意光栅扫描图像（图1-4c）和（图1-4d）中直线和圆弧的锯齿形状，我们将简短地讨论这些人为的视觉缺陷。



a) 理想的线图　b) 矢量扫描

c) 有轮廓图元的光栅扫描　d) 有填充图元的光栅扫描

图1-4 随机扫描与光栅扫描。我们用填充了灰色的圆角矩形代表屏幕的白色背景，图像用黑色画在背景上

在早期的光栅图形中，刷新是以电视的频率30 Hz进行的；现在，我们用60 Hz或更高的刷新频率，以避免图像的闪烁。在向量系统中，刷新缓冲器只存放了操作码和端点坐标值；而在光栅系统里，整幅图像，也就是1024行且每行1024个像素点，都必须直接保存到刷新缓冲器中。位图这个术语仍被广泛用于描述刷新缓冲器和与屏幕像素一一对应的像素值数组。位图图形比向量图形的优越之处在于图像的实际显示可采用廉价的扫描逻辑：实现有规则的重复光栅扫描与向量系统的随机扫描相比，要方便和便宜得多，因为向量系统的向量发生器必须以高精度保证电子束偏转的线性和重复一致性。

20世纪70年代初，便宜的固态随机访问存储器（RAM）被用于位图是一个突破，它导致了光栅图形成为占主导地位的硬件技术。二值（或称为单色）CRT使用黑白或黑绿两色绘图，有些等离子显示器则使用橙黑两种颜色。二值位图的每个像素占用单独一位，与一个分辨率为1024 × 1024像素相对应的完整位图仅有2^{20}位，即大约128 000字节。低端彩色系统每个像素有8位，允许同时显示256种颜色；较高级的系统每个像素占用24位，可供选择的颜色有1600万种；现在，甚至个人计算机也使用了每个像素占32位，屏幕分辨率达1280 × 1024的刷新缓冲器。如第4章所述，32位中的24位被用来表示颜色，余下的8位用于控制。除此之外，在第18章讨论的高端系统中，使用了同样屏幕分辨率，但每个像素占用96位[⊖]（或更多）的缓冲器。一个典

⊖ 在96位当中，有64位用于两个32位的颜色和控制缓冲器，来实现两幅图像的双缓冲功能：当缓冲器中的一幅图像处于刷新状态时，另一幅则处于内容更新状态。余下的32位缓冲器用于完成一种被称为“Z缓冲器”的硬件技术，该技术在创建真实感三维图像中被用来确定可见的表面（见第14章和第15章）。

型的每个像素占24位，分辨率为1280 × 1024的彩色系统需要3.75 MB的RAM，按现今的标准并不昂贵。严格地讲，“位图”这个术语只适用于每个像素一位的二值系统；对于每个像素多位的系统，我们使用更广义的术语“像素图”（pixmap，pixel map的缩写）。照一般的说法，像素图既可指刷新缓冲器的内容，也可指缓冲存储器的本身，所以当我们指的是实际缓冲存储器时，我们使用术语“帧缓存”。

12
~
13

与向量图形相比，光栅图形的主要优点是造价低和具有对显示区域填充颜色或图案的能力。区域填充是一种极为有力的信息交流手段，对于显示三维真实感图形尤为重要。此外，光栅图形的刷新过程与图像的复杂性（多边形的数目等）无关，因为硬件的速度足以保证在每个刷新周期中读出缓冲器内所有的像素。当刷新频率高于70 Hz时，大多数人感觉不到屏幕上的闪烁。与此相反，当缓冲器中的图元太多时，向量显示器发生闪烁。通常，短向量数目不超过几十万条时，不致出现闪烁。

与向量系统相比，光栅系统的主要缺点来自于像素表达的离散性质。首先，线和多边形这样的图元是用其端点（顶点）定义的，必须通过扫描转换为帧缓存中的像素单元。扫描转换是指这样一个概念，程序员以随机扫描方式输入图元中端点或顶点的坐标，这些图元信息再由系统简化为光栅扫描方式显示的像素。在个人计算机和低端工作站中，扫描转换通常由软件实现，微处理器CPU负责处理所有的图形。为了获得更高的性能，扫描转换可由专用硬件处理，包括用做辅助处理器或加速器的光栅图像处理器（RIP）芯片。

由于光栅系统中每个图元都必须经过扫描转换，所以实时动态显示所需的计算量比向量系统要大得多。首先，变换向量系统中的1000条线段，在最坏的情况下意味着变换2000个端点。在下一个刷新周期里，向量发生器硬件会自动地在变换过的新位置上重画这些线段；然而，在光栅系统中，不仅端点必须被变换（使用与向量系统相同的硬件变换单元），而且每一个被变换的图元还必须利用其变换得到的新端点进行扫描转换，确定其新的尺寸和位置。帧缓存中内容不必保留。当CPU同时负责端点变换和扫描转换时，只能实时变换少量图元。因此，在光栅系统中，需要采用变换和扫描转换硬件提高其动态性能。作为大规模集成电路（VLSI）技术不断进步的结果，现在即使在低端系统，硬件加速技术也是可行的。

光栅系统的第二个缺点来自光栅本身的性质。向量系统能够从CRT屏上的任何一点与其他点之间画连续的、平滑的直线（甚至光滑的曲线）；但光栅系统在理论上只能用光栅网格上的像素近似地描绘平滑的直线、多边形和诸如圆与椭圆那样的曲线图元的边界。它引起了锯齿或阶梯状这个大家熟悉的问题，见图1-4c和图1-4d。这种视觉人工痕迹是信号处理理论中一种被称为“走样”的错误采样的表现。当用离散采样点来逼近一个亮度急剧变化的连续函数时，就会出现这种现象。现代计算机图形学的理论与实践都要涉及灰阶系统和彩色系统的“反走样”技术。这些技术是使图元边缘邻近像素点上的亮度逐渐过渡，而不仅是将这些像素点的亮度值置为最大或零。对于这个重要主题的深入讨论，请参见第3章、第14章和第19章。

14

1.5.2 输入技术

过去这些年，输入技术也有了很大的改进。向量系统中笨拙、脆弱的光笔已被无处不在的鼠标（最早是由办公自动化的先驱Doug Engelbart于20世纪60年代中期开发出来的[ENGE68]）、数据输入板和贴在屏幕上的透明触敏膜所取代。甚至连那些不仅能够输入屏幕上的（x, y）坐标，而且还能输入三维甚至更高维数的输入值（自由度）的新式输入设备都变得很普遍（将在第8章中讨论）。声音通信也有令人激动的潜力，因为它允许不用手的输入、简单指令的自然输出以及反馈等。借助标准的输入设备，用户可以通过键入或画新的信息或指向屏幕上既存的信

息，来指定操作或图像成分。这些交互不需要编程知识，并且只需少量使用键盘：用户通过简单地选择菜单按钮或图标做出选择，通过检查选择项或在表格中键入少量字符来回答问题，在屏幕上粘贴预先定义的符号；通过指定一系列的端点绘图，端点间可用直线连接或用平滑曲线插值，通过在屏幕上移动光标涂色，以及用灰度的浓淡、色彩和各种图案填充由封闭多边形或轮廓线包围的区域。

1.5.3 软件的可移植性与图形标准

硬件技术的持续发展使图形显示器有可能从有针对性的特殊输出设备演化成为标准的人机界面。我们也许很想知道软件是否跟得上硬件的发展。例如，对于那些由过于复杂、麻烦而且昂贵的图形系统带来的早期困难，应用软件解决到了什么程度？其中的许多困难是由可供使用的最基本的图形软件造成的，通常这些软件要经过一个缓慢和漫长的过程才能达到成熟。我们已经从由制造商为其特殊的显示设备开发的低层次的设备相关软件包转向更高层次的设备无关的软件包。这些软件包能够驱动许多种类显示设备，从激光打印机和绘图仪到胶片记录仪和高性能实时显示器。使用与设备无关的软件包和高级编程语言的主要目的在于提高*应用程序的可移植性*。这种可移植性是以非常类似于那些与机型无关的高级语言（如Fortran、Pascal 和C语言）的方式提供的：把程序员从大量的设备特性中解脱出来，并提供预先完成的、针对各种处理器的语言特征。“程序员可移植性”也得到了增强，这使得程序员能够从一个系统转移到另一个系统，甚至从一个装置转移到另一个装置，而且看到熟悉的软件。

普遍意识到需要为这种与设备无关的图形软件包制定标准是在20世纪70年代中期。1977年ACM SIGGRAPH[⊖]委员会终于达成了*三维核心图形系统*（3D Core Graphics System，简称Core）规范[GSPC77]，1979年对该规范进行了细化[GSPC79]。本书的前3位作者积极参与了1977年Core的设计与成文工作。 15

Core规范起到了基本规范的作用。它不仅提供了许多方法，而且在ANSI（美国国家标准协会）和ISO（国际标准化组织）内部，也被用做官方（政府）标准项目的输入。第一个成为官方标准的图形规范是GKS（图形核心系统）[ANSI85b]。它是一个经过精心制作，由Core简化而成的标准。与Core不同的是，它只局限于二维。1988年GKS-3D[INTE88]，（一个GKS的三维扩展）成了官方标准；与此同时，一个更完善但也更复杂的图形系统，即PHIGS（程序员分层交互图形系统[ANSI88]），也成为官方标准。GKS支持把逻辑上有关联的图元（如直线、多边形和字符串）与它们的属性聚集在一起，称为*段*（segment），但这些段不可以嵌套。正如其名称所示，PHIGS支持三维图元的嵌套式多层分组，称为*结构*（structure）。在PHIGS中，为了实现动态运动，所有的图元，包括子结构的引用，都要经过几何变换（缩放、旋转和位移）。PHIGS也支持结构的保留数据库，程序员可以有选择地编辑这些结构；任何时候只要数据库发生了变化，PHIGS就会自动更新屏幕。PHIGS已经被扩展到带有现代的、在光栅显示器上对物体进行伪真实感绘制[⊜]的若干特性。这一扩展称为PHIGS + [PHIG88]。由于许多特性和规范的复杂性，PHIGS的实现表现为大量的软件包。PHIGS和PHIGS + 在有硬件支持其变换、裁剪和绘制功能时，运行最佳。

⊖ SIGGRAPH是Special Interest Group on Graphics（图形特别兴趣小组）的简称，它是计算机学会（Association for Computing Machinery, ACM）中的一个专业群体。ACM是计算机专业人员的两大专业团体之一，另一个团体为国际电气与电子工程师协会（IEEE）的计算机分会（IEEE Computer Society）。SIGGRAPH出版研究杂志，并主办一个年会发表和展出该领域的研究论文与设备。IEEE的计算机分会也出版图形学研究杂志。

⊜ 伪真实感绘制仅模拟物体对光线反射的简单光学定律，而真实感绘制则需要更准确地近似物体对光线的反射和折射。这样的近似计算量较大，但生成图像的质量更接近于摄影图片（见彩图E）。

本书将用一些篇幅讨论图形软件的标准。我们首先研究简单光栅图形软件包（Simple Raster Graphics Package，SRGP），它是一种仿照苹果计算机的Quickdraw整数光栅图形软件包[ROSE85]和麻省理工学院的XWindow System（X 视窗系统）[SCHE88A]的功能作为输出、仿照GKS和PHIGS的功能作为输入的软件。在看过这种低层光栅图形软件包的应用之后，我们再来研究这些软件包中用来在帧缓存中产生图元图像的扫描转换和剪裁算法。在建立起二维和三维几何变换以及三维平行投影和透视投影的数学基础之后，我们再研究一个功能更加强大的软件包SPHIGS（Simple PHIGS）。SPHIGS是PHIGS的一个子集，它以定义在一个独立于任何显示技
16 术的浮点、抽象和三维世界坐标系统中的图元为操作对象，并且支持一些简单的PHIGS + 功能。我们的讨论针对PHIGS和PHIGS + ，因为我们相信它们对交互式三维图形的影响比GKS-3D要大得多，特别是在更多地使用硬件支持实时变换和绘制伪真实感图像的情况下更是如此。

1.6 交互式图形学的概念框架

1.6.1 概述

图1-5所示的高级概念框架可被用来描述几乎任何一种交互式图形系统。在硬件层次上（图中未明确画出），计算机从交互设备接收输入，并向显示设备输出图像。软件有三个部分：第一部分是*应用程序*，它创建（图元），并将之存入应用模型或由应用模型取回（图元）。*应用模型*是软件的第二部分，代表了将在屏幕上显示的数据和物体。应用程序也处理用户的输入。它通过向*图形系统*发送一系列图像输出命令产生视图，这些命令既包括了将被观看对象的详细几何描述，也包括了描述这些对象将如何显现的属性特征。图形系统是软件的第三部分，负责从对象的细节描述产生实际的图像，并将用户的输入传递给应用程序，以供处理。

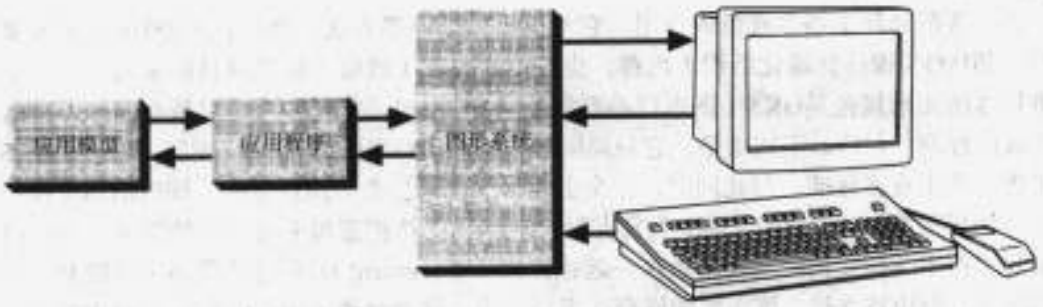


图1-5 交互式图形的概念框架

因此，图形系统是应用程序和显示硬件之间的中介，起着将应用模型中的物体转化为模型视图的*输出变换*作用。与此相对称，它还起着把用户的活动转变为应用程序的输入的*输入变换*作用，这些输入将使应用软件改变其模型和/或图像。一个交互式图形系统应用程序设计者的基本任务是，规定生成哪些类的数据项或物体，并将它们表示为图像，以及说明用户和应用程序之间如何交互，以创建和修改模型及其视觉表达。大多数程序员只涉及创建和编辑模型和处
17 理用户交互，并不实际创建视图，因为那是由图形系统处理的。

1.6.2 应用建模

应用模型捕捉所有的数据、物体和它们之间的相互关系，这些内容与显示和应用程序的交互部分相关，而且与非图形后处理模块有关联。后处理模块可以是分析一个电路或一个机翼中应力的瞬态行为、人口模型和气象系统的仿真和建筑物的价格计算。在标有画图字眼的一类应

用程序中，如MacPaint和PCPaint，程序的目的是通过让用户设置或修改每一个像素来产生图像。这里，不需要一个明确的应用模型，图像本身既是手段又是结果，而且被显示的位图或像素图起着应用模型的作用。

然而，还有一种更典型的等价应用模型，它通过数据和独立于特殊显示设备的过程描述的组合表达应用对象。例如，过程描述被用来定义分形，如20.3节所述。数据模型可以像数据点的数组那样简单，也可以像链表、网状数据结构或存储了许多关系的关联数据库那样复杂。我们常说将*应用模型*存入*应用数据库*，这两个术语在这里可以互换使用。模型通常存储图元的描述（二维或三维的点、线和多边形及三维的多面体和自由曲面），它们定义了物体中各组成部分的形状，物体的线型、颜色或表面纹理等*属性*，以及说明物体各部分如何组装到一起的*连接*关系与位置数据。

存储于模型中的物体在定义它们所需的固有几何量上可以有很大的差别。作为“任何物体皆为几何”类型的代表，一个如第7章所述的工业机器人几乎可以用多面体完整地描述。每一个多面体都是一些三维多边形面片的集合。这些面片在以公共顶点定义的公共边上互相连接，形成一个封闭体。一份电子表格则几乎没有固有的几何形状。相邻数据元素间的空间关系被存储下来，但每个元素在“纸”上的准确尺寸或位置却没有记录；取而代之的是，这些值由电子表格程序根据元素的内容动态地决定。作为“没有几何形状”类型的代表，一个存储了某人群的个人收入和年龄统计的人口统计模型不具备固有的几何形状。这些统计数据可以被进一步处理，导出一些几何解释，如二维图、散点图或柱状图等。

另一类没有固有几何形状的应用是工程和项目管理等领域中使用的有向图网络的处理。这些网络的内部关系可用反映节点如何连接的邻接矩阵，并附加一些节点和边的性质数据来表述。为了创建图网络的视图，该应用必须按照预先规定的格式导出一个布局安排。这一表述可以创建一次，然后逐次编辑。例如，就大规模集成电路（VLSI）而言，一次布局的计算超过数小时。这样，模型就同时包含了电路的非几何的和近乎纯几何的描述。作为另一种选择，如果一 18
个特定应用的布局能够足够简单和快速地导出，例如一个具有文字框和箭头的项目日程表，则布局可以在每一次数据改变时随时生成。

应用模型中的几何数据经常伴随着非几何的文本或数值特征信息，这些信息对于后处理程序或交互式的用户非常有用。例如，在CAD应用中，这种数据包括制造数据，价格与供应商数据，热、机械、电气或电子特性，以及机械或电气公差。

1.6.3 对图形系统描绘观察的内容

应用程序创建应用模型，或是在前面的计算阶段预先完成，例如超级计算机上的工程或科学仿真，或是作为交互会话的一部分，在显示设备上由用户逐步引导，选择元件和几何及非几何特性数据。用户可以随时要求应用程序显示已生成模型的视图（view）。（这里有意使用“视图”这个词，既指建模对象的某些几何特性的视觉形象，也指技术数据库中模型的某些子集的某些特性的二维表达。）

模型因应用而异，其产生与具体的显示系统无关。因此，应用程序必须把模型中要被观察的部分从内部的几何表示（无论是在模型中显式存储，还是随用随导出）转换为图形系统用来生成图像的过程调用或命令。这个转换过程分为两步。首先，应用程序遍历存储模型的应用数据库，用一些选择或查询准则抽出要观察的部分。然后，抽出的几何形状被转换为可以向图形系统发送的格式。选择的准则可以是自然界中的几何形状（例如，要被观察的模型的部分通过图形发生移动，等效于照相机的镜头摇动和景物拉伸操作），或者也可以类似于传统数据库的

查询准则（例如，列出撰写本书的时间进度表中所列的自1989年3月15日后的所有活动）。

在数据库遍历期间取出的数据必须是几何形状的数据，或者必须是已转换为几何的数据。这些数据可用图形系统直接显示的基本图元以及控制这些图元外观属性的形式记入图形系统。显示图元一般与几何模型中存储的图元相匹配。这些图元包括：二维的线、矩形、多边形、圆、椭圆和文本，以及三维的多边形、多面体和文本。像PHIGS + 那样先进的图形系统还支持其他图元，其中包括用高阶多项式定义的曲线和曲面。

如果模型存在着图形软件包不直接支持的几何图元，应用程序必须将它们简化为系统可以接受的形式。例如，如果图形系统不支持球体和自由曲面，应用程序就必须用图形系统可以处理的多边形网格“粘帖”或“镶嵌”在这些曲面上来近似它们。图形软件包支持的外观属性同
19 样也与模型中存储的属性相对应，如颜色、线型和线宽。除了图元和属性之外，诸如PHIGS这样的先进图形软件包还支持特定的几何变换（如缩放、旋转和定位元件），决定如何在三维空间中观看元件以及将有逻辑关联的图元、属性和变换分组，以便可以由一个命名结构引用。

图形系统一般包括一组与各种图元、属性和其他要素相对应的输出子程序。它们被收集在*图形子程序库*或*软件包*中，可被C、Pascal和LISP等高级语言调用。应用程序向这些子程序指定几何图元和属性，然后这些子程序驱动特定的显示设备，并使之显示图像。就像常规的I/O系统创建逻辑I/O单元使应用程序员不必面对大量的硬件和设备驱动细节一样，图形系统也要创建一个*逻辑显示设备*。因此，图形程序员可以忽略生成图像的哪部分工作是由显示硬件完成的，哪部分工作是由图形软件包实现的，或者显示器的坐标系是什么这些细节。这种显示设备的抽象既适合于图像的输出，也适合于通过逻辑输入设备进行的交互。例如，鼠标、数据输入板、触摸板、2D游戏杆或轨迹球都可以被视为返回屏幕坐标(x,y)的位置的定位逻辑输入设备。应用程序可以要求图形系统对输入设备进行*采样*，或者在一个特定的点等待，直到发生了用户操作处于等待状态设备的*事件*。借助从采样或等待事件获取的输入值，应用程序可以处理用户交互，这些交互操作改变模型、显示或操作模式。

1.6.4 交互处理

典型的应用程序交互处理方法为*事件驱动循环*。它可以被简单地看做是一台有限状态机，具有一个中央等待状态，用户输入事件使之转移到其他状态。处理一条命令可能需要一连串具有相同格式事件的嵌套循环，它们拥有各自的状态和输入转移。一个应用程序也可能通过随时查询它们的数值对输入设备（例如定位器）进行采样；然后程序使用返回值作为处理过程的输入。处理过程也改变应用程序、图像或数据库的状态。事件驱动循环可以用下面的程序伪代码来表示：

```
产生初始显示，由应用模型适当地导出
while (!退出){  /*用户没有选择“退出”操作*/
    使选择命令有效
    /*程序停顿于“等待状态”，直至用户干预*/
    等待用户选择
    switch(选择){
        执行选择以完成命令或执行已完成的命令，
          按要求更新模型和屏幕
    }
}
```

让我们更详细地查看应用程序对输入的反应。应用程序通常以一种或两种模式响应用户的
20 交互。首先，用户的动作可能仅要求屏幕更新，例如，通过点亮选中的对象或通过生成新的选

择菜单。那么，应用只需要更新它的内部状态，并调用图形软件包去更新屏幕；而不必更新数据库。然而，如果用户要求改变模块，例如增加或者删除组件，应用就必须更新模型，然后调用图形软件包从模型去更新屏幕。或者遍历整个模型重绘图像，或者应用更复杂的增量更新算法，有选择地更新屏幕。如果没有模型的变化，屏幕上显示的对象就不会发生明显的变化，理解这一点非常重要。屏幕的确是计算机的窗口。在其中，用户通常操作的不是图像，而是字面和象征意义上隐藏在图像后面的模型。只有在绘画和图像增强应用中，图像和模型才是等同的。所以，应用程序的工作是解释用户的输入。图形系统没有责任在最初或在与用户的交互中，建立和修改模型；它的惟一的工作仅仅是从几何描述中创建图像，并读入用户的输入数据。

事件循环模型，尽管对现在计算机图形应用来说很基本，仍有一定的局限性。其人机对话是一种顺序乒乓模式，用户动作与计算机反应轮流进行。将来，我们可能期望看到更多的并行交谈，利用多种通信信道（例如图形和语音）同时进行输入和输出。无须顾及编程语言的结构，进行这种自由形式交谈的形式目前尚未确立，这里我们将不做更多的讨论。

1.7 小结

图形界面已经取代了文本界面，成为人机交互的标准手段。在商业、科学、工程和教育等许多领域中，图形也已成为一项交流思想、数据和趋势的关键技术。利用图形，我们可以创建人工现实，进行基于计算机的“探险”，用自然和直观的方法检验物体和现象，这种方法则利用了我们在视觉模式识别中高度发达的技巧。

直到20世纪80年代后期，大量的计算机图形应用还只是处理二维对象，三维应用相对稀少，这既是因为三维软件本来就比二维软件复杂得多，也是因为绘制接近真实的图像需要大量的计算能力。因此，直到最近，用户与三维模型和伪真实感图像间的实时交互，只是在一些带有专用图形硬件并且非常昂贵的高性能工作站上才获得使用。担负发展廉价微处理器和存储器责任的VLSI半导体技术的引人注目的进步，导致了20世纪80年代早期基于二维位图图形的个人计算机的出现。同样的技术在不到10年之后，已使得仅用几个芯片制成的图形子系统成为可能，该子系统能够绘制含有数千个多边形的复杂物体的彩色光照图像，实现了实时三维动画。图形子系统可以作为三维加速器附加在工作站甚至是使用普通商业芯片的个人计算机上。很明显，三维应用的爆炸性成长将会与当前的二维应用的成长形成并存局面。而且，在本书1982年版中 21
还认为是非同寻常的照片真实感绘制，现在已经成为当前技术的一部分，可在图形软件和日益增多的图形硬件中使用。

实现有效的图形交流的首要任务，无论是二维还是三维，在于建立我们想要产生图像的对象的模型。图形系统在应用模型和输出设备之间，起着中介的作用。应用程序负责根据用户的交互作用创建和更新模型。在生成对象的视图和向应用传递用户事件的过程中，图形系统完成容易理解的常规工作。需要指出的是，尽管在撰写本书时，将建模与图形相分离是可以接受的现实；但我们的关于建模的章节（第11、12、20章）和动画的章节（第21章），以及日益增多的、有关各种基于物理建模的文献，表明图形已远远超出了绘制和交互处理的范畴。图像和动画不再仅仅是科学和工程中的图示，它们已成为了科学和工程的内容中的一部分，并且影响着科学家和工程师们如何管理他们的日常工作。

习题

1.1 列出在你的“知识工作”中经常使用到的交互式图形程序，例如书写、计算、绘图、编程

和调试等。其中哪些程序也可以在文字终端上实现?

1.2 词组“观感”已被广泛用于图形程序的用户界面中。详细列出你喜爱的字处理或窗口管理程序的图形界面的主要组件的外观，如图标、窗口、滚动条和菜单等。列出这些窗口小部件所需图形能力的种类。你看到有什么机会将颜色和三维绘制应用于这些外观吗？例如，如何使杂乱的办公室在空间上更好地隐喻，以易于组织和存取信息，而不再是一个“凌乱的桌面”？

1.3 与习题1.2类似，你看到有什么机会用动态图标增强或取代现有桌面隐喻中的静态图标?

1.4 利用图1-5所示的概念模型作为引导，将你喜爱的图形应用拆解为主要的模块。有多少模块实际是处理图形的？又有多少是处理数据结构生成与维护的？有多少是处理计算和仿真的?

1.5 “仿真”与“动画”两个术语在计算机图形中经常一起使用，有时甚至可以互换。在显示某些物理的或抽象的系统的行为（或结构）随时间变化时，这是自然的。请构造一些能够体现可视化优点的示例系统，确定仿真的形式和运行方法。

1.6 作为习题1.5 的变形，请为科学、数学或工程领域的一个重要主题创建一个图形“探索”的高级设计。讨论交互顺序如何工作以及用户试验中应具备哪些工具。

1.7 考虑一幅含有10000条互不相连的1英寸长向量的图像。请将其向量显示表占用的存储量与
22 显示同一图像的1024 × 1024二值光栅图像所占用的存储量相比较。假定在向量显示表中，定义“向量图”的“操作码”为8位，并用4个10位（即6个字节）来存储一个向量。在光栅显示中，需要的存储量与向量的数目和尺寸、每个像素的位数及分辨率有什么关系？两种形式所需刷新存储器容量间的相对比值是多少?

1.8 不看第3章，写出一个直接的算法用于扫描转换一条位于第一象限的直线。

1.9 走样是一个严重的问题，它产生难看的甚至令人误解的视觉失真。讨论在哪些情况下，会或不会发生走样。讨论使锯齿效应最小的各种方法，并解释这些补救措施的“代价”可能
23 是多大。

第2章 简单光栅图形软件包（SRGP）的编程

在第1章，我们已经介绍了采用不同的硬件技术生成屏幕上图像的向量显示器和光栅显示器。因为光栅显示器有许多适合现代应用的特点，所以它成了目前的主流硬件技术。首先，光栅显示器既可以用同一种颜色均匀填充区域，也可以用具有两种或两种以上颜色的重复图案填充区域；而向量显示器最多只能用空间中平行的向量序列来模拟区域填充的效果。第二，光栅显示器存储的图像易于操作：既可对单独的像素进行读或写的操作，也可拷贝或移动图像中的任意部分。

我们首先要讨论的图形软件包SRGP（Simple Raster Graphics Package），是与设备无关的软件包，它利用了光栅显示技术。SRGP图元（直线、矩形、圆、椭圆和文本串）的组成类似于流行的Macintosh QuickDraw光栅软件包和X Window System中的Xlib 图形软件包。另一方面，它的交互处理功能只是用于显示三维图元的图形软件包SPHIGS（将在第7章介绍该软件包）的一部分。SPHIGS （Simple PHIGS）实际上是可同时用于向量和光栅硬件标准的图形软件包PHIGS（Programmer's Hierarchical Interactive Graphics System）的简化版本。尽管SRGP和SPHIGS是专为本书写的，但它们也是主流图形软件包的精髓，这里所学到的技术可以立即应用于其他商业软件包。在本书中，我们将介绍这两个软件包，有关它们的完整信息，请参考随软件包发行的用户手册。

我们从考察应用软件在屏幕上绘制图形所进行的操作入手来讨论SRGP：图元的规格以及影响图像的属性。（由于图形打印机的信息显示原理基本类似于光栅显示器，所以此处我们仅考虑光栅显示器，而图形打印机将在第4章中介绍。）然后，我们将学习如何使用SRGP的输入功能使应用程序具有较好的交互性。接着介绍仅适合光栅显示器的像素操纵的实用工具。最后，我们将讨论整数光栅图形软件包（如SRGP）的局限性。

虽然在讨论SRGP时，我们假设它控制整个屏幕，但是我们是在窗口环境中（见第10章）设计软件包，因此可以将窗口的内部看做是虚拟屏幕，进而可以控制窗口的内部。因而应用程序员不必关心在窗口管理器控制下的运行细节。

2.1 用SRGP画图

2.1.1 图形图元的规格

用SRGP这样的整数光栅图形软件包绘图类似于在方格纸上绘图。在常规的显示器中坐标网格的变化范围是每英寸包含80~120个点，在高分辨率显示器中的变化范围是80~300个点。显示器的分辨率越高，图形显示效果越好。图2-1给出了在SRGP的整数笛卡儿坐标系统下的显示屏幕。SRGP的像素位于网格线的交点。

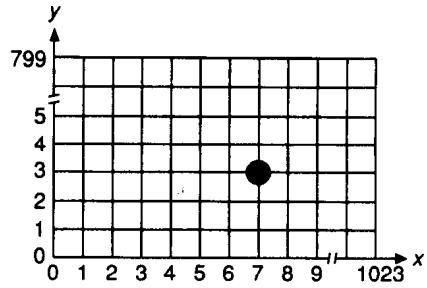


图2-1 一个宽1024像素、高800像素的显示屏幕的笛卡儿坐标系统，图中标出了像素（7,3）

原点（0, 0）位于屏幕的左下角；*x*轴正向向右增加，*y*轴正向向上增加。右上角像素点坐标为（width-1, height-1），其中，width（宽）和height（高）是与设备相关的屏幕尺寸。

在方格纸上，我们可以在任意两点之间画一条连续的线；然而，在光栅显示器上，我们只能在网格点间画线，而且只能用在直线上的点或最靠近直线的点来近似代表这一直线。同样，像多边形区域或圆这样的填充图形由其内部的和边界上的网格点像素生成。因为指定直线或封
26 闭图形的每一像素非常麻烦，所以图形软件包只要求程序员指定图元，如给定了顶点的直线和多边形，然后由软件包利用扫描转换算法（将在第3章讨论）实现具体的细节。

SRGP支持一个基本的图元集合：直线、多边形、圆、椭圆和文本。[⊖] 为了指定一图元，应用软件需要将定义图元形状的坐标传给SRGP中恰当的图元生成程序。当指定的点位于屏幕的矩形区域外时也是合法的，但只有位于屏幕内的图元部分是可见的。

1. *直线和多边形*

以下SRGP程序画一条从点（*x*1, *y*1）到（*x*2, *y*2）的线段：

```
void SRGP_lineCoord (int x1, int y1, int x2, int y2);
```
[⊖]

因此，要画一条从（0, 0）到（100, 300）的线段，我们只需简单地调用

```
SRGP_lineCoord (0, 0, 100, 300);
```

用线段顶点来代替单个的*x*，*y*坐标是很自然的想法，因此SRGP还提供了如下的线段绘制函数：

```
void SRGP_line (point pt1, point pt2);
```

这里，“point”是已定义的结构类型，该结构包含两个整数，分别用于表示点的*x*，*y*坐标值。

```
typedef struct {
    int x, y;
} point;
```

连接一系列点的一组线段称为折线。尽管通过反复调用线段生成程序可以生成折线，SRGP还是对折线进行了特殊处理。SRGP中有两个折线生成程序，类似于线段生成程序的参数形式，该程序将数组作为参数。

```
void SRGP_polyLineCoord (int vertexCount, int *xArray, int *yArray);
void SRGP_polyLine (int vertexCount, point *vertices);
```

其中“*xArray*”，“*yArray*”和“*vertices*”是指向用户定义的数据数组的指针，分别指向整数数组和点数组。

折线生成程序中的第一个参数通知SRGP输入的顶点数。在第一个调用程序中，第二和第三个参数分别是输入点的*x*坐标数组和*y*坐标数组。折线从顶点（*xArray*[0], *yArray*[0]）到
27 （*xArray*[1], *yArray*[1]），到（*xArray*[2], *yArray*[2]）等。这种形式非常方便，例如，当绘制的数据是位于轴上的标准集时，其中*xArray*是预先决定的独立变量值集，*yArray*是用户输入或计算得出的数据集。举个例子，让我们来绘制一个经济分析程序的输出，该程序计算每个月的交易数据，并将其存储在*balanceOfTrade* 数组中，该数组包含12个整数数据。我们从点（200,200）开始绘制，为了能够看出连续两点之间的区别，在*x*轴上以10个像素点为间距绘制每一点。因

⊖ 专门用来画单独一个像素或一组像素的过程在SRGP参考手册里有所描述。

⊖ 我们使用C语言，遵从下面的排版习惯。C语言的关键字和内置类型使用粗体，用户定义的类型使用标准字体。常量符号使用大写字母，变量使用斜体。注释使用“/* */”表示，伪代码使用斜体。为了简单，我们省略了那些显而易见的常量和变量的声明。

而，我们生成一个整数数组*months*来表示12个月，并将所需的*x*值200, 210, …, 310存入数组中。类似地，*y*方向上的12个值均增加200。可以用以下代码绘制出图2-2中的图形：

```
/* 画坐标轴 */
SRGP_lineCoord (175, 200, 320, 200);
SRGP_lineCoord (200, 140, 200, 280);

/* 画数据 */
SRGP_polyLineCoord (12, months, balanceOfTrade);
```

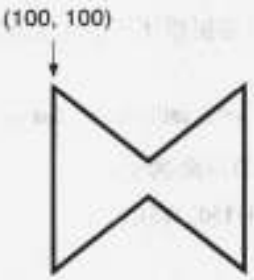
我们也可以将*x*和*y*坐标值对作为点的定义，将点数组传给SRGP，通过调用第二种形式的折线绘制程序来绘制通过这些点的图形。通过调用如下的折线生成程序：

```
SRGP_polyLine (7, bowtieArray);
```

我们生成了图2-3中蝴蝶结。图2-3中的表格给出了*bowtieArray*的定义。



图2-2 绘制数据点列



	x	y
0	100	100
1	100	60
2	120	76
3	140	60
4	140	100
5	120	84
6	100	100

bowtieArray

图2-3 绘制折线

2. 标记和多点标记

在图形的数据点处设置标记（例如，点、星号或圆）通常是十分有用的。因而，SRGP提供了与直线和折线生成程序相配套的标记函数。以下程序在点（*x*, *y*）处生成标记符号：

```
void SRGP_markerCoord (int x, int y);
void SRGP_marker (point pt);
```

可以用2.1.2节介绍的方法改变标记的风格和大小。调用以下程序中的任意一个都可以在一组点 28
上生成相同的标记。

```
void SRGP_polyMarkerCoord (int vertexCount, int *xArray, int *yArray);
void SRGP_polyMarker (int vertexCount, point *vertices);
```

所以，调用以下过程可以在图2-2中的图形上生成标记而得到图2-4。

```
SRGP_polyMarkerCoord (12, months, balanceOfTrade);
```

3. 多边形和矩形

为了绘制一多边形，我们可以定义一条首尾顶点相同的折线（和图2-3中一样），或者调用以下SRGP函数：

```
void SRGP_polygon (int vertexCount, point *vertices);
```

这一调用将自动在首顶点和尾顶点之间画一条线段。现在只需六个点，我们就可以画出图2-3中的蝴蝶结。

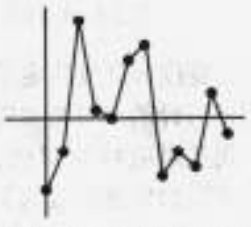


图2-4 用标记绘制数据点列

SRGP_polygon (6, *bowtieArray*);

任何一个矩形（rectangle）可以定义为有四个顶点的多边形，而只需两点（左下角和右上角），就可以利用SRGP的“矩形”图元画出一个正矩形（边平行于屏幕边界）。

void SRGP_rectangleCoord (**int** *leftX*, **int** *bottomY*, **int** *rightX*, **int** *topY*);
void SRGP_rectanglePt (point *bottomLeft*, point *topRight*);
void SRGP_rectangle (rectangle *rect*);

其中的“rectangle”记录存储了待绘制矩形的左下角和右上角坐标：

typedef struct {
  point *bottomLeft*, *topRight*;
} rectangle;

29

以下调用可以画出一个宽101个像素，高151个像素的正矩形：

SRGP_rectangleCoord (50, 25, 150, 175);

为了由坐标数据创建矩形和点SRGP提供了下面的函数：

point SRGP_defPoint (**int** *x*, **int** *y*);
rectangle SRGP_defRectangle (**int** *leftX*, **int** *bottomY*, **int** *rightX*, **int** *topY*);

调用以下函数可以生成我们所举的矩形例子：

rect = SRGP_defRectangle (50, 25, 150, 175);
SRGP_rectangle (*rect*);

4. 圆和椭圆

图2-5给出了由SRGP绘制的圆弧和椭圆弧。因为圆是椭圆的特殊情况，所以不论是圆或椭圆，也不论它是否封闭，我们均用椭圆弧代表。SRGP只能画主轴平行于坐标轴的标准椭圆。

尽管在数学上有许多方法定义椭圆弧，但是对于程序员来说，用椭圆的外接正矩形定义它是比较方便的（如图2-6所示）；这一正矩形称为包围盒或范围。

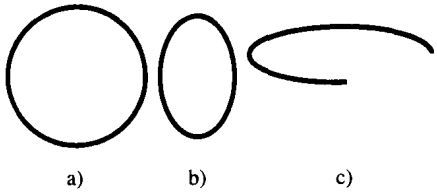


图2-5 椭圆弧

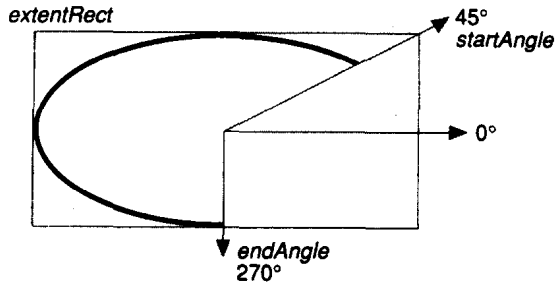


图2-6 界定椭圆弧

范围的宽度和高度决定了椭圆的形状。弧的起始角和终止角决定曲线是否封闭。为方便起见，使用逆时针的矩形角来度量每一角度，*x*轴的正向对应0°，*y*轴的正向对应90°，从原点到右上角的对角线对应45°。显然，仅当范围是正方形时，矩形角等价于圆弧角。

30

一般椭圆的生成函数如下：

void SRGP_ellipseArc (rectangle *extentRect*, **double** *startAngle*, **double** *endAngle*);

2.1.2 属性

1. 线型和线宽

图元的外观由它的属性[⊖]控制，在SRGP中，直线、折线、多边形、矩形和椭圆弧的属性包括线型、线宽、颜色和画笔类型。

属性是模态上设定的，即属性是一种全局状态变量，它将一直保持原值直至被显式地改变。图元定义后，将使用当时有效的属性值来绘制；因此改变属性值决不会改变以前创建的图元——它仅影响属性值改变后才定义的图元。采用模态属性十分方便，因为程序员不必为每一个图元都设定长长的参数列表，通常一个生产系统中可能有几十种不同的属性。

调用以下函数可以设置线型和线宽。

void SRGP_setLineStyle (CONTINUOUS / DASHED / DOTTED/... lineStyle);[⊖]
void SRGP_setLineWidth (**int** *width*);

线的宽度由屏幕单元度量，即用像素度量。每一个属性都有一个默认值：线型的默认值是CONTINUOUS（连续线），线宽为1。图2-7显示了具有不同线型和线宽的直线。图2-8给出了生成图2-7中图形的代码。

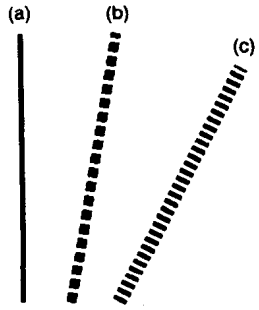


图2-7 不同线型和线宽的直线

```
SRGP_setLineWidth (5);
SRGP_lineCoord (55, 5, 55, 295);        /* 直线 a */

SRGP_setLineStyle (DASHED);
SRGP_setLineWidth (10);
SRGP_lineCoord (105, 5, 155, 295);      /* 直线 b */

SRGP_setLineWidth (15);
SRGP_setLineStyle (DOTTED);
SRGP_lineCoord (155, 5, 285, 255)       /* 直线 c */
```

图2-8 生成图2-7中图形的代码

我们可以认为线型是一个位掩码，使SRGP在对图元做扫描转换时，能够选择性地写像素。掩码中的零表示不写该像素，而保留存储在缓存中像素的原值。可以认为该像素是透明的，所以能看到底下的内容。因而CONTINUOUS对应的串位值均为1，DASHED对应的串为1111001111001111…，虚线长度为透明部分的两倍。

每一属性都有一个默认值；例如，线型的默认值是CONTINUOUS，线宽的默认值是1，等等。在前面的例子中，我们没有为绘制的第一条线段设置线型，因而我们使用的是线型的默认值。然而，在实际的应用中，假定当前的属性值是不安全的，在后面的过程中，我们将明确地设定每一属性，使程序标准化，便于调试和维护。在2.1.4节中，我们将看到，对于程序员来说，为每一个过程显式地保存和恢复属性值是比较安全的。

可为标记图元设置的属性有

void SRGP_setMarkerSize (**int** *markerSize*);
void SRGP_setMarkerStyle (MARKER_CIRCLE / MARKER_SQUARE/... markerStyle);

⊖ 这里对于SRGP属性，特别是不同属性的互相作用的描述经常缺少足够的细节。之所以缺少细节，是因为一个属性确切的效果是它的实现的一个函数，并且由于性能的原因，在不同的系统中使用了不同的实现；如果要进一步了解这方面的细节，请查阅专门的参考手册。

⊖ 在这里和以后的行文中，我们使用一个简化的符号。在SGRP里，这些符号常量实际上是枚举数据类型“lineStyle”的一个值。

标记的大小规定了包含该标记的正方形边的像素长度。参考手册中给出了全部标记类型的集合。图2-4中所示的圆类型是标记的默认值。

2. 颜色

目前所给出的属性只影响SRGP的一部分图元，但是整型颜色属性将影响全部图元。十分明显，颜色属性的意义很大程度上依赖于支持它的硬件设备；任一系统上的两个颜色值就是0和1。在二值显示系统中，这些颜色的外观很容易预测——对于黑白设备，颜色1代表像素是黑色的，颜色0代表像素是白色的。对于绿黑设备，颜色1代表像素是绿色的，颜色0代表像素是黑色的，等等。

整型颜色属性并不直接指定颜色值；更确切地说，它是SRGP*颜色表*的索引，该表中的每一项定义一个程序员不需要知道的颜色或灰度值。在颜色表中有2^d个条目，其中d是帧缓存的深度（每一像素的存储位数）。在二值设备上，颜色表固化在硬件上；然而，在多色系统中，SRGP允许应用程序修改颜色表。第4、17和21章将讨论颜色表提供的一些间接应用。

31 ~ 32

应用程序可以有两种方法确定颜色。如果该应用程序强调与硬件无关，则应当直接使用0、1确定颜色值，这样应用程序可同时在二值和彩色显示设备上运行。如果应用是颜色支持的，或者是为特定的显示设备编写的，那么它可以使用由SRGP支持的与实现相关的*颜色名*。这些名字都是符号常量，它们指示在特定设备的默认颜色表中标准颜色的位置。例如，黑白设备提供两个颜色名COLOR_BLACK（1）和COLOR_WHITE（0）；在本章的示例代码段中，我们将使用这两个值。注意对于修改颜色表的应用程序来说，颜色名字是没有用的。

可以通过调用以下函数选定颜色：

void SRGP_setColor (**int** *colorIndex*);

2.1.3 填充图元及其属性

包含区域的图元（所谓的*区域定义图元*）可以用两种方法绘制：*描绘轮廓*或者*区域填充*。前一节给出的程序生成的是前一种类型的图形：轮廓封闭，内部没有填充。SRGP区域定义图元的填充版本绘制不含轮廓线的区域内部像素。图2-9给出了SRGP的填充图元的种类，包括填充椭圆弧，即*薄片*（pie slice）。

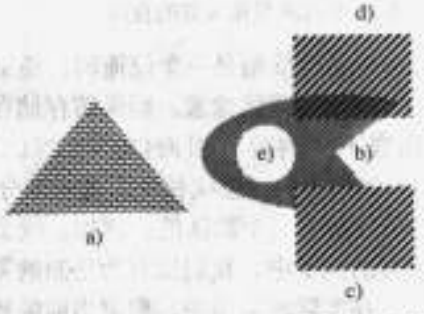


图2-9 填充图元。a)~c) 不透明的位图模式，d) 透明的位图模式，e) 实区域

注意到SRGP并不画出区域外轮廓，如一个像素宽的实线边界，所以需要绘制轮廓的应用程序必须再显式地画出它。是应该画出区域定义图元边界上的像素，还是仅画出严格位于内部的像素是一个敏感的问题。这一问题将在3.5节和3.6节详细讨论。

33

为了生成填充多边形，我们使用函数SRGP_fillPolygon或者SRGP_fillPolygonCoord，它们的参数与非填充版本相同。用同样的方法可以定义其他区域填充图元，这时要在它们原有的函数名前添加前缀“fill”。因为多边形可能是凹的或者甚至自相交，所以我们需要一个规则来规定什么区域是填充内部并因此应该被填充，什么区域是填充外部。SRGP多边形遵循*奇偶规则*。为了决定一个区域位于一个给定多边形的内部还是外部，在这个区域内任意选择一个点作为测试点；接下来，从这个点向任意方向画一条射线，这条射线不经过任何顶点，如果这条射线与

这个多边形的轮廓相交次数为奇数，这个区域被认为在其内部（见图2-10）。

当绘图的时候，SRGP并不对每个点都进行这种测试；它利用第3章中介绍的优化多边形扫描技术，在这种技术中，奇偶规则能非常有效地运用于相邻的整行像素，无论这些像素是位于外部，还是位于内部。奇偶射线相交测试也可用于关联拾取（pick correlation），用来确定用户用光标选中的对象，这将在第7章中进行论述。

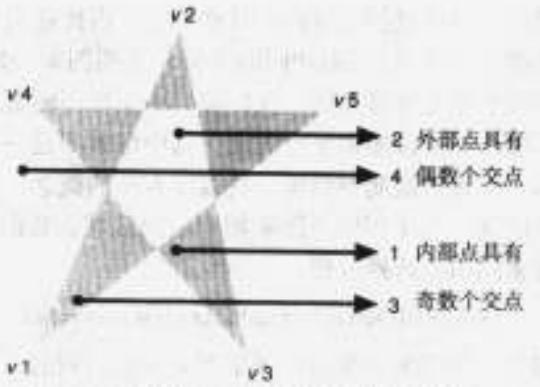


图2-10 确定多边形内部区域的奇偶规则

1. 区域填充类型和填充图案

利用以下函数，填充类型属性能用四种不同的方式来控制一个被填充图元内部的显示：

```
void SRGP_setFillStyle (
    SOLID / BITMAP_PATTERN_OPAQUE / BITMAP_PATTERN_TRANSPARENT /
    PIXMAP_PATTERN  drawStyle);
```

第一个选项SOLID利用当前的颜色值来产生一个均匀填充的图元（图2-9e，颜色被设成COLOR-WHITE）。接下来的两个选项BITMAP_PATTERN_OPAQUE和BITMAP_PATTERN_TRANSPARENT，用规则的非实心图案来填充图元，前一种用当前的颜色或者另一种颜色重写该图案F的所有像素（图2-9c），后一种用当前的颜色重写这个图元下面的某些像素，而让其他像素可透视（图2-9d）。最后一个选项PIXMAP_PATTERN，写图案时可包含任意数量的颜色，总是处于不透明的模式。

位图填充图案是一个以0和1为元素的位图矩阵，通过调用以下函数从可用的图案表中选取： 34

```
void SRGP_setFillBitmapPattern (int patternIndex);
```

该图案表中的每一个条目中存储一个惟一的图案；由SRGP提供（参见参考手册）的图案表包括灰度色调（范围从黑色到白色）和各种各样的规则和随机图案。在透明方式中，这些图案通过如下方式产生。把图案表中的任何图案都当成一个小位图（比如8×8），在需要的时候通过重复拼贴来填充图元。在一个二值显示系统中，位图图案中值为1的地方显示当前颜色值，而值为0（孔）的地方保留原来颜色，从而使图案有一种透明的效果。这样，位图图案对透明方式中的图案起了“内存可写掩码”的作用，就像直线和边界图元的线型位掩码一样。

在更普遍应用的BITMAP_PATTERN_OPAQUE模式下，1置为当前颜色，但是，0置为另一种颜色，背景色事先通过以下函数设置：

```
void SRGP_setBackgroundColor (int colorIndex);
```

在二值显示器上，OPAQUE方式下的每一种位图图案仅仅能产生两种不同的填充图案。例如，如果当前颜色被设置成黑色（背景色是白色），大部分为1的位图图案能在黑白显示器上产生暗灰色填充图案，如果当前颜色被设置成白色（背景色是黑色），则可产生亮灰色填充图案。在一个彩色显示器中，任一种前景和背景颜色的组合可以被用来实现两种色调的变化。二值显示系统中的一个典型应用程序通常在设置前景色的同时设置背景色，因为如果这两种颜色是一样的，不透明的位图模式是不可见的；应用程序可以生成一个SetColor过程，在系统设置前景色的时候，自动对照前景色来设置背景色。

图2-9是通过图2-11中所示的代码片段产生的。具有双色调位图图案的优点是颜色不被明确

规定，而通过颜色属性有效地确定，因此能由任何颜色组合生成。它的缺点是仅仅可以产生两种颜色，这也是SRGP同时支持像素图图案的原因。通常，我们喜欢采用一个明确的图案以多种颜色填充屏幕上的一块区域。一个位图图案是一个可以用来平铺图元的小位图，同样一个小的像素图能被用来平铺图元，其中像素图是一个指向颜色表索引的图案矩阵。因为在像素图中每一个像素被明确设置，所以没有孔的概念，因此在透明填充模式和不透明填充模式之间不存在区别。为了用彩色图案填充一块区域，我们选择PIXMAP_PATTERN填充类型并利用相关的像素图图案选择过程：

void SRGP_setFillPixmapPattern (**int** *patternIndex*);

因为位图图案和像素图图案都是用索引到的当前颜色表中的颜色值来产生像素的颜色值，所以如果程序员改变了颜色表的内容，填充图元的结果将被改变。SRGP参考手册讨论了如何在位图和像素图图案表中改变和添加内容。虽然SRGP提供了位图图案表的默认内容，但它却没有给出一个默认的像素图图案表，因为有无数的彩色像素图图案都可能是有用的。

```
SRGP_setFillStyle (BITMAP_PATTERN_OPAQUE);
SRGP_setFillBitmapPattern (BRICK_BIT_PATTERN);          /* 砖形图案 */
SRGP_fillPolygon (3, triangleCoords);                   /* a */

SRGP_setFillBitmapPattern (MEDIUM_GRAY_BIT_PATTERN);    /* 50%灰度 */
SRGP_fillEllipseArc (ellipseArcRect, 60.0, 290.0);      /* b */

SRGP_setFillBitmapPattern (DIAGONAL_BIT_PATTERN);
SRGP_fillRectangle (opaqueFilledRect);                  /* c */

SRGP_setFillStyle (BITMAP_PATTERN_TRANSPARENT);
SRGP_fillRectangle (transparentFilledRect);             /* d */

SRGP_setFillStyle (SOLID);
SRGP_setColor (COLOR_WHITE);
SRGP_fillEllipse (circleRect);                          /* e */
```

图2-11 生成图2-9的代码

2. 轮廓线的画笔图案

图案不仅可以用于填充区域定义的图元，也可以通过设置画笔类型属性用于改变线和轮廓图元的外观。利用线宽、线的类型和画笔类型属性，可以创建一个5像素宽的点划线椭圆，而且其边界的实线段部分又是某种图案。图2-12显示了宽实线和宽虚线在透明和不透明模式下用不同图案填充的结果以及它们与事先画出的图元相交的例子；图2-13是产生这些图像的代码。我们并不建议对特别窄的线（1或2个像素宽）使用画笔图案，因为在这样的情况下，图案是无法辨别的。

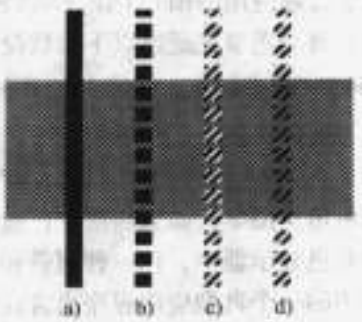


图2-12 画笔类型和线型之间的相互关系。a) 连续实线，b) 虚实线，c) 不透明位图图案的虚线，d) 透明位图图案的虚线

线型和画笔类型之间的相互关系是简单的：线型掩码为0完全保护了落在其中的像素，于是画

笔类型仅仅影响线型掩码是1的像素。

```
/* 我们只说明怎样画线，而不是背景矩形 */
/* 我们按从左到右的顺序画线 */

SRGP_setLineWidth (15);      /* 粗线显示的效果更好 */

SRGP_setLineStyle (CONTINUOUS);
SRGP_setPenStyle (SOLID);
SRGP_line (pta1, pta2);      /* a: 连续实线 */

SRGP_setLineStyle (DASHED);
SRGP_line (ptb1, ptb2);      /* b: 虚实线 */

SRGP_setPenBitmapPattern (DIAGONAL_BIT_PATTERN);
SRGP_setPenStyle (BITMAP_PATTERN_OPAQUE);
SRGP_line (ptc1, ptc2);      /* c: 不透明位图图案的虚线 */

SRGP_setPenStyle (BITMAP_PATTERN_TRANSPARENT);
SRGP_line (ptd1, ptd2);      /* d: 透明位图图案的虚线 */
```

图2-13 生成图2-12中图形的代码

画笔类型与填充类型具有相同的四种选项和填充图案。它们都使用了相同的位图图案和像素图图案表，但用了不同的索引，因此重新设置一个画笔类型的图案索引不会影响到填充类型的图案索引。

```
void SRGP_setPenStyle (SOLID / BITMAP_PATTERN_OPAQUE/... drawStyle);
void SRGP_setPenBitmapPattern (int patternIndex);
void SRGP_setPenPixmapPattern (int patternIndex);
```

3. 应用屏幕背景

在不透明的位图图案中，我们定义“背景色”为0位上绘制的颜色，但是术语背景的定义却不同。一般来说，用户希望在一个覆盖了不透明窗口或者整个屏幕的统一的应用屏幕背景图案上显示图元。应用屏幕背景图案经常是纯色0，即SRGP初始化屏幕后的颜色。然而，背景图案有时是非纯色的或者是其他纯色的；在这些情况下，在画任何其他图元之前，应用程序需要通过绘制一个全屏大小的、用所需背景图案填充的矩形域，来完成应用程序背景设置。

一种常见的“擦去”图元的技术是以应用程序背景图案重画该图元，而不是每次图元被删掉之后都要重画整个图形。然而，当这个被擦去的图元与其他图元交迭时，这种“迅速和不干净的”更新技术会产生一个被损坏的图像。例如，在图2-9中假设屏幕背景图案是纯白色，如果我们用纯白色重画标记为(c)的矩形，则将在被填充的椭圆弧(b)的下方留下一个缺口。“修补损坏的图形”需要回到应用程序数据库中重新定义图元（见习题2.9）。 37

2.1.4 存储和恢复属性

如前所述，SRGP对于各种图元支持多种属性。为了以后能恢复，私有属性可以被存储起来；这一功能对于设计那些无副作用（即不影响全局属性状态）的应用程序是很有用的。对于每一个设置属性的SRGP过程，都有一个相应的查询过程来确定当前的属性值；例如，

```
lineStyle SRGP_inquireLineStyle (void);
```

为方便起见，SRGP允许通过以下过程查询和恢复整个属性集——称为属性组：

void SRGP_inquireAttributes (*attributeGroup *group*);
void SRGP_setAttributes (*attributeGroup *group*);

与以往的版本不同，在当前的SRGP中，应用程序能访问属性组（attributGroup）结构的内部域。然而，直接修改这些域对程序员来说是一件既麻烦又要冒风险的事。

2.1.5 文本

在图形软件包中定义和实现文字绘图往往是比较复杂的，因为文字包含大量的选项和属性，其中有字符的风格或字体（Times Roman，Helvetica，Clarinda,等等）、外观（"Roman"，**bold**，*italic*，underlined，等等）、尺寸（通常以磅[⊖]为单位）和宽度、字符间距、行距、画出字符的角度（水平，垂直，或者一指定的角度），等等。

在简单的软硬件环境中，最基本的字控制功能提供固定宽度和等间距字符，即所有字符占据同样的宽度，且字符之间的间距是常数。而另一方面，均衡间距技术则通过改变字符宽度和字符间距使得文本尽可能清晰和美观。书、杂志、报纸都用均衡间距，大多数的光栅显示器和激光打印机也是如此。SRGP提供折衷的功能：文本水平排列，字符宽度是变化的，但是字符间距是一定的。在均衡间距的简单形式下，应用程序可以注释图表，通过文本菜单和填充表格与用户交互，甚至执行简单的字处理。然而，基于文本的应用程序，如处理高质量文档的桌面排版系统，需要特殊的软件包以提供比SRGP更多的属性和规范控制。PostScript[ADOB87]提
38 供了许多这样的先进功能，并且成为描述包含大量属性和选项的文本以及图元的工业标准。

文本通过调用如下函数产生：

void SRGP_text (point *origin*, **char** *text*);

文本图元的位置通过定义它的原点（或称为锚点）来控制。原点的x坐标标记了第一个字符的左边界，y坐标确定了待显示字符串的基线。（基线是字符下方的一条假想线，如图2-14中所示的文本菜单按钮。一些字符，如"y"和"q"，其尾部（称为字母下部）延伸至基线的下方。）

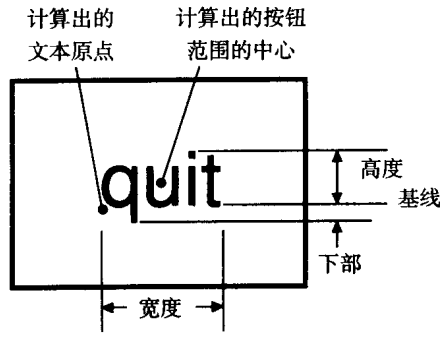


图2-14 将文本置于一个矩形按钮中央的尺寸以及利用这些尺寸计算出的用于居中的点

文本图元的外观仅由两个属性决定，即当前的颜色和字体，而字体是一个与实现相关的不同尺寸和风格的字体表中的索引值：

void SRGP_setFont (**int** *fontIndex*);

字体中的每一个字符都被定义为一个矩形位图，SRGP用字符的位图作为图案来填充一个矩形以绘制一个字符（在透明位图图案模式）。位图中的1定义字符的内部，0定义字符周围的空间和类似于字母"o"的中孔的字符内部间隙。（许多复杂的软件包用像素图的方式定义字符，允许对字符内部进行图案填充。）

格式化文本

由于SRGP提供有限的字体和尺寸功能，而且在不同硬件环境下的具体实现中很少提供等价的功能集，所以应用程序对于字符宽度和高度的控制是受限制的。为了将文本放置在一个平衡的位置（例如，在一个矩形框架内居中放置一个字符串），需要文本范围信息，所以SRGP提供了以下程序，利用字体属性的当前值查询一个给定字符串的范围：

⊖ 点（磅）是出版业中经常使用的一个单位；它大约等于1/72英寸。

```
void SRGP_inquireTextExtent (
    char *text, int *width, int *height, int *descent);
```

虽然SRGP不支持不透明位图模式来写字符，但是这样的模式是容易模仿的。举个例子，图2-15中的过程显示了如何利用当前字体的范围信息和文本特有属性来产生黑色文本，并如图2-14所示居中放置于一个白色封闭的矩形内。这个过程首先创建一个指定尺寸并具有单独边界的背景矩形按钮，然后在它的内部居中显示文本。习题2.10是对于这个问题的一个变体。

39

```
void MakeQuitButton (rectangle buttonRect)
{
    point centerOfButton, textOrigin;
    int width, height, descent;

    SRGP_setFillStyle (SOLID);
    SRGP_setColor (COLOR_WHITE);
    SRGP_fillRectangle (buttonRect);
    SRGP_setColor (COLOR_BLACK);
    SRGP_setLineWidth (2);
    SRGP_Rectangle (buttonRect);

    SRGP_inquireTextExtent ("quit", &width, &height, &descent);

    centerOfButton.x = (buttonRect.bottomLeft.x + buttonRect.topRight.x) / 2;
    centerOfButton.y = (buttonRect.bottomLeft.y + buttonRect.topRight.y) / 2;

    textOrigin.x = centerOfButton.x - (width / 2);
    textOrigin.y = centerOfButton.y - (height / 2);

    SRGP_text (textOrigin,"quit");
}  /* MakeQuitButton */
```

图2-15 生成图2-14中图形的代码

2.2 基本交互处理

既然我们知道了如何绘制基本的形状和文本，下一步就是学会如何编写利用键盘和鼠标等输入设备与用户有效通信的交互程序了。首先，我们看一看编写有效并且易于使用的交互程序的一般原则；然后，讨论逻辑（抽象）输入设备的基本注意事项。最后，我们将考察SRGP实现各种交互处理的机制。

2.2.1 人的因素

交互程序的设计者必须处理许多在非交互程序或批处理程序中不会出现的问题。这些问题被称为程序的人的因素，例如程序的交互风格（经常被称为“视感”）、易学性和易用性，它们与程序功能的完整性和正确性同样重要。第8章和第9章将详细讨论展示良好的人的因素的人机交互技术。其中，主要的原则包括：

40

- 提供简单一致的交互序列。
- 不要以太多的选项和风格加重用户的负担。
- 在交互中的每一步清晰展示可选项。
- 给用户提供正确的反馈。
- 允许用户从错误中很好地恢复。

我们将在实例程序中遵守这些展现良好人的因素的技术原则。例如，用户通常以鼠标点击菜单的一个文本按钮来指出他想执行的下一个功能。同样通用的交互途径还有基本几何图元的调色板（图标菜单）、应用程序符号或者填充图案。菜单和调色板满足了前三个原则，因为它们列出了可用选项列表，并对这些选项提供了简单一致的选择方式。不可用的选项将被暂时删除，或者“灰显”，即以低亮度的灰度图案代替实色图案（见习题2.15）。

为了满足第四个原则，在菜单功能的每一步都提供反馈功能：应用程序会加亮菜单或对象选项（例如，把它放在一个矩形框内或利用相反的视觉显示）以引起用户注意。软件包本身也可以对输入设备的操作提供及时的“回应”。例如，当键盘输入时，在光标的位置字符迅速显现，随着鼠标在桌面上移动，光标回应在屏幕上相应的位置。图形软件包提供多样的光标形状以便应用程序用它来反映当前的程序状态。在许多显示系统中，光标形状可以根据光标在屏幕上的位置动态改变。例如，在许多字处理程序中，在菜单区光标变成箭头形状，而在文本区变成闪烁的垂直线。

第五个原则，即从错误中很好地恢复，通常以“*cancel*”“*undo/redo*”功能实现。这需要应用程序保持操作的记录（见第9章）。

2.2.2 逻辑输入设备

1. SRGP中的设备类型

设计图形软件包的一个主要目标是实现设备无关性，它增强了程序的可移植性。为了实现图形输出的设备无关性，SRGP在抽象的整数坐标系统中定义图元，从而使帧缓存中独立像素的设置与应用程序脱离开来。同样，为提供图形输入抽象，SRGP支持一系列的逻辑输入设备使应用程序与物理输入设备的细节分离。SRGP支持的两种逻辑设备如下：

- 定位器，定义屏幕坐标和一个或多个相应按钮状态的设备。

41

- 键盘，字符串输入的设备。

SRGP将逻辑设备映射到可用的物理设备。例如，定位器可以映射为鼠标、游戏杆、输入板或触摸屏。这种从逻辑到物理的映射类似于传统的从过程语言到操作系统的映射，其中I/O设备，如终端、磁盘和磁带驱动设备被抽象为逻辑数据文件，这样既保证了设备无关性，又简化了应用程序设计。

2. 其他软件包中的设备处理

SRGP的输入模型实际上属于GKS输入模型和PHIGS输入模型的一部分。SRGP仅仅支持一种逻辑定位器和一种键盘设备，而GKS和PHIGS对每一种设备允许多种类型。这些软件包也支持其他的设备类型：笔划设备（返回由物理定位器输入的光标位置序列构成的折线），选择设备（将功能键输入板抽象化并返回一个键标识符），定值器（将一个滑块或控制转盘抽象化并返回一个浮点数），以及拾取设备（将一个带有选择按键的定位设备（如鼠标或数据输入板）抽象化，返回所选择的逻辑选项的标识）。其他程序包（如QuickDraw和X Window System）则采用更加设备相关的方式控制输入设备，这样程序员可以对各个设备的操作进行更细的控制，但是应用程序也因此变得更为复杂。

第8章将介绍逻辑设备的历史，并进一步阐述它们的特性。这里，我们简单概括一下与逻辑设备交互的模式，然后再详尽讨论SRGP的交互过程。

2.2.3 采样与事件驱动处理

接收来自用户的信息有两种基本的方法。一种是采样（也称为轮询）方法，应用程序通过访问逻辑设备的当前值（称为该设备的度量）来执行下一步动作。不管自上一次采样后设备的度量是否发生变化，采样过程一直在执行；事实上，只有通过设备的连续采样才能让应用程序

察觉设备状态的改变。对于交互式应用，这种方法是比较浪费的，因为在采样循环中要消耗大量CPU周期等待度量的变化。

为了避免CPU频繁的轮询循环，另一种办法是利用中断驱动进行交互；利用这种技术，应用程序使一种或多种设备可以输入，然后程序正常运行直到被某一输入事件（由用户动作引起的设备状态的改变）中断；接着，控制被异步传递到一个响应该事件的中断过程。对于每一个输入设备，都定义了一个事件触发器；事件触发器就是导致该事件发生的用户动作。通常触发器是一个按钮动作，如按下鼠标键（"mouse down"）或敲击键盘。

为了使应用程序员从繁杂而困难的异步传输控制中解脱出来，许多图形程序包（包括GKS、PHIGS和SRGP）提供事件驱动的交互过程，以同步方式模拟中断驱动交互。利用这种技术，应用程序在启用设备后继续执行；在程序执行的同时，由软件包监控设备并把每一个事件的信息存储于一个事件队列中（图2-16）。应用程序可在方便的时候随时检查事件队列并按时间顺序处理这些事件。实际上，这等于让应用程序自己决定被"中断"的时刻。 42

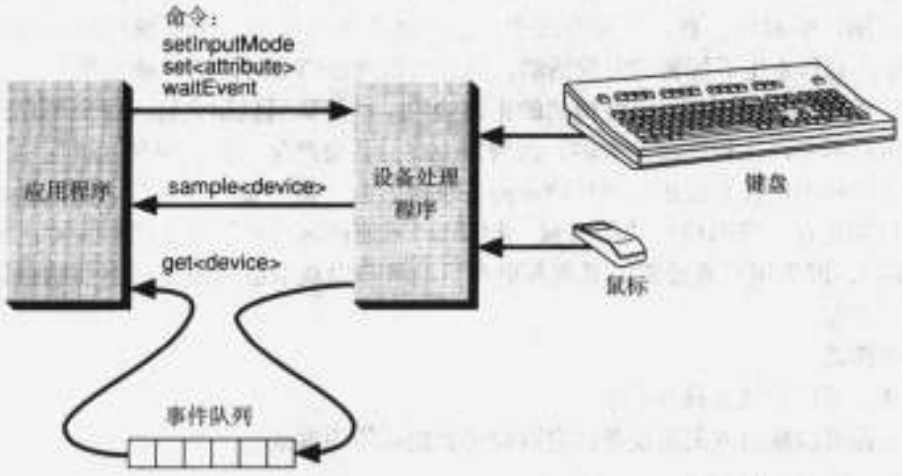


图2-16 采样与使用事件队列的事件处理

当应用程序检查事件队列的时候，它决定是否进入等待状态。如果队列中包括一个或多个事件报告，应用程序取出头事件（代表最早发生的事件）并读取其信息。如果队列是空的且下一个状态不是等待状态，应用程序将被告知无可用事件，并继续原来的执行。如果队列是空的并且即将进入等待状态，应用程序将暂停，直到下一个事件发生或等待事件超过程序指定的最长等待时间。事实上，事件模式用效率更高的事件队列等待代替了输入设备的轮询。

总之，在采样模式中，不管是否有用户动作，都要轮询设备并搜集事件度量。在事件模式中，应用程序可以获得用户动作的事件报告或等待直到有用户动作（或超时）发生。这种"仅当用户有动作时才反应"的事件模式的行为特征是事件驱动输入与采样输入的最重要的不同点。事件驱动编程看起来可能比采样更复杂，但是你可能已经对一个类似的技术比较熟悉了，如C语言程序中使用的scanf函数：C启用了键盘，应用程序在scanf函数中等待用户完成一行文本的输入。你也可以利用C语言中的getc函数访问独立的按键事件。

SRGP或类似软件包中的简单的事件驱动程序遵循1.6.4节中介绍并由图2-17的伪代码描述的"乒乓"交互。这种交互能被很好地建模为一个有限状态自动机。允许程序和用户动作同时发生的更多复杂类型的交互，将在第8章至第10章中讨论。

```
初始化，包括生成初始图像；
激活交互设备使之处于事件模式；
while (用户没有请求退出) {   /* 主事件循环 */
    等待任何设备上的用户触发事件；
    switch (引发事件的设备) {
        case DEVICE_1: collect DEVICE_1 事件度量数据、处理、响应；
        case DEVICE_2: collect DEVICE_2 事件度量数据、处理、响应；
        ...
    }
}
```

图2-17 事件驱动的交互模式

事件驱动的应用程序大部分时间处在等待状态，因为交互过程主要是由用户决定下一步行动的“思考时间”控制的；即使在快节奏游戏程序中，一秒钟里用户所能产生的事件数量也只是应用程序所能处理的一小部分。由于SRGP主要使用实（硬件）中断来实现事件模式，等待状态并不占用CPU时间。在多任务系统中，这一优点是明显的：事件模式的应用程序只需要CPU完成用户动作发生后短暂的突发活动，因此可以腾出CPU时间给其他任务。

还要注意一点，就是关于事件模式的正确使用。虽然队列机制允许程序和用户交互异步进行，但是用户不允许比程序超前太多，因为每一事件都会产生一些回应和来自应用程序的一些反馈。有经验的用户在系统正在处理较早的请求时，会“提前键入”文件名甚至操作系统命令等参数，特别是在字符回应很快的时候。相反，“提前鼠标操作”对于图形命令毫无用处（而且非常危险），因为用户通常都需要观看更新后的屏幕以获取应用模型的当前状态，才能进行下一步图形交互。

2.2.4 采样模式

1. 激活、关闭和设置设备模式

以下过程可以激活或关闭设备；它以设备和模式作为参数：

```
void SRGP_setInputMode (
    LOCATOR / KEYBOARD  inputDevice, INACTIVE / SAMPLE / EVENT  inputMode);
```

这样，为了把定位器设为采样模式，我们调用

```
SRGP_setInputMode (LOCATOR, SAMPLE);
```

最初，两个设备都处于非激活状态。设置一个设备为某种模式决不会影响其他输入设备——即使两者被同时激活，也不一定要在同一模式下工作。

2. 定位器的度量

定位器是鼠标或数据输入板的逻辑抽象，返回光标的屏幕坐标（x, y）、最近发生变化的按钮的编号以及以*chord*数组形式返回按钮的状态（因为能同时按下多个按钮）。其中第二个域可以让应用知道是哪个按钮触发了那一事件。

43
~
44

```
typedef struct {
    point position;
    int buttonOfMostRecentTransition;
    enum {UP, DOWN} buttonChord[MAX_BUTTON_COUNT];      /* 通常是1~3 */
} locatorMeasure;
```

在使用SRGP_setInputMode过程以采样模式激活定位器后，我们就可通过调用以下过程获取定位器的当前度量：

void SRGP_sampleLocator (locatorMeasure *measure);

让我们看一下图2-18中的采样应用程序原型：一个简单的仅使用定位器上按钮1的“着色”循环。这个着色过程会沿着用户按下该按钮同时挪动定位器的轨迹着色；定位器在用户移动时被循环采样。首先，我们必须对定位器按钮进行采样，直到它被按下，从而确定用户开始着色；然后，我们对定位器进行采样，并在每个采样点上着色（在这个例子中是一个填充的矩形），直到用户释放这个按钮。

```
设置颜色/图案属性，并设画刷尺寸为halfBrushHeight和halfBrushWidth;
SRGP_setInputMode (LOCATOR, SAMPLE);

/* 首先，一直采样直到按钮按下 */
do {
    SRGP_sampleLocator (&locMeasure);
} while (locMeasure.buttonChord[0] == UP);

/* 执行画图循环 */
/* 不断地放置画刷然后采样，直到按钮抬起 */
do {
    rect = SRGP_defRectangle (locMeasure.position.x - halfBrushWidth,
                              locMeasure.position.y - halfBrushHeight,
                              locMeasure.position.x + halfBrushWidth,
                              locMeasure.position.y + halfBrushHeight);
    SRGP_fillRectangle (rect);
    SRGP_sampleLocator (&locMeasure);
} while (locMeasure.buttonChord[0] == DOWN);
```

图2-18 着色的采样循环

这个过程的结果是粗糙的：着色的矩形不是过于紧凑，就是过于分散，它们的密度完全依赖于定位器在两个连续采样之间移动的距离。采样速率主要由CPU运行操作系统、软件包和应用程序的速度决定。

虽然两种逻辑设备都可以使用采样模式，但是键盘设备几乎总是在事件模式下运作，所以这里没有提及关于它的采样技术。

2.2.5 事件模式

1. 使用事件模式启动采样循环

虽然着色例子中的两个采样循环（一个检测按钮按下，另一个完成着色直到按钮松开）都能运行，但是它们无谓地增加了CPU的工作量。虽然这种方法在个人计算机上可能不会产生严重的问题，但是在一个多任务系统中却是不适宜的，更不用说要实现分时了。尽管对定位器重复采样的着色循环是必不可少的（因为我们需要知道按钮按下时定位器在任何时间的位置），我们并不需要使用采样循环等待启动着色过程的按钮按下事件。在等待事件发生而不需要度量信息的时候，我们可以采用事件模式。 45

2. SRGP_waitEvent

当SRGP_setInputMode在事件模式下激活一个设备后，程序可以通过调用以下过程进入等待状态来检测事件队列：

inputDevice SRGP_waitEvent (**int** *maxWaitTime*);

如果队列非空，这个过程立即返回；否则，第一个参数将指定这个过程在队列为空时的最大等待时间（以1/60秒计）。负的*maxWaitTime*值（由符号常量INDEFINITE定义）将导致过程无限期等待，0值导致过程立即返回，不论队列状态如何。

发生头事件的设备标识在*device*参数中返回。如果指定的时间内没有事件发生，即如果设备超时，返回特殊值NO_DEVICE。接下来可以通过检测设备以决定如何提取头事件的度量（将在本节后面描述）。

3. 键盘设备

键盘设备的触发事件取决于键盘设备所在的处理模式。在EDIT模式中，应用程序接收用户输入的字符串（例如，文件名和命令），用户在键入并编辑字符串后按下回车键触发事件。在RAW模式的交互过程中，键盘受到紧密监控，每按下一个键就触发一个事件。应用程序可以使用以下过程设置处理模式：

void SRGP_setKeyboardProcessingMode (EDIT / RAW keyboardMode);

在EDIT模式中，用户可以键入完整的字符串，需要时可用退格键改正，最后使用回车键（或输入键）作为触发器。这种模式适用于键入完整的字符串，如文件名或图形标签。除了退格和回车键之外所有的控制键都被忽略，因为字符串出现在触发的时刻，所以度量就是字符串本身。然而，在RAW模式中，每一个键入的字符，包括控制字符，都是一个触发器并作为度量单独返回。这种模式在单个键盘字符作为命令（例如，移动光标、简单的编辑操作或者视频游戏）时使用。RAW模式不提供回应，而EDIT模式回应字符串并显示在屏幕上，同时在下一文本输入位置显示一个文本光标（如下划线或块字符）。每按一次退格键文本光标回撤一格并删掉一个字符。

46 在SRGP_waitEvent返回设备代号KEYBOARD后，应用程序通过调用以下过程获得与该事件相关的度量：

void SRGP_getKeyboard (**char** **measure*, **int** *measureSize*);

当键盘设备在RAW模式下被激活后，它的度量保持一个字符的长度。在这种情况下，度量字符串的第一个字符作为RAW模式的度量返回。

从图2-19所示的程序段可以看到EDIT模式的使用方法。它从用户端接收一系列文件名，同时删除相应的文件。如果用户输入一个空字符串（不按任何其他字符直接按回车键），交互结束。在交互过程中，程序无限期等待直到用户输入下一个字符串。

```
SRGP_setInputMode (KEYBOARD, EVENT);      /* 假设只有键盘激活 */
SRGP_setKeyboardProcessingMode (EDIT);
pt = SRGP_defPoint (100, 100);
SRGP_text (pt, "Specify one or more files to be deleted; to exit, press Return.");

/* 主事件循环 */
do {
    device = SRGP_waitEvent (INDEFINITE);
    SRGP_getKeyboard (measure, measureSize);
    if (*measure != NULL)
        DeleteFile (measure);                 /* DeleteFile做确认等 */
} while (*measure != NULL);
```

图2-19 EDIT模式下的键盘交互

虽然这段代码明确地指定了文本提示符的显示位置，但是它并没有指定用户键入字符串（以及用退格键修正字符串）的位置。键盘回应的位置是由程序员指定的，我们将在2.2.7节中讨论这个问题。

4. 定位器设备

定位器设备的触发事件是按下或松开鼠标键。在SRGP_waitEvent返回设备代号LOCATOR后，应用程序通过调用以下过程获得与该事件相关的度量：

```
void SRGP_getLocator (locatorMeasure *measure);
```

通常度量的*position*字段用来确定用户指定的点在屏幕上的位置。例如，如果定位器光标位于一个显示菜单按钮的矩形域中，事件将被视为某一动作的请求；如果光标位于主绘画区域中，事件则可以表示选中某一个已经存在的图形对象或是放置一个新的图形对象的位置，这取决于光标点在图形对象的内部还是外部。

图2-20所示的伪代码（与前面所示的键盘伪代码类似）实现了定位器的另一个用途，即让用户指定放置标记的位置。当光标指向一个屏幕按钮——一个标有“*quit*”的矩形域——时，按下定位器按钮就可以退出标记定位循环。

```
const int QUIT_BUTTON = 0, QUIT_MASK = 0x1;

在屏幕上创建退出按钮;
SRGP_setLocatorButtonMask (QUIT_MASK);
SRGP_setInputMode (LOCATOR, EVENT);      /* 假设只有定位器激活 */

/* 主事件循环 */
terminate = FALSE;
while (!terminate) {
    device = SRGP_waitEvent (INDEFINITE);
    SRGP_getLocator (&measure);
    if (measure.buttonChord[QUIT_BUTTON] == DOWN) {
        if (PickedQuitButton (measure.position))
            terminate = TRUE;
        else
            SRGP_marker (measure.position);
    }
}
```

图2-20 定位器交互

在这个例子中，只有定位器按钮1的按下事件有意义；按钮的释放被忽略。注意在下一个按按钮事件发生之前定位器按钮必须松开——这个事件是通过转变触发的，而不是通过按钮的状态来触发。此外，为保证交互过程不受其他按钮事件的干扰，应用程序调用以下函数通知SRGP哪个按钮将触发定位器事件：

```
void SRGP_setLocatorButtonMask (int activeButtons);
```

默认的定位器按钮屏蔽位为1，但无论屏蔽位是几，每个按钮都有自己的度量。在支持少于三个按钮的实现中，指向任何不存在按钮的引用都会被SRGP忽略，而这些按钮的度量都包含UP。

函数PickedQuitButton将度量位置与退出按钮矩形框的边界进行比较，并返回一个布尔值来表示用户是否选中退出按钮。这个过程是一个简单的关联拾取（pick corralation）的例子，

我们将在下一节讨论。

5. 等待多重事件

图2-19和图2-20所示的代码段并没有说明事件模式最大的优越性：同时等待多个设备的能力。SRGP将来自启用设备的事件按时间顺序放入队列，应用程序调用SRGP_waitEvent过程从队列中取出第一个事件。与硬件中断不同，后者以优先权顺序处理中断，而前者严格按照时间顺序处理事件。应用程序通过检查返回的设备代号来确定哪个设备引发了该事件。

图2-21中所示的过程允许用户将任意多个小圆形标记放置在矩形绘图区内的任意位置。用户把光标挪到想要的位置，按下键1就可放置一个标记；用户可以按下按钮3或输入“q”或“Q”字符终止交互。

```
const int PLACE_BUTTON = 0, PLACE_MASK = 0x1,
          QUIT_BUTTON = 2, QUIT_MASK = 0x4;

生成初始的屏幕布局;
SRGP_setInputMode (KEYBOARD, EVENT);
SRGP_setKeyboardProcessingMode (RAW);
SRGP_setInputMode (LOCATOR, EVENT);
SRGP_setLocatorButtonMask (PLACE_MASK | QUIT_MASK);     /* 忽略按钮2 */

/* 主事件循环 */
terminate = FALSE;
while (!terminate) {
    device = SRGP_waitEvent (INDEFINITE);
    switch (device) {
        case KEYBOARD:
            SRGP_getKeyboard (keyMeasure, keyMeasureSize);
            terminate = (keyMeasure[0] == 'q') || (keyMeasure[0] == 'Q');
            break;
        case LOCATOR:
            SRGP_getLocator (&locMeasure);
            switch (locMeasure.buttonOfMostRecentTransition) {
                case PLACE_BUTTON:
                    if ((locMeasure.buttonChord[PLACE_BUTTON] == DOWN)
                        && InDrawingArea (locMeasure.position))
                        SRGP_marker (locMeasure.position);
                    break;
                case QUIT_BUTTON:
                    terminate = TRUE;
                    break;
            } /* 按钮情况 */
    } /* 设备情况 */
} /* while */
```

图2-21 同时使用多个设备

2.2.6 交互处理中的关联拾取

图形应用程序通常将屏幕划分为一些完成特定功能的区域。当用户点击定位器按钮时，应用程序必须准确地判断用户选择了哪个屏幕按钮、图标或其他对象，以做出适当的响应。这个判断过程称为关联拾取，它是交互式图形应用的基本组成部分。

使用SRGP的应用程序通过确定光标位于哪个区域，以及位于该区域内的哪个对象上来实

现关联拾取。空区域内的点将被忽略（例如，菜单上菜单按钮之间的点），或用于指定放置新对象的位置（如果点位于主绘图区域内）。因为屏幕上许多区域都是正矩形，所以几乎所有与关联拾取有关的工作都可以由一个简单的、常用的布尔函数完成，这个函数判定给定的点是否在给定的矩形内部。随SRGP发布的GEOM软件包提供了这个函数（GEOM_ptInRect）以及其他坐标运算工具。（有关关联拾取的更详细的资料，参见7.12节。）

47 ~ 49

让我们看一个关联拾取的经典例子。现有一绘图应用程序，程序中屏幕顶端设有一菜单条。菜单条中包含各下拉菜单的名字，称为菜单标题。当用户选取一个标题时（将光标放置在显示标题的文字串上并按下定位器键）[⊖]，相应的菜单体显示在屏幕上标题的下方，同时标题高光显示。当用户选择菜单体上的一项后（释放定位器按钮），菜单体消失，标题恢复正常显示。屏幕其他部分是供用户放置和拾取对象的主绘图区。应用程序在创建每个对象时，分配给该对象一个惟一的标识符（ID），由关联拾取过程返回并进行进一步的处理。

通过按下按钮事件从定位器得到一个点后，程序按照图2-22所示的过程进行高层交互处理；本质上这是一个调度过程，它根据关联拾取在菜单条中还是在主绘图区中分别调用菜单拾取过程和对象拾取过程。首先，如果光标在菜单条内，会有一个辅助相关过程判断用户是否选择了一个菜单标题。如果是，将调用一个过程（细节见2.3.1节）来执行菜单交互；同时返回菜单体中被选中的项的下标（如果有的话）。菜单ID和项目下标在一起可以惟一确定响应操作。如果光标不在菜单条内而是在主绘图区内，另一个辅助相关过程将被调用以确定被拾取的对象（如果有的话）。如果一个对象被拾取，程序会调用一个处理过程做出正确的响应。

```
void HighLevelInteractionHandler (locatorMeasure measureOfLocator)
{
    if (GEOM_pointInRect (measureOfLocator.position, menuBarExtent)) {
        /* 找出用户选择哪个菜单头（如果有） */
        /* 然后，到菜单体 */
        menuID = CorrelateMenuBar (measureOfLocator.position);
        if (menuID > 0) {
            chosenItemIndex = PerformPulldownMenuInteraction (menuID);
            if (chosenItemIndex > 0)
                PerformActionChosenFromMenu (menuID, chosenItemIndex);
        }
    } else {      /* 用户在绘图区域内拾取，找出内容和响应 */
        objectID = CorrelateDrawingArea (measureOfLocator.position);
        if (objectID > 0)
            ProcessObject (objectID);
    }
}  /* HighLevelInteractionHandler */
```

图2-22 菜单处理的高层交互概要

CorrelateMenuBar过程通过对菜单条内的每个菜单标题调用一次GEOM_pointInRect实现了较好的关联拾取；它访问的是一个存储各标题矩形屏幕区的数据库。CorrelateDrawingArea 过程的关联拾取过程更加复杂，因为通常绘图区内的对象会重叠，而且不一定都是矩形的。

2.2.7 设置设备度量和属性

各输入设备都有其自己的属性集合，应用程序可以通过设置这些属性来定制反馈给用户的

⊖ 这种Macintosh菜单交互方式相对应的序列，仅是多种用户界面设计方式的一种。

信息。（前面提到的按钮屏蔽位也是一个属性；它与这里介绍的属性的不同之处在于它不影响反馈。）与输出图元属性类似，输入设备属性由特定的过程模态地设置。设备属性可以随时设置，无论设备是否处于激活状态。

另外，各输入设备的度量（通常都由用户的操作决定）也可由应用程序来设置。与输入设备的属性不同，当输入设备被关闭时，该设备的度量复位为默认值；这样，设备在被重新激活时就有可预知的值，对编程者和用户都很方便。这种自动复位功能可以由显式设置设备的度量替代。

1. 定位器回应属性

定位器可以使用多种回应类型。程序员可以调用函数

```
void SRGP_setLocatorEchoType (
    NO_ECHO / CURSOR / RUBBER_LINE / RUBBER_RECT  echoType);
```

同时控制回应类型和光标形状。参数的默认值为CURSOR，SRGP软件包提供一个光标表供应用程序选择所需的光标形状（参阅参考手册）。动态指定光标的形状通常用于根据光标所在区域改变光标的形状以提供反馈。RUBBER_LINE和RUBBER_RECT回应通常用来指定一条直线或一个框。当用户移动定位器时，SRGP通过设置这两个属性自动绘制连续更新的直线或矩形。直线或矩形是由两个点，即锚点（定位器的另一属性）和当前的定位器位置来定义的。图2-23说明了如何使用这两种模式定义直线和矩形。

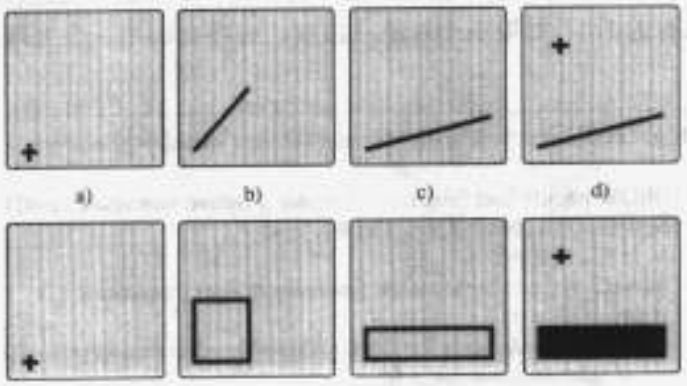


图2-23 Rubber-echo图解

在图2-23a中，光标是个十字游标，用户正要按下定位器按钮。应用程序启动橡皮回应（rubber echo）响应按下按钮的动作，锚点定在当前定位器的位置。在图2-23b和图2-23c中，程序以橡皮图元回应定位器的移动。当用户释放按钮时，图2-23c中的定位器位置返回给应用程序，而应用程序则画出相应的线图元或矩形图元，并恢复正常的光标形状（见图2-23d）。

橡皮回应的锚点是通过调用以下函数设置的：

```
void SRGP_setLocatorEchoRubberAnchor (point position);
```

应用程序通常将从最近一次按定位器按钮事件获得度量的*position*域指定为锚点的位置，因为按键动作通常触发橡皮回应序列。

2. 定位器度量控制

一旦定位器被关闭，定位器度量的*position*值自动复位到屏幕的中心。除非程序员重新设置，否则度量（如果回应被激活，还包括反馈位置）在设备被重新激活时将被初始化到相同的位置。任何时候，无论设备正在工作还是不在工作，编程者可以调用函数

void SRGP_setLocatorMeasure (point *position*);

重置定位器的度量（*position*部分，而不是与按键有关的域）。

当定位器处于关闭状态时重置度量，结果不会立刻在屏幕上反映出来；但当定位器处于激活状态时重置度量，会及时改变相应的回应（如果有的话）。这样，如果程序中要让光标在定位器激活时位于除中心点以外的其他位置，必须在调用SRGP_setInputMode前以所需初始位置调用SRGP_setLocatorMeasure函数。这一技术通常用于实现光标位置的连续性：定位器关闭前的最后一个度量被存储起来，当设备被重新激活时，光标可以回到该位置。

3. 键盘属性和度量控制

定位器回应的是物理设备的运动位置，而键盘设备的回应在屏幕上没有确定的位置。因此位置作为键盘设备的属性之一（默认值与具体实现有关），可通过调用以下函数设置：

void SRGP_setKeyboardEchoOrigin (point *origin*);

当键盘设备关闭时，键盘的默认度量自动复位为空串。如果在激活键盘前将度量设置为一个非空的初始值，可以很方便地实现一个默认输入串的显示（在回应一开始就由SRGP显示），用户可以接受它或在修改后按回车键，从而减少击键次数。键盘的度量是通过调用以下函数设置的：

void SRGP_setKeyboardMeasure (**char** **measure*);

2.3 光栅图形特性

到现在为止，我们已经介绍了SRGP的大多数特性。这一节将讨论其余的功能，它们充分利用了光栅显示硬件的优势，尤其是保存和恢复屏幕中被其他图像（例如，窗口或临时菜单）覆盖区域的能力。这样的图像操作是在窗口管理应用程序和菜单管理应用程序的控制下完成的。我们还引入屏外位图（称为画布）来存储窗口和菜单，并将讨论如何进行矩形框裁剪。

50 ~ 52

2.3.1 画布

让复杂的图标或菜单快速地显示或消失的最好方法是在内存中创建它们，然后将它们按需要复制到屏幕上。光栅图形软件包首先在所需尺寸的不可见的屏外位图或像素图（SRGP中称为的画布）中生成图元，然后将画布复制到显存或从显存中读出。这实际上是一种缓冲技术。使用了我们即将讨论的快速SRGP_copyPixel操作后，整块地来回移动像素一般要比重新生成信息更快。

SRGP画布是一个将图像作为2D像素数组存储的数据结构。它也存储一些有关图像大小和属性的控制信息。每个画布在各自的笛卡儿坐标系下显示图像，这与图2-1中所示的屏幕显示相同；实际上，屏幕本身就是个画布，特殊之处仅在于它是惟一被显示的画布。要显示一个存储在屏外的画布上的图像，应用程序必须将它复制到屏幕画布上。在新图像（例如菜单）即将显示的屏幕部分内的图像，其像素将预先复制到其他屏外画布上存储起来。待菜单选中之后，这些像素再从画布复制回以恢复原图像。

在任何时候，只有一个当前活动画布，新的图元将绘制到这个画布上，而新的属性设置也将对它生效。该画布可能是屏幕画布（我们使用的默认画布）或是一个屏外画布。传给图元过程的坐标是以当前活动画布的局部坐标空间形式表示的。每个画布都有自己完整的SRGP属性集，这些属性影响所有存储在该画布上的图形，而且当创建画布时，这些属性被设置为标准的默认值。对属性设置过程的调用仅修改当前活动画布的属性。为方便起见，可以把画布看成一个虚拟屏幕，它具有程序指定的尺寸、与自己关联的像素图、坐标系和属性组。画布的这些特性有时也被称为是画布的状态或内容。

当SRGP被初始化时，屏幕画布自动创建并被激活。我们讨论过的所有程序只能在该画布

内生成图元。它是惟一在屏幕上可见的画布，而且其ID是SCREEN_CANVAS，为SRGP常量。通过调用以下过程可创建一个新的屏外画布，该过程返回分配给新画布的ID：

```
canvasID SRGP_createCanvas (int width, int height);
```

与屏幕类似，新画布的局部坐标系的原点（0，0）在左下角，右上角位于（*width*−1，*height*−1）。一个1×1的画布定义的宽和高均为1，它的左下角和右上角均在（0，0）！这与我
53 们对网格交点处的像素的处理相一致：在1×1的画布上的单个像素位于（0，0）。

新创建的画布被自动激活，并且其像素初始化为颜色0（在任何图元显示前，屏幕画布也完成同样操作）。一旦画布被创建，它的尺寸不能再改变。同样，由于SRGP使用的像素的位数与硬件要求一致，所以程序员也不能控制画布中各像素所占的位数。画布的属性作为其“局部”状态信息的一部分被保存；这样，在创建一个新的活动画布之前，程序不需要显式地保存当前活动画布的属性。

应用程序通过调用以下函数选择一个已经创建的画布作为当前活动画布：

```
void SRGP_useCanvas (canvasID id);
```

画布被激活决不意味着该画布变为可见；屏外画布上的图像必须拷贝到屏幕画布上（使用刚才提到的SGRP_copyPixel过程）才能显示。

可以通过调用以下过程删除画布：

```
void SRGP_deleteCanvas (canvasID id);
```

但该过程不能用来删除屏幕画布或当前活动画布。以下过程允许查询画布的尺寸；其中一个返回定义画布坐标系（左下角点通常为（0，0））的矩形，另一个将宽和高分别作为独立量返回。

```
rectangle SRGP_inquireCanvasExtent (canvasID id);
void SRGP_inquireCanvasSize (canvasID id, int *width, int *height);
```

让我们来看如何使用画布实现在图2-22和2.2.6节中提到的高层交互处理程序调用的PerformPulldownMenuInteraction过程。该过程由图2-24给出的伪代码实现，图2-25展示了它的动作序列。每个菜单有各自惟一的ID（由CorrelateMenuBar函数返回），它可用来定位包含以下有关菜单体外观信息的数据库记录：

- 存储菜单体的画布的ID。
- 用屏幕画布坐标来表示的矩形区（伪代码中称为*menuBodyScreenExtent*），即当用户点击菜单标题下拉菜单时，菜单体的显示区域。

```
int PerformPulldownMenuInteraction (int menuID);
/* 画布矩形区域的保存/复制在2.3.3节描述 */
{
    高光显示菜单条中的菜单头;
    menuBodyScreenExtent = screen-area 菜单体将要出现的屏幕区域矩形;
    在临时画布上保存menuBodyScreenExtent的当前像素;
        /* 参见图2-25a */
    将菜单体图像从体画布拷贝到menuBodyScreenExtent;
        /* 参见图2-25b和图2-28中的C代码 */
    等待按钮弹起，提醒用户作出选择，然后得到定位器的度量；将暂时的画布上所保存
    的图像拷贝到menuBodyScreenExtent;
        /* 参见图 2-25c */
```

图2-24 PerformPulldownMenuInteraction的伪代码

```
    if (GEOM_pointInRect (measureOfLocator.position, menuBodyScreenExtent))
        使用所测位置的Y坐标计算并返回所选项的索引;
    else
        return 0;
} /* PerformPulldownMenuInteraction */
```

图2-24（续）

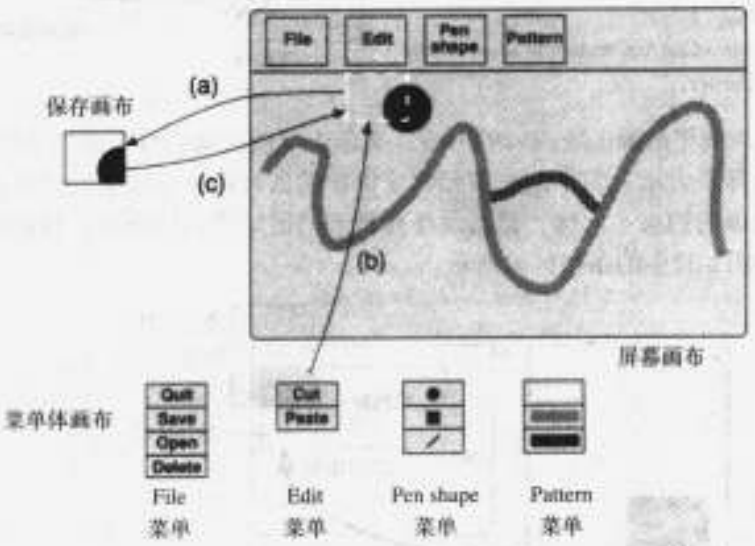


图2-25 保存和恢复被菜单体覆盖的区域

2.3.2 矩形框的裁剪

通常，为了保护画布的其他区域，需要把图形图元的作用限制在活动画布的一个子区域内。SRGP通过设置裁剪矩形（clip rectangle）属性实现该功能。所有的图元都被该矩形的边界裁剪；也就是说，落在裁剪矩形外的图元（或图元的一部分）不画出来。与其他属性一样，裁剪矩形属性可在任何时候改变，它最近一次的设置存储在画布的属性组中。默认的裁剪矩形（我们至今使用的）是整个画布；它可以变得比画布小，但不能扩大超出画布的边界。裁剪矩形属性的相关设置与查询调用如下：

```
void SRGP_setClipRectangle (rectangle clipRect);
rectangle SRGP_inquireClipRectangle (void);
```

如2.2.4节中介绍的绘图应用程序就可以利用裁剪矩形把着色的位置限制在屏幕的绘图区内，从而保证周围的菜单区域不受影响。尽管SRGP仅提供了一个正矩形裁剪边界，一些更高级的软件（如POSTSCRIPT）则提供了多重、任意形状的裁剪区。

54
|
55

2.3.3 SRGP_copyPixel操作

强大的SRGP_copyPixel命令是典型的光栅命令，用硬件直接实现时经常被称为bitBlt（位块传输）或pixBlt（像素块传输）；在20世纪70年代早期，Xerox Palo Alto研究中心在开拓ALTO位图工作站上首先实现了其微代码[INGA81]。该命令用来从画布的矩形区域（源区域）将一个像素数组拷贝到当前活动画布的目的区域（见图2-26）。SRGP仅提供有限的功能，即目的矩形必须与源矩形具有相同的尺寸。在更强大的版本中，源区域可以自动变换尺寸并被拷贝到一个与它大小不同的目的区域（见19章）。同时还有一些额外的功能，例如屏蔽

（mask）功能可以选择性地屏蔽源或目的像素拷贝（见19章），而半色调图案（halftone pattern）可以用来“筛掩”（即浓淡遮蔽）目的区域。

SRGP_copyPixel可以在任何两个画布间拷贝，其调用方法如下：

```
void SRGP_copyPixel (
    canvasID sourceCanvas, rectangle sourceRect
    point destCorner);
```

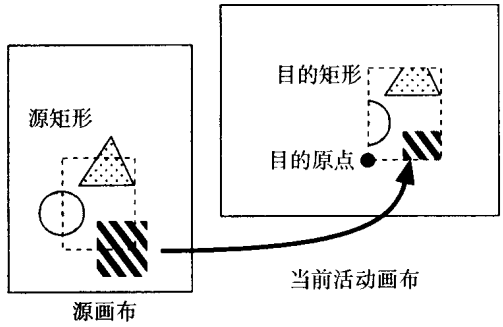


图2-26 SRGP_copyPixel

*sourceRect*指定任意画布中的源区域，*destCorner*定义位于当前活动画布中的目的矩形的左下角，它们都以各自的坐标系表示。为了防止图元在受保护的区域内生成像素，拷贝操作同样受到活动画布上裁剪矩形的约束。这样，像素最终拷贝到的区域是目的画布，目的区域和裁剪矩形三者的交集，如图2-27中的条纹区域所示。

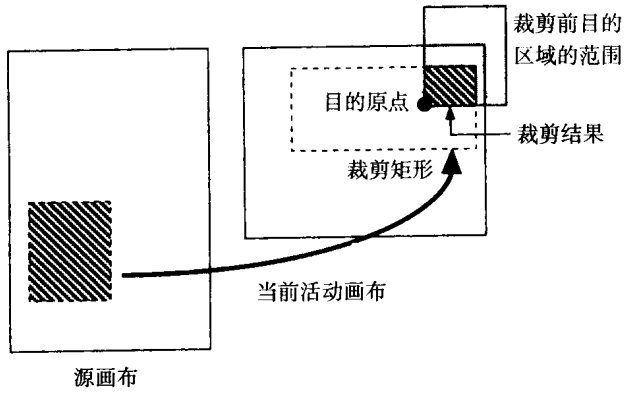


图2-27 copyPixel中的裁剪

为了介绍如何使用copyPixel处理下拉菜单，我们执行PerformPulldownMenuInteraction函数（图2-24）中的第4句伪代码—— “copy menu body image（拷贝菜单体图像）”。在该伪代码的第3句，我们已经把菜单体将要显示的屏幕区域存储在一个屏外画布中；现在，我们希望将菜单体拷贝到屏幕上。

C代码如图2-28所示。我们必须区别两个大小相同但用不同的坐标系表示的矩形。第一个矩形，我们在代码中称之为*menuBodyExtent*，就是菜单体的画布在其坐标系中的区域。该区域用来作为SRGP_copyPixel操作的源矩形，将菜单放到屏幕上。*menuBodyScreenExtent*是一个大小相同的矩形，它在屏幕坐标中指定菜单体将要显示的位置；该区域的左下角是与菜单标题的左边水平对齐的，其右上角紧靠菜单条的底部。（图2-25用点划线框表示Edit菜单的屏幕区域，它的菜单体区域用实线表示。）*menuBodyScreenExtent*左下角点用来指定拷贝菜单体的SRGP_copyPixel的目的位置。另一方面，它也用于存储要被菜单体覆盖的屏幕区域的源矩形和最后恢复操作的目的矩形。

请注意保存和恢复应用程序的状态以消除副作用。在拷贝前，我们应将屏幕裁剪矩形设置为SCREEN_EXTENT，或者我们可以把它设为*menuBodyScreenExtent*的具体值。

```
/* 这段代码将菜单体的图像拷贝到屏幕上， */
/* 其在屏幕上的位置由存放在菜单体的记录决定 */

/* 保存当前活动画布的ID */
saveCanvasID = SRGP_inquireActiveCanvas();

/* 保存当前屏幕画布的裁剪矩形的属性值 */
SRGP_useCanvas (SCREEN_CANVAS);
saveClipRectangle = SRGP_inquireClipRectangle ();

/* 临时设置屏幕剪切矩形，允许向整个屏幕上写 */
SRGP_setClipRectangle (SCREEN_EXTENT);

/* 将菜单体从它的画布中拷贝到菜单栏上的标题下面的正确区域 */
SRGP_copyPixel (menuCanvasID, menuBodyExtent, menuBodyScreenExtent.bottomLeft);

/* 恢复屏幕属性和活动画布 */
SRGP_setClipRectangle (saveClipRectangle);
SRGP_useCanvas (saveCanvasID);
```

图2-28 将菜单体拷贝到屏幕上的代码

2.3.4 写模式或RasterOp

SRGP_copyPixel的功能不仅仅是将一个像素数组从源区域移到目的区域。它也可以执行源区域和目的区域内的各对应像素组对之间的逻辑（按位）操作，然后将结果放到目的区域中。该操作可以用以下符号表示：

$$D \leftarrow S \ \mathbf{op} \ D$$

式中的**op**常常称为*RasterOp*（光栅操作）或写模式，通常由16种布尔运算符组成。SRGP支持其中最常用的几种，包括置换（**replace**）、或（**or**）、异或（**xor**）和与（**and**）；在图2-29中以1位/像素的图像为例说明了四种操作的差异。

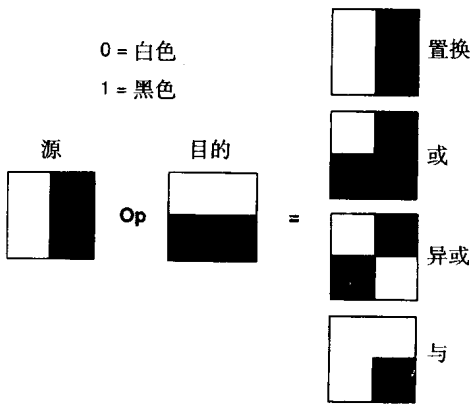


图2-29 结合源像素和目的像素的写模式

写模式影响的不仅是SRGP_copyPixel，还有所有写到画布上的新图元。每个像素（SRGP_copyPixel操作的源矩形像素或图元像素）存储于各自的内存单元，写操作时既可以采用破坏性的**replace**模式，也可以将它的值与当前存储的像素值进行逻辑结合。（这个源值与目的值的逐位结合类似于在读-修改-写内存循环中CPU硬件对内存单元的内容执行算术或逻辑操作。）尽管**replace**是最常用的模式，**xor**模式在生成动态对象（例如，我们即将要讨论的光标和橡皮回应）时也很有用。

56
~
58

可以调用以下函数设置写模式属性：

```
void SRGP_setWriteMode (
    writeMode WRITE_REPLACE / WRITE_XOR / WRITE_OR / WRITE_AND );
```

由于所有的图元是根据当前的写模式生成的，SRGP程序员必须确定已显式地设置该模式，而不能依赖于默认设置WRITE_REPLACE。

为了了解RasterOp的工作原理，我们现在来看软件包内部是如何完成像素的存储和操纵的。这是我们从抽象角度讨论光栅图形至今，惟一一次涉及硬件和具体实现的问题。

RasterOp是对作为颜色表索引的像素值进行操作，而不是对颜色表中存储的硬件颜色定义操作。这样，在一个二值的1位/像素的系统中，RasterOp的操作对象是两个1位的索引。在一个8位/像素的颜色系统中，RasterOp则对两个8位的索引做逐位逻辑运算操作。

尽管图2-29中显示的四个基本运算对于1位/像素单色图像的操作很直观，但对于n位/像素图像（$n>1$），除**replace**模式以外的其他模式的结果都很不自然。这是因为对源索引和目的索引的按位逻辑运算将产生第三个索引，而该索引的颜色值可能与源颜色和目的的颜色完全不同。

replace模式涉及改写屏幕（或画布）上已有的内容。这种破坏性写操作是绘制图元的正常模式，通常用来移动和弹出窗口。它也可用于以应用程序背景图案重绘的方式“擦除”旧图元。

二值显示中的**or**模式将待显示图像非破坏性地加到画布中的原图像上。颜色0代表白色背景，颜色1代表黑色背景，如果在白色背景上以一个灰色填充图案做或（**or**）运算，会改变原来的位并显示出灰色图案。但如果在黑色区域上以一个灰色图案做或（**or**）运算，将不会对屏幕产生影响。因此，如果在一个以砖块图案填充的多边形上以一道亮灰色的画线做或（**or**）运算，将只以画刷的图案填充砖块，而并不会像**replace**模式那样擦除砖块的边界。所以，在绘图中常用**or**模式（见习题2.7）。

二值显示中的**xor**模式可以用来反转一个目的区域。例如，为了突出显示用户选择的按钮，我们设置**xor**模式并用颜色1生成一个填充矩形图元，这样就转换了该按钮的所有像素：0 **xor** 1 = 1，1 **xor** 1 = 0。为了恢复按钮的原来的状态，我们只要保持**xor**模式并再次画矩形，这样就把这些位转换到原来的状态。SRGP内部也利用这个技术来提供定位器的橡皮线和橡皮矩形回应模式（见习题2.4）。

在许多二值图形显示中，**xor**技术被硬件（或在某些情况下的软件）用来以非破坏性的方
59 式显示定位器的光标图像。这种简单的技术也有它的缺点：当光标位于几乎50%黑色和50%白色的细致背景图案上时，光标可能无法看清。因此，许多二值显示器和大多数彩色显示器使用**replace**模式回应光标；而这种技术使软件和硬件回应机制更加复杂（见习题2.5）。

and模式可以用于有选择地将目的区域内的像素复位为颜色0。

2.4 SRGP的局限性

尽管SRGP是个能支持多种应用的功能强大的软件包，但其固有的限制使它对某些应用来说并不是最佳的选择。最显著的是，SRGP不能提供对3D几何显示的支持。另外，还有更多的甚至会影响许多2D应用的细微局限性：

- SRGP采用机器相关的整数坐标系，不适用于那些需要使用具有更高的精度、更大的范围、更加便利的浮点数的应用程序。
- SRGP以自由语义方式将图像以不连接的像素值矩阵形式存储在画布中，而不是以图形对象（图元）集合的形式，因此不支持对象级的操作，例如“删除”、“移动”、“改变颜色”。由于SRGP不记录生成当前屏幕图像的操作，一旦图像遭其他软件的破坏，它也无法刷新屏幕，另外它也不能通过重新扫描转换图元而将图像显示在具有不同分辨率的设备上。

2.4.1 应用程序坐标系统

在第1章中，我们介绍了下面的观点：对大多数应用程序，绘图仅仅是到达目标的一种手段，应用程序数据库的主要任务是支持诸如分析、模拟、验证和制造等过程。因此数据库必须

按这些过程所要求的范围和精度存储几何信息，这与显示设备的坐标系和分辨率无关。例如，VLSI CAD/CAM程序中表示的线路只有1到2厘米长，需要精确到半微米，而一个天文学程序可能要表示1到10^9光年的范围，精度一百万英里。为了达到最大的适应性和表示范围，许多应用程序在数据库中使用浮点数世界坐标系（world coordinate）存储几何特性。

这样的应用程序可以自己将世界坐标映射到设备坐标；但是，考虑映射过程的复杂性（我们将在第6章中讨论），更方便的做法是使用一个图形软件包，它接受以世界坐标表示的图元并以机器无关的方式把它们映射到显示设备中。如今市场上的廉价浮点芯片能提供与整数运算性能大致相当的浮点运算，这大大地减少了使用浮点运算所付出的时间代价——这样的花费对于适应性要求很高的应用程序来说是完全值得的。

对2D图形，提供浮点坐标的最常用的软件是Adobe公司的 PostScript（见第19章），它既是驱动硬拷贝打印机的标准页面描述语言，也是一些工作站上视窗系统的图形软件包（扩充版本 Display PostScript）。对3D浮点图形，PHIGS和PHIGS + 已经得到普遍应用，PostScript中也出现了许多3D扩充功能。 60

2.4.2 为了重新定义存储图元

现在我们考虑如何使用SRGP以不同的尺寸重新绘制一幅图，或在具有不同分辨率的显示设备上（如高分辨率打印机）绘制大小相同的图。由于SRGP对已绘制的图元没有记录，应用程序必须在换算坐标后，为SRGP重新指定整个图元集合。

如果SRGP增强了保留所有指定图元记录的功能，应用程序就可以让SRGP通过读取存储器而重新生成图元信息。这样，SRGP便可支持刷新屏幕这个常用操作。在一些图形系统中，应用程序的屏幕图像会被其他用户或应用程序发出的消息破坏。在这种情况下，除非屏幕画布可以在屏外画布中保存的冗余拷贝中刷新，否则修复被损坏的图元的惟一途径只有重新定义图元。

让软件包存储图元的最主要的优点是可以支持编辑操作，而该操作是制图或设计应用程序的核心。这些程序与我们在本章的例子中所描述的绘图应用程序很不相同。一个绘图程序（painting program）中用户可以使用可变尺寸、形状、颜色和图案的画刷绘制任意的线条。在更完善的绘图程序中用户还可以放置预定义的形状，如矩形、多边形和圆。画布的任何部分都能进行像素级编辑；对象的一部分可以被图画覆盖，画布的任意矩形区域也可以被拷贝或移动到别的地方。但是用户不能把一个已有的形状或画好的长条作为一个连续的不可见的对象进行删除或移动操作。这一限制的存在，是因为绘图程序允许画布上已有的对象被损坏或分割，这样就丧失了对象的连续性。例如，如果一个对象已被分割成散落在屏幕上不同区域的小片，那么用户指向其中一个对象碎片意味着什么呢？用户是指碎片本身还是指原来的整个对象？从本质上说，具备单个像素操作能力的应用程序是无法实现关联拾取、以及对象的拾取和编辑的。

相反，一个制图程序（drawing program）允许用户在任何时候拾取和编辑对象。这些应用程序也被称为布局编辑程序（layout editor）或图形演示程序（graphical illustrator），它们允许用户放置标准形状（也称为符号、模板或对象），并通过这些形状的删除、移动、旋转和缩放操作来编辑布局。类似地，允许用户用简单的3D对象组合复杂对象的交互程序称为几何编辑器（geometric editor）或构造程序（construction program）。

缩放、屏幕刷新和对象级编辑都需要应用程序或图形软件包对图元进行存储和重新定义。如果应用程序存储了图元，它就能完成重新定义；但是，这些操作远比它们看上去复杂得多。例如，图元的删除可以通过擦除屏幕并重新指定所有的图元（当然除了被删除的图元）完成； 61
但是，更有效的方法是以应用屏幕背景重新绘制该图元，然后重新定义可能被破坏的图元。这些操作很复杂，但要经常使用，所以有必要让图形软件包来完成它们的功能。

对象级的几何图形软件包（如GKS或PHIGS）可以让应用程序使用一个2D或3D浮点坐标系统定义对象。软件包在内部将对象存储，应用程序可以对存储的对象进行编辑，并在编辑操作需要的时候随时更新屏幕。软件包也执行关联拾取，对给定的屏幕坐标生成相应的对象ID。由于这些软件包操纵的是对象，它们不允许像素级操纵（如copyPixel和写模式）——这就是保持对象连续性的代价。因此，没有图元存储的光栅图形软件包和有图元存储的几何图形软件包都不能满足所有的需求。第7章将讨论在图形软件包中保存图元的利与弊。

通过像素复制进行图像缩放

如果应用程序和软件包都没有图元的记录（这是大多数绘图程序的特征），缩放不能通过重新定义变换后的图元端点坐标来完成。只有通过读像素和写像素操作来缩放画布的内容。放大位图/像素图图像（使它变大）的一种简单、快速的方法是通过像素复制（如图2-30a和图2-30b）。这种方法用*N*乘*N*的像素块代表每个像素，即得到了放大*N*倍的图像。

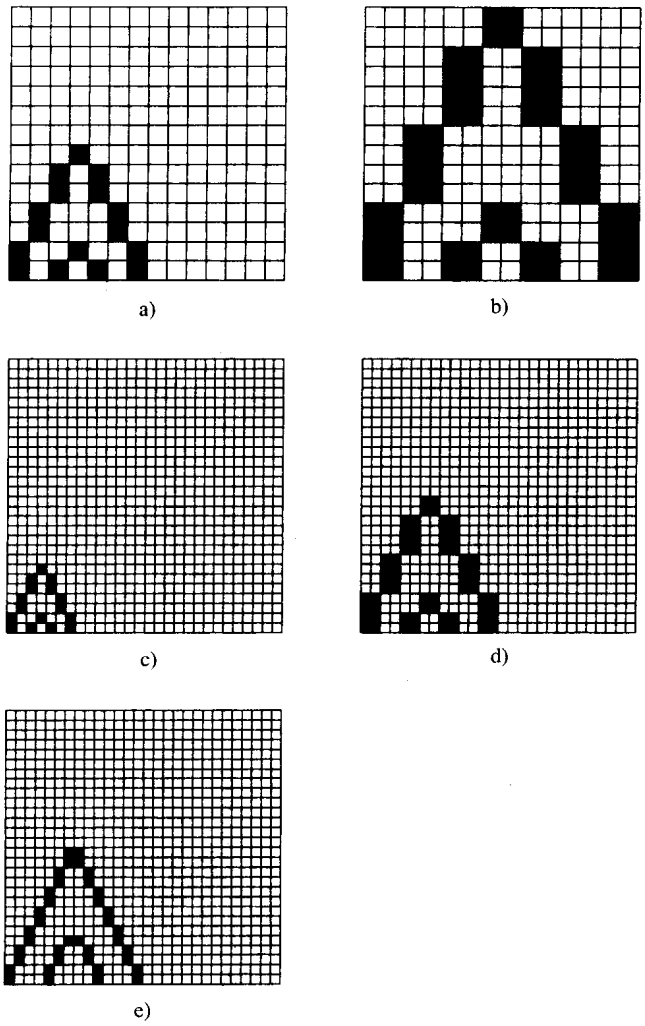


图2-30 像素复制的结果。a) 以屏幕分辨率显示的原始图像；b)以屏幕分辨率显示的放大2倍的图像；c)在分辨率是屏幕的两倍的设备上打印的原始图像；d)在c)中描述的设备上放大的图像，使用像素复制来保持图像的尺寸；e)在c)中描述的设备上打印的原始图像，使用重新扫描转换来保持图像的尺寸

像素经过复制后，图像变大，但由于没有提供新的除原始像素级以外的信息，图像也变粗糙了（比较图2-30a和图2-30b）。另外，像素复制只能将图像的尺寸放大整数倍。我们必须使用另一种技术——区域采样和过滤（在第3、14、17和19章讨论），来实现非整数的放大或任意量的缩小。过滤技术在深度大于1的像素图上能达到最好的效果。

图像缩放是经常会遇到问题，尤其在要打印绘图程序创建的图像的时候。假设现在将画布传到一个分辨率是屏幕分辨率两倍的打印机上。各像素变成其原有尺寸的一半；因此我们只能以同样数目的像素和一半的尺寸来显示原始图像（图2-30c），或是通过像素复制来产生同样大小但分辨率为打印机本身性能一半的图像（图2-30d）。总之，大小和质量不能兼顾。只有重新定义技术才能保证缩放的质量（图2-30e）。

2.5 小结

在本章中，我们讨论了一个简单而功能强大的光栅图形软件包SRGP。应用程序可以使用它绘制具有不同显示属性的2D图元，这些属性影响图元的外观。图像可以直接在屏幕画布上或在任意尺寸的屏外画布上绘制。通过设置裁剪矩形属性，绘图可被限制在画布的一个矩形区域内。除了标准2D形状的绘制，SRGP还支持画布内或画布间的矩形区域拷贝。拷贝和绘制操作受写模式属性的影响，该属性使得目的像素的当前值对其新值的确定起一定作用。

62
~
63

SRGP引入了逻辑输入设备的概念，它是物理输入设备的高层抽象。SRGP键盘设备将物理键盘抽象化，而定位器设备将鼠标、数据输入板或游戏杆等设备抽象化。逻辑设备可以在采样（轮询）模式或事件模式下工作。在事件模式下，用户动作触发的事件信息放入事件队列，供应用程序随时访问。在采样模式下，应用程序要持续地访问设备度量以检测发生的变化。

由于SRGP扫描将图元转换为其成员像素，而不存储它们的原始几何形状，因此SRGP惟一允许的编辑操作是通过绘制一个新图元或通过使用对像素块的copyPixel操作来更改单个像素。应用程序必须自行实现对象的操作，如移动、删除或缩放等，并在SRGP中重新定义更新的图像。

其他系统为图像提供了不同的功能集。例如，PostScript语言支持浮点图元和属性，包括更自由的曲线形状和裁剪功能。PHIGS子程序软件包提供了分层建模的、定义在3D浮点世界坐标系中的对象的操作功能。这些对象存储在一个可编辑的数据库中；在任何编辑操作之后，软件包将根据存储信息自动重新生成图像。

SRGP是个子程序软件包，而且现在许多开发人员则认为解释语言，例如Adobe的PostScript，可以提供更强大的功能和适应性。至于子程序软件包（整数或浮点数，保留或不保留图元的）和显示语言（例如不保留图元的PostScript）哪个能成为标准，大家意见不一。它们各自有其适用的应用领域，而且都将继续使用下去。

在第3章里，我们将看到SRGP如何通过扫描转换和裁剪进行绘图。在后面的几章中，我们先要对硬件进行简要的介绍，然后讨论图形变换和3D视图的数学原理，为PHIGS的学习做准备。

习题

2.1 SRGP在视窗环境中运行，但不允许应用程序利用多窗口的优越性：屏幕画布被映射到屏幕上的一个窗口，而其他画布都不可见。如何改进SRGP的设计和应用-程序员界面，从而允许应用程序利用视窗系统的优势？

2.2 仅当SRGP应用程序只使用颜色0和1时，它才能完全实现机器无关。设计一种增强SRGP的

策略，使SRGP能够在必要时模拟颜色，这样应用程序可以在二值显示器上正常工作的同时充分利用颜色特性。讨论这样的策略会产生的问题和冲突。

2.3 生成一个由一些普通对象移动和缩放组成的动画序列。首先通过擦除屏幕后重新定义对象的方法生成各帧。然后尝试双缓存，使用一个屏外画布作为每一帧被拷贝到屏幕画布之前被绘制到的缓冲区。比较这两种方法的结果。另外，考虑SRGP_copyPixel的用法。它在什么条件下有利于动画的生成？

64

2.4 在不使用内置定位器回应的情况下，实现一个橡皮回应交互过程。注意观察制品，尤其是在交互反馈开始和结束的时候。

2.5 在不使用SRGP的内置光标回应的情况下，实现非破坏性的光标跟踪。使用位图或像素图图案存储光标图像，图案中的0代表透明区域。在二值显示环境中实现异或（**xor**）光标，并在二值或彩色显示环境中实现置换模式光标。为了测试跟踪，应该用SRGP定位器设备执行一个采样循环，并在非空的屏幕背景上移动光标。

2.6 考虑在画图应用程序中实现以下功能：用户可以画一条异或（**xor**）线，将画刷经过的区域的颜色反转。可以设置写模式然后执行图2-28中的代码，实现起来似乎很容易。这样会带来什么样的复杂情况？提出解决方案。

2.7 一些画图应用程序提供一种“喷漆绘图”模式，这种模式以随机方式对画刷扫过的区域内的少数像素着色。每次画刷扫过一个区域，会对不同的像素着色，所以画刷经过的次数越多，区域内着色的密度就越大。在二值显示环境中实现一个喷漆绘图交互过程。（注意：普通的算法会产生条纹，或者不能实现密度的增加。你必须创建一个稀疏位图或图案库；有关创建自定义图案的信息，请见参考手册。）

2.8 在不使用SRGP的内置文本图元的情况下，为二值显示实现透明背景文本。可以使用一个屏外画布存储不同字符的位图形状，但支持的字符数不要超过6个——这不是字形设计的课程！（提示：要用两种不同的算法来处理颜色0和1。）

2.9 制图程序在执行删除操作后，可以通过以应用屏幕背景图案填充被删对象的形状来更新屏幕。当然这会损坏屏幕上的其他对象。为什么简单地通过重新定义所有与被删对象区域相交的对象来修复损失是不够的？试讨论优化图形修复问题的解决方法。

2.10 实现一个过程，在带细边界的不透明的矩形里居中绘制文本。调用者可以定义文本的颜色、背景和边界、“按钮”的中心所在的屏幕位置、宽和高的最小/最大尺寸以及字体和字符串本身。如果字符串长度超过按钮的最大行显示长度，则在合适的位置（如空格处）将该串断开，并以多行显示。

2.11 在屏幕上实现一个定值逻辑输入设备，用户可以使用鼠标改变模拟水银柱的长度来指定温度。设备的属性应当包括度量的范围、初始度量、度量的粒度（如精确到华氏2度）以及温度计的屏幕图像的长度和位置。为了测试设备，可以使用交互过程模拟一个无限等待状态（waitEvent），其中只将定值器激活。

2.12 假设通过在输入模型中增加一个屏幕上的定值器（与习题2.11中描述的类似）并同时支持事件和采样模式来定制一个SRGP实现。如果将该实现安装到一台只有一个物理定位器设备的工作站上会出现什么样的问题？提出解决方案。

2.13 实现一个“圆角矩形”图元——四个角是90°的椭圆弧的矩形。应用程序可以控制椭圆弧的半径。该图元支持轮廓和填充方式。

65

程序设计项目

2.14 实现一个下拉菜单软件包，其高层设计见2.2.6节、2.3.1节、2.3.3节中的代码段。软件包通过从输入文件读入字符串来初始化菜单条和菜单体。程序可以通过关闭菜单条使标题消失，也可以激活一个菜单（以各菜单标题在菜单条上的水平位置作为参数）来使菜单出现。

2.15 通过实现禁止所选菜单项这一功能来增强习题2.14中的菜单软件包。被禁止的菜单项应当灰显。由于SRGP不支持用画笔风格来绘制文本，所以在二值显示中，必须在写模式中覆盖实体文本以达到这种效果。

2.16 通过在用户选取菜单体上的项目时突出显示定位器当前指向的项目来增强习题2.14中的菜单软件包。

2.17 实现一个布局应用程序，该程序使用户可以在屏幕上的一个方形子域内放置图形对象。程序应可以支持椭圆、矩形和等边三角形。用户可通过点击屏幕按钮来选择一种对象类型或初始化一个操作（重画屏幕、将场景保存到文件中、从文件中恢复场景或者退出）。

2.18 在习题2.17的布局程序中增加对象编辑功能。用户应当能够对对象进行删除、移动和调整大小等操作。可以使用以下简单的关联拾取方法：在应用数据库中扫描对象，并选择第一个其矩形域范围覆盖定位器位置的对象。（这种简单的方法有个很明显的缺陷：很可能一个可见的对象无法被拾取！）注意通过突出显示当前选中的对象给用户反馈。

2.19 在习题2.17的布局应用程序中，实现覆盖优先权功能。用户应当能够压下/弹出一个对象（改变其优先权为最低/最高）。增强关联拾取来使用覆盖优先权解决冲突。试分析在关联拾取中使用优先权并提供压下/弹出功能，是如何使用户可以克服原关联拾取的不精确问题的。

2.20 使用习题2.9的结果，优化习题2.17中的布局应用程序的屏幕更新算法，使响应编辑操作时需重新定义的对象数为最小。

2.21 增强习题2.17中的布局应用程序，以便同时启动键盘和定位器以支持常用操作的快捷键。例如，敲击“d”键可删除当前选择的对象。

2.22 为习题2.17的布局应用程序所支持的三种图形对象设计并实现关联拾取的分析技术。新的技术应具有百分之百的准确性，而用户也不必再通过弹出/压下来拾取一个优先权低的可见对象。

66

第3章　二维图元的基本光栅图形学算法

光栅图形软件包近似作出数学意义上的（“理想的”）图元，这些图元定义在笛卡儿坐标系的网格点上，用适当灰度或色彩的像素点集来表示。这些像素一般作为位图或像素图存储在CPU内存或帧缓存中。在前一章，我们讨论了图形包SRGP的特点。从应用程序员的角度看，这是一个典型的光栅图形包。在本章，我们则从一个软件包实现者的角度来讨论SRGP，探讨将图元转换到像素的扫描转换算法，即如何根据图元的特点，在一个直立的矩形裁剪框内画出图元。图3-1中所示的是关于图元扫描转换及裁剪的一些例子。

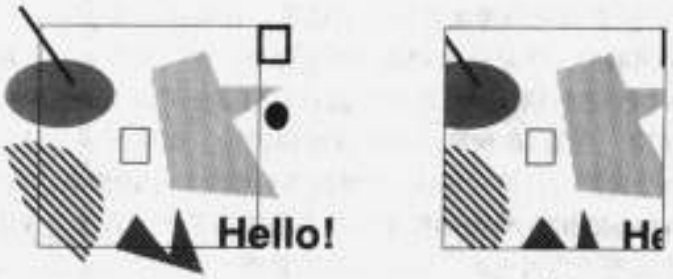


图3-1　在一个矩形裁剪框内裁剪SRGP的图元。a) 图元和裁剪框，b) 裁剪结果

在一些比较成熟和功能强的软件包中，采用了一些高级算法来处理一些SRGP并不支持的操作。这些高级算法将在第19章进行讨论。本章讨论的算法是基于二维整数笛卡儿坐标网格的，而其中大多数扫描转换算法可推广到浮点数的情况，其裁剪算法则可推广到浮点数及三维的情况。本章最后一节讨论反走样的概念，即通过调节像素的灰度来尽可能地消除图元显示时的锯齿状表现。

3.1　概述

3.1.1　显示系统体系结构的含义

在1.7节中介绍的基本概念模型中，我们给出了一个图形软件包。它介于应用程序（及其数据结构和模型）和显示硬件之间，以便为应用程序使用硬件提供一个与设备无关的接口。如图3-2所示，SRGP的程序可以分成两部分，一部分是输出流水线，另一部分是输入流水线。

在输出流水线中，应用程序根据应用模型或数据结构中存储的或推导的图元和属性对物体进行描述，并将这些信息传递给图形包，由图形包将它们裁剪和扫描转换为最终在屏幕上显示的像素。图形包中的图元生成程序确定要生成什么图元，其属性程序确定要怎样生成图元。SRGP_copyPixel程序确定对图像进行怎样的修改，而画布控制程序确定在什么地方生成图像。在输入流水线中，在显示终端的用户交互操作由图形包的采样程序或事件驱动输入程序转换成度量信息，并将这些度量信息传递给应用程序。然后，应用程序根据这些度量信息对模型或者屏幕上的图像进行修改。与输入相关的程序包括：初始化和控制输入设备的程序，以及在交互过程中从输入设备获取度量信息的程序。在本书我们将不讨论SRGP的画布管理和它的输入处理，因为这些内容主要是一些数据结构和底层的系统软件问题，与光栅图形学关系不大。

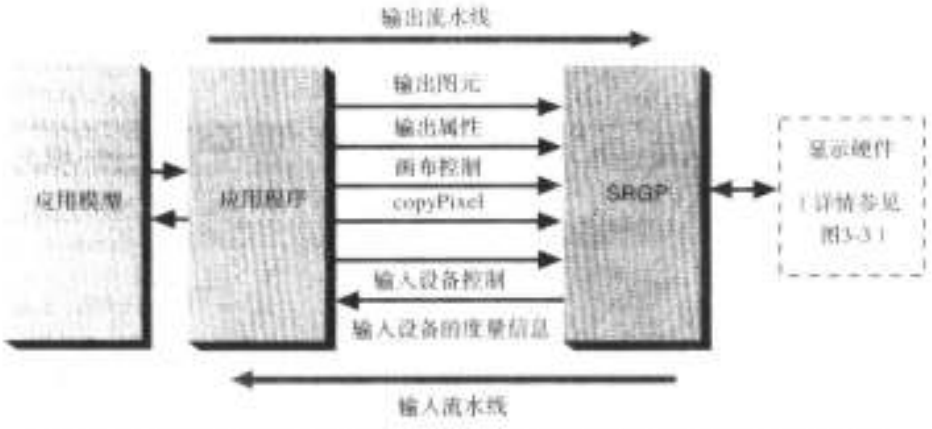


图3-2 作为应用程序和图形系统的中介，SRGP提供了输出流水线和输入流水线

实现一个SRGP图形包必须充分考虑各种显示设备。有些显示系统带有自己的帧缓存和显示控制器。这些显示控制器的工作是解释和执行绘图命令，将生成像素写入帧缓存。另有一些简单的系统只可直接由CPU进行刷新。其图形包的只输出部分可以驱动光栅硬拷贝设备。这些各具特点的硬件体系结构，在第4章和第18章有详细的讨论。在任何一个显示系统中，CPU必须能够对帧缓存中的每个像素进行读写操作。如果能够对帧缓存中的像素进行成块的读写操作，则能很方便地实现copyPixel(bitBlt)类型的操作。这一功能不是为了直接生成图元，而是为了使屏幕画面外的位图或像素图成为可见的，以及在窗口管理、菜单处理、滚动等操作中保留和恢复屏幕上的一些显示片段。

可直接由CPU进行刷新的系统的实现过程基本上是一致的，因为这些刷新工作都是由软件来完成的。然而显示控制器和硬拷贝的系统的实现过程则差别较大，这取决于硬件设备能做哪些工作以及哪些工作要由软件去完成。显然，在任何体系结构中，那些不能由硬件直接支持的图元和特征，必须通过软件进行扫描转换来生成。让我们简单地看一下体系结构及其实现的有关内容。

1. 具有帧缓存和显示控制器的显示器

如果SRGP驱动的显示控制器自己能够进行扫描转换并直接处理SRGP的图元和属性，那么SRGP只需做很少的工作。在这种情况下，SRGP只需将它关于图元、属性和写模式的内部表示转换成显示外设能够直接绘图的方式即可（参见图3-3 a）。

如果存储映射允许CPU直接访问帧缓存，显示控制器也能直接访问CPU，那么显示控制器的体系结构的功能将是最强的。此时，CPU用其自身的指令就能对单个像素或copyPixel像素块进行读写操作，而显示控制器亦能在屏幕画面外的画布上进行扫描转换，并用它的copyPixel指令在两个存储器之间或它自己的帧缓存内进行像素的传递。当CPU和显示控制器可以异步运行时，则必须具有同步机制以避免存储器读写的冲突。通常，CPU是将显示控制器作为一个协处理器进行管理。如果显示设备的显示控制器只能在它自己的帧缓存中进行扫描转换，而不能将像素写入CPU的内存，则我们要找出一种方法以便在屏幕画面外的画布上生成图元。在这种情况下，图形包可以用显示控制器在屏幕的画布上进行扫描转换，而对于屏幕画面外的画布，就必须用它自身的软件进行扫描转换。当然，图形包可以通过copyPixel操作将硬件数据扫描转换形成的帧缓存中的图像复制到屏幕画面外的画布。

67
~
69

2. 只有帧缓存的显示器

在没有显示控制器时，SRGP就必须自身进行扫描转换以生成屏幕画面外的画布和帧缓存

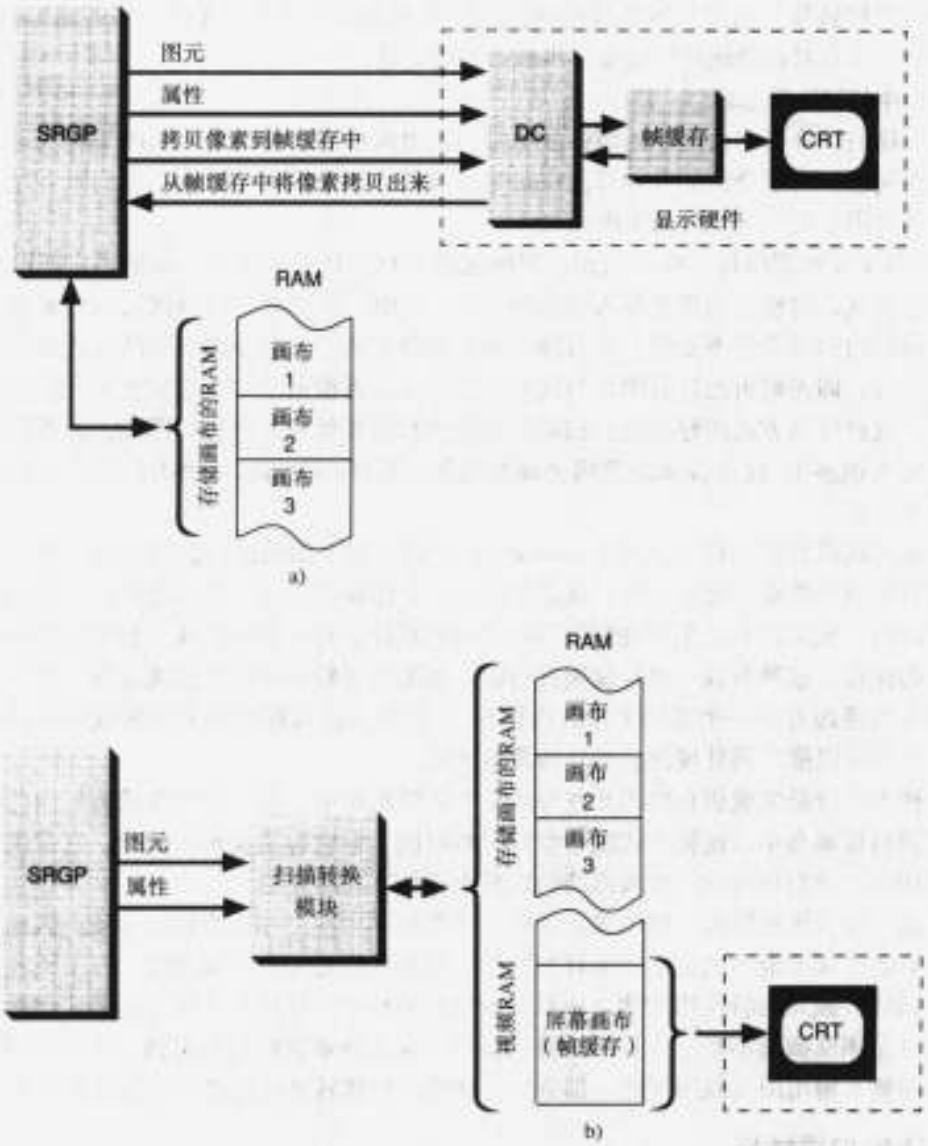


图3-3 SRGP驱动的两种显示系统。a) 带有显示控制器和帧缓存的显示外设，b) 无显示控制器，只有存储共享的帧缓存的显示系统

图像。这种情况的一个典型结构如图3-3b所示，SRGP驱动一个存储共享的帧缓存。请注意，图中只给出了那些作为帧缓存和存储由SRGP管理的画布的存储器。另有一些存储器是为一般 70
软件和数据（包括SRGP自身）所用的，则未包括在内。

3. 硬拷贝设备

如同在第4章将介绍的那样，具有不同功能的硬拷贝设备多种多样。最简单的设备一次只接受一条扫描线，并且在胶片或纸上绘制时要靠软件来定位扫描线。对于这种简单的硬件，SRGP必须生成完整的位图或像素图，并一次扫描出一条线传递给输出设备。好一点的设备，一次可接受一整页（帧）。而功能更强的设备则自身带有扫描转换硬件，它们通常被称为光栅图像处理器（RIP）。作为最高档的外设，PostScript打印机有自己的内部“引擎”，以读取描述页面的PostScript程序（该描述与设备无关），这些“引擎”能解释这种程序以生成图元和属性，

并随后进行扫描转换。由于基本的裁剪和扫描转换算法在本质上与光栅设备的输出技术无关，因此，我们在本章对各种硬拷贝设备不再做过多的论述。

3.1.2 软件中的输出流水线

在此，我们只分析驱动简单帧缓存显示器的输出流水线，只是为了引出用软件进行裁剪和扫描转换的问题。将要介绍的各种算法在讨论时与具体设备无关。因此，它们既可以用于软件实现，也可以用于硬件或微代码实现。

对于SRGP要处理的每个输出图元，图形包都要*扫描转换*该图元，即根据它们可用的属性和当前的写模式，将相关的像素写入当前画布中。同时，要用裁剪框对图元进行*裁剪*，即图元中不在裁剪框内的像素将不显示。裁剪的处理有多种方式。一种显而易见的方式是在扫描转换以前进行裁剪，即先解析地计算图元与裁剪框的交点，再根据这些交点生成图元被裁剪后的结果。显然，这种处理方式的好处是：扫描转换操作只需要处理裁剪后的图元，而不是原来的图元（它可能大很多）。这种技术经常用来裁剪线条、矩形和多边形，因为处理它们的裁剪算法相当简单和有效。

最简单的裁剪方式，称为*截剪*（scissoring），即对整个图元进行扫描转换，但只显示位于画布上裁剪框内的像素。理论上讲，就是在显示一个像素前将其坐标与裁剪框边界的(x, y)坐标区间进行比较。但实际上，正如我们后面将讨论的那样，对一条扫描线上相邻的像素可以避免一些这样的比较。这种剪裁一般是快速进行的。如果边界检查可以快捷地进行（例如，利用微代码或指令高速缓存的一个紧凑工作的内循环），这种方法可能实际上比先裁剪再扫描转换的方法快。它也可以推广到处理任意形状的裁剪区域。

第三种方式就是生成所有的图元并写入一个临时画布中，然后只对在裁剪框内的像素进行复制以送到目标画布中。这种方式既费空间又费时间，但它容易实现，因此，它常用来处理文
71 字。在第19章，我们将讨论一些数据结构以减少这种方式的开销。

每生成一幅图像或修改一幅图像，光栅显示都要调用裁剪和扫描转换算法。因此，裁剪和扫描转换算法不仅要能生成视觉效果好的图像，而且执行还要尽可能地快。在下面各节的详细讨论中，扫描转换算法将运用*增量方法*来减少每次循环中的计算量（特别是乘法和除法运算），并且其运算是用整数而不是浮点算术运算来进行。在第18章我们还将看到，通过使用多重并行处理器来对整个输出图元或图元的一部分同步地进行扫描转换还将进一步加速其运算。

3.2 直线的扫描转换

对直线的扫描转换，就是要在二维光栅格上计算接近或位于理想的无限细的直线上的像素的坐标。理论上，这些像素应尽可能地接近线，并且这些像素的队列要尽可能地直。假设对线近似表达的宽度是一个像素，那么，这样的线会有一些什么样的特点呢？如果线的斜率在1和 -1之间（包括1和 -1），则每一列上必定只有一个像素被显示；若线的斜率在此范围之外，则每一行上必定只有一个像素被显示。无论长度和方向如何，整条线应该以相同的亮度尽可能快地生成。此外，还要为画线提供其他的功能：所画的线可以宽于一个像素，其宽度相对于理论上的线中心对称；它可以具有不同的线型和笔型，可以具有高质量图形所需的其他效果。例如，在程序员的控制下，端点区域的形状可以是斜的、圆的和勾角的。在每个像素具有多位的显示系统中，运用反走样技术可以加强像素的灰度变化能力，这样，我们就可以尽可能地改善对线进行离散的拟合所产生的锯齿形状。

至此，我们只讨论“最优”线的情况，即每列只显示一个像素的斜线（若是陡的线，则每

行上只有一个像素显示）。在本章后面的部分，我们将讨论宽的图元及其处理方式。

我们知道，为了可视化几何属性，SRGP将一个像素表达成一个圆形的点，并且其中心就是像素在整数栅格上的(x, y)坐标。这种表达方式便于拟合CRT电子光束的近似圆形的横截面，但实际显示时，光束斑点之间的间距在不同的系统中变化很大。在有些系统中，相邻的斑点是相重叠的；而另有一些系统中，垂直方向相邻的像素之间不相重叠；在大多数系统中，水平方向上的间距要比垂直方向上的间距小。在不同的系统中，坐标系的表达也有差异，比如在苹果牌Macintosh机器中，像素位于相邻栅格线构成的矩形框内，而不是在栅格线上。这样，在数学上由两个对角点定义的矩形在这种机器上就由此矩形内的所有像素来表示。如此就存在零宽度的画布，比如，由(x, y)到(x, y)定义的矩形不包含任何像素。而在SRGP中，这个画布会有一个在该点的像素。至此，我们依然将像素表达为不相接的圆，它们的圆心位于栅格点上。只有在讨论反走样时，此定义才略有改变。

图3-4中所画的是放大了很多倍的一条单个像素宽的线和它要拟合的真实的线，实心圆表示被显示的像素，空心圆则表示没有被显示的像素。在真实的屏幕上，圆形像素的直径要大于像素间的间距，所以这种符号表示实际上是夸大了像素的离散性。 72

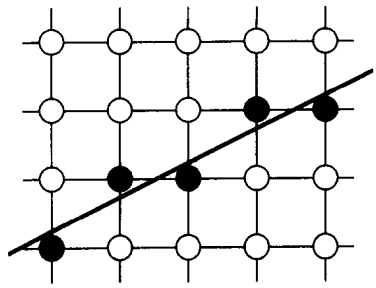


图3-4 实心圆表示扫描转换一条线所显示的像素

因为SRGP的图元是定义在整数栅格上的，所以线的端点是整数坐标。但实际上，若先用一个矩形框裁剪线段，则线段与裁剪边的交点作为裁剪后的端点，其坐标很可能是非整数的。对于浮点数光栅图形包，情况也一样。（在3.2.3节中，我们将讨论非整数交点的情况。）在下面的讨论中，我们将假设线的斜率$|m| \leqslant 1$。至于其他斜率的情况，只需做适当的调整就可以处理。而对于水平线、垂直线、斜率为± 1的这些常见的线，只需作为平常的特例处理即可，因为它们只可能穿过像素中心（见习题3.1）。

3.2.1 基本增量算法

对线的扫描转换，最简单的策略就是将斜率m计算为$\Delta y/\Delta x$，然后，从最左端的点开始，对x每次递增一个单位，而对每个x_i，计算其相应的$y_i = mx_i + B$，并显示坐标为（x_i，Round(y_i)）的像素，其中，Round(y_i) = Floor(0.5 + y_i)（即对0.5 + y_i进行取整）。这种计算是为了选择最接近线的像素，即到实际的线距离最短的像素[⊖]。当然，这种简单的方式并不很有效，因为每次循环都要用浮点（或二进制分数）计算一次乘法、一次加法并调用一次取整运算。我们可以去掉其中的乘法，由于

$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$$

当$\Delta x = 1$时，$y_{i+1} = y_i + m$。

因此，x每增加一个单位，y就加上一个m，m是线的斜率。对于线上的所有点(x_i, y_i)，我们知道，如果$x_{i+1} = x_i + 1$，那么$y_{i+1} = y_i + m$；也就是说，x和y的值可以根据前一点的值推算出来（见图3-5）。这就是将该方法称为增量算法的原因：在每一步，我们只需根据前一步的结果进行增量计算即可。 73

增量运算从一个端点的整数坐标(x_0, y_0)开始。值得注意的是，增量算法避免了对y轴上的

⊖ 在第19章，我们将为直线和一般曲线讨论各种度量近似性的方法，也称为误差测量。

截距B进行任何处理。如果$|m|>1$，当x变化一步时，y变化的步长将大于1。此时，我们就得将x和y的角色进行倒换，即y每次增加一个单位，x变化的增量为$\Delta x = \Delta y/m = 1/m$。图3-6中的程序Line实现了这种增量运算。其起始点必须是左端点，并且只适于$-1 \leqslant m \leqslant 1$的情况。其他斜率的情况，只需要相应地做一些调整即可。在此我们省略了对水平线、垂直线或对角线等特殊情况进行检测的处理。

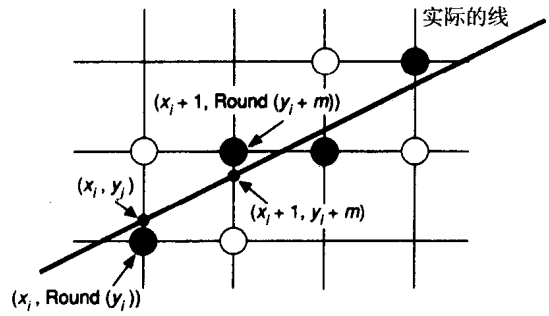


图3-5 关于(x_i, y_i)的增量计算

```
void Line (                                  /* 设 -1≤m≤1, x0 < x1 */
      int x0, int y0,                        /* 左端点 */
      int x1, int y1,                        /* 右端点 */
      int value)                             /* 赋给线上像素的值 */
{
   int x;                                    /* x以单位步长从x0增长到x1 */

   double dy = y1 - y0;
   double dx = x1 - x0;
   double m = dy / dx;
   double y = y0;
   for (x = x0; x <= x1; x++) {
      WritePixel (x, Round (y), value);      /* 置像素的值为value */
      y += m;                                /* y移动的步长是斜率m */
   }
} /* Line */
```

图3-6 增量的线扫描转换算法

Line中的WritePixel是由设备层的软件提供的一个底层程序，它的作用是将一个值写入画布中的一个像素，该像素的坐标由它的前两个参数确定[⊖]。在此，我们假设只在替换（**replace**）模式下进行扫描转换。对于SRGP的其他写模式，我们必须用一个底层的ReadPixel程序去读取目标像素，并将该像素与源像素进行逻辑操作，然后用WritePixel将结果写入目标像素。

这一算法常被称为*数字微分分析器*（DDA）算法，DDA是用数值方法求解微分方程的一种机械设备，即根据x和y的一阶导数，在x和y方向上渐进同步地以小步长移动，由此生成连续的像素坐标(x, y)。在目前所考虑的情况下，x方向上的增量为1，y方向上的增量为$dy/dx = m$。由于计算机中实数变量的精度是有限的，不精确的m的重复迭加会产生累加误差并导致偏离实际的$\text{Round}(y_i)$值；然而对于大多数的（短）直线，将不会引起什么问题。

3.2.2 中点线算法

上节中Line程序的缺点是：对y值取整要花费时间，而且，因为斜率是一个小数，y和m必须是实数或二进制小数。为此，Bresenham提出了一个只使用整数运算的经典算法[BRES65]，它能够根据前一个已计算的坐标(x_i, y_i)进行增量运算得到(x_{i+1}, y_{i+1})，而不必进行取整操作。该算法也可扩充其浮点数功能以处理端点坐标是任意实数的直线。Bresenham的增量技术还可以

⊖ 如果不存在这样的底层程序，可以使用SRGP_pointCoord程序，请见SRGP的参考手册。

用来对圆进行整数的计算，尽管它还很难处理一般的二次曲线。因此，我们使用一种稍有不同的方法，叫作中点技术。该技术最早由Pitteway发表[PITT67]，而后Van Aken[VANA84]和另外一些研究人员对它进行了一些改进。Van Aken[VANA85]指出：中点方法在处理直线和整数型的圆时就简化成了Bresenham方法，会产生同样的像素。Bresenham证明并得出结论：通过对误差（离实际图元的距离）进行最小化处理，他的直线和圆的整数型算法能给出对精确图元的最好的逼近结果[BRES77]。在[KAPP85]中，Kappel讨论了多种误差标准的结果。

在下面的讨论中，我们将假设直线的斜率在0和1之间。对于其他斜率的情况，根据主轴对称的方式作适当调整就能处理。我们还假设左下端点为(x_0, y_0)，右上端点为(x_1, y_1)。

考察图3-7中的线，黑色的圆表示已被选择的像素，当前被选中的像素后面的两个空心圆表示下一步要选择的候选像素。假设我们刚选择了在(x_p, y_p)的像素P，下一步要选择的像素可能是其右边的第一个像素E（称为东像素），也可能是其右上的第一个像素NE（称为东北像素）。假设Q是要被扫描转换的线与栅格线$x = x_p + 1$的交点，根据Bresenham的方法，先计算E和NE到Q的垂直距离，然后根据这两个距离的差的符号挑选离Q最近的像素，作为被扫描转换线的最好逼近。用中点方法时，我们是考察中点M在线的哪一边。显然，如果中点在线的上方，像素E就更靠近线；否则，中点在线的下方，则是像素NE更靠近线。当然，线也可能在E和NE之间的中点穿过或两个像素都在线的同一边。但不管哪种情况，中点检测的方法都会选择最靠近的像素。此外，这样处理的误差（即被选择的像素到实际线的垂直距离）一定小于等于1/2。

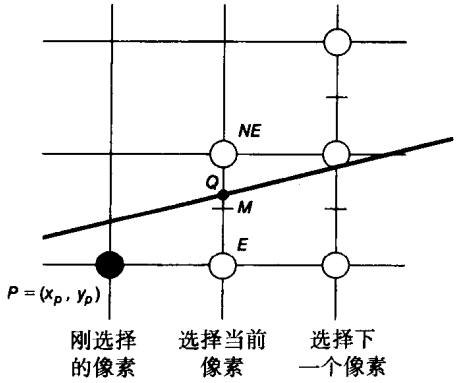


图3-7 中点线算法的像素栅格，包括中点M和两个候选像素E和NE

在图3-7中，被选择的像素是NE。现在，我们讨论如何计算中点在线的哪一边。假设直线由其隐式函数[⊖] $F(x, y) = ax + by + c = 0$表示。（在斜截式中，$y$的系数$b$与$y$方向上的截距$B$无关。）如果$dy = y_1 - y_0, dx = x_1 - x_0$，其斜截式可以写为

$$y = \frac{dy}{dx}x + B$$

因此，

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$$

其中$a = dy, b = -dx, c = B \cdot dx$[⊖]。

容易证明，对于线上的点，$F(x, y)$等于0；对于在线下方的点，$F(x, y)$是正数；而对于在线上方的点，$F(x, y)$是负数。运用中点法则时，我们只需计算$F(M) = F(x_p + 1, y_p + 1/2)$并考察它是正数还是负数。由于是根据在点$(x_p + 1, y_p + 1/2)$的函数值进行判定，我们定义一个判定变量$d = F(x_p + 1, y_p + 1/2)$。根据定义，$d = a(x_p + 1) + b(y_p + 1/2) + c$。对于$d>0$和$d<0$，我们分别选$NE$和$E$。如果$d = 0$，两者都可以选，我们在此就选$E$。

当栅格线移到下一条时，我们考察中点M的位置和d的值是如何变化的。当然，它们都依赖于我们是选择了E还是NE。如果选择了E，M就沿x方向递增一步。那么

74 ~ 76

⊖ 这个函数形式可以很方便地扩展成关于圆和椭圆的隐式方程。

⊖ 为确保中点算法的正确操作，选择a为正数是很重要的；在此，由于$y_1>y_0$，如果dy是正数，该条件可得到满足。

$$d_{new} = F(x_P + 2, y_P + \tfrac{1}{2}) = a(x_P + 2) + b(y_P + \tfrac{1}{2}) + c$$

但是，

$$d_{old} = a(x_P + 1) + b(y_P + \tfrac{1}{2}) + c$$

从d_{new}中减去d_{old}得到一个增量差，因此，$d_{new} = d_{old} + a$。

我们将选择E后所得的增量称为Δ_E，$\Delta_E = a = dy$。换句话说，不必直接计算$F(M)$，我们就能根据当前的判定变量值得到下一步判定变量值，即简单地加上Δ_E。

如果选择了NE，M在x和y方向上都要移动一步。那么，

$$d_{new} = F(x_P + 2, y_P + \tfrac{3}{2}) = a(x_P + 2) + b(y_P + \tfrac{3}{2}) + c$$

从d_{new}中减去d_{old}得到一个增量差，我们将其写为

$$d_{new} = d_{old} + a + b$$

我们将选择NE后所得的递增量称为Δ_{NE}，$\Delta_{NE} = a + b = dy - dx$。

基于上面的讨论，增量的中点技术可以概括为：在每一步，我们可根据上一步所得的判定变量值的符号去选择下一个像素；然后根据所选择的像素，用Δ_E或Δ_{NE}递增判定变量的值。

因为第一个像素就是第一个端点(x_0, y_0)，我们能直接计算d的初始值，由此来选择E或NE。第一个中点的坐标是$(x_0 + 1, y_0 + 1/2)$，因此

$$\begin{aligned} F(x_0 + 1, y_0 + \tfrac{1}{2}) &= a(x_0 + 1) + b(y_0 + \tfrac{1}{2}) + c \\ &= ax_0 + by_0 + c + a + b/2 \\ &= F(x_0, y_0) + a + b/2 \end{aligned}$$

由于(x_0, y_0)是在线上的点，所以$F(x_0, y_0)$等于0；因此，d_{start}就是$a + b/2 = dy - dx/2$。根据d_{start}我们可得到第二个像素，然后依此类推得到后续的像素。为去掉d_{start}中的小数部分，我们对原来的F略加修改，即将它乘以2变成$F(x, y) = 2(ax + by + c)$。这样，方程中的常量和判定变量都被乘以2。但这不会影响判定变量值的符号，因此也不会影响中点检测的效果。

我们已经看到，计算d_{new}时只需用简单的加法而无需费时的乘法。特别是，如图3-8中的中点算法所示，算法中的内循环非常简单。在这个循环中，首先是检测d以选择一个像素，但实际上，我们是在修改了判定变量后再对当前像素在x和y方向上递增（这是为了与绘制圆和椭圆的算法一致）。注意，图3-8中所示的算法只适应于斜率在0和1之间的线，对于一般情况的处理被留作为习题 3.2。在[SPRO82]中，通过对原先基本的算法进行一系列的程序变换，Sproull很巧妙地推导出这一算法的Bresenham方法。而对于圆和椭圆算法却尚未见到类似的推导，但其后正如我们将会看到的，中点技术事实上作了推广。

77

```
void MidpointLine (int x0, int y0, int x1, int y1, int value)
{
    int dx = x1 - x0;
    int dy = y1 - y0;
    int d = 2 * dy - dx;              /* d的初始值 */
    int incrE = 2 * dy;               /* 选择E时所用的增量 */
    int incrNE = 2 * (dy - dx);       /* 选择NE时所用的增量 */
    int x = x0;
    int y = y0;
```

图3-8 中点线扫描转换算法

```
    WritePixel (x, y, value);          /* 起始像素 */

    while (x < x1) {
        if (d <= 0) {                  /* 选择E */
            d += incrE;
            x++;
        } else {                       /* 选择NE */
            d += incrNE;
            x++;
            y++;
        }
        WritePixel (x, y, value);      /* 所选择的像素最靠近线 */
    } /* while */

} /* MidpointLine */
```

图3-8（续）

如图3-9所示，画一条从点(5, 8)到点(9, 11)的线，d值的连续变化是：2, 0, 6和4，其像素的选择顺序相应地为NE, E, NE和NE。为了清晰地揭示算法的几何特性，像素的间距和所画的像素均人为地放大了，这使得线的锯齿状特别明显。基于同样的原因，在下面各章节图中所画的图元都要比它们在真实屏幕上所显示的要粗糙。

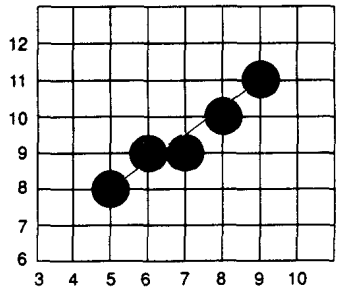


图3-9 从点(5, 8)到点(9, 11)的中点线

3.2.3 补充要点

1. 端点顺序

画线时必须考虑的一个问题是：画一条从P_0到P_1的线应该和画一条从P_1到P_0的线具有相同的像素序列，也就是说，画出的线应该与线的端点顺序无关。在前面的讨论中，像素选择有赖于线的方向的惟一情况是线正好穿过中点而判定变量是0的时候。此时，从左向右扫描时我们会选择E（东像素）。根据对称，如果是从右向左扫描，我们就要选择W（西像素），而这样所选择的像素，就要比从左向右扫描时所选择的相应的像素在y方向上高出一个单位。因此，在从右向左扫描时，当判定变量$d = 0$时我们就应该选择SW（西南像素）。关于其他斜率的情况，需做类似的调整。

一般地，通过改变给定线的端点顺序，就能使扫描转换的操作总是按同一个方向进行。但在显示具有线型的线段时，就不能这样处理。线型操作应该总是从线的起始点开始使用设定的写掩码。而按前述方法（交换起始点），使得起始点总是位于左下端的点，与线的方向无关，这样处理可能会产生不正确的效果。例如，对于点划形式的线型，如111100，当我们希望这个线型从所定义的起始点开始操作而不是直接从左下点开始时，就可能出现问题。再者，如果算法以规范次序排列端点，线型在运作时可能对一个线段是从左往右进行，而对另一个邻接的线段则可能是从右往左进行，因为它们的斜率不同。这样，在这两条线段的共享点处就可能出现不希望的断裂情况，而实际上线型在相邻的线段间应该是无缝过渡的。

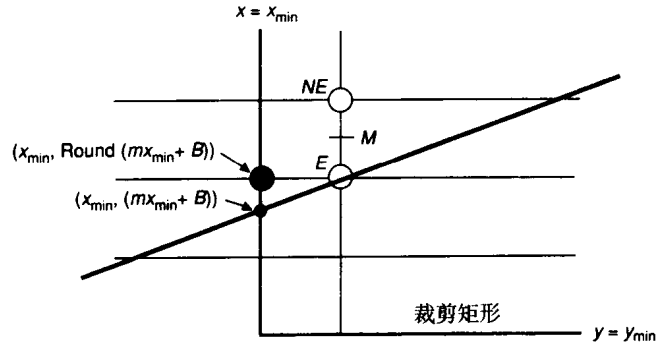
2. 起始于裁剪矩形的边的线

我们必须修改我们的算法以处理被裁剪算法裁剪掉一部分的线。（裁剪算法将在3.12中讨论。）图3-10a 中画了一条线，它被裁剪矩形的左边剪去一段，左边是：$x = x_{min}$。这条线与左边线的交点的x坐标是一个整数，而y坐标是一个实数。根据增量算法，在x_{min}值处为这条线所选

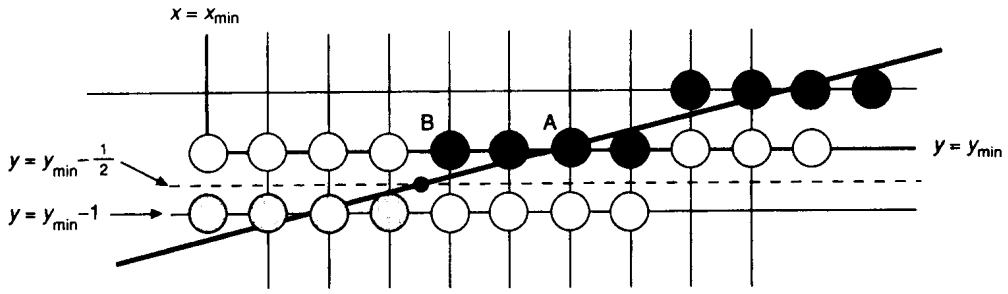
择的像素正是左边上的像素(x_{min}, Round(mx_{min} + B))[⊖]。给出了此初始像素值以后，我们还须初始化下一列上像素E和NE之间中点处的判定变量。在此，选择正确的像素序列是很重要的。否则，将这条线在边界x_{min}处裁剪后，再用整数型的中点线算法对裁剪后从(x_{min}, Round(m x_{min} + B))到(x_1, y_1)的线进行扫描转换，就会产生偏差，因为线的斜率发生了变化。

如果线与裁剪矩形的水平边而不是垂直边相交，则情况要复杂一些，如图3-10b所示。对于所示的线为窄线类型时，会有多个像素在对应于裁剪区域的底边的扫描线$y = y_{min}$上。这些像素应该都包含在裁剪矩形内。但如果只简单地计算这条线与扫描线$y = y_{min}$的交点，再对交点的x坐标取整，就会得到像素A，而不是所示的几个像素中最左的B。从图中可清楚地看到，像素B是当线刚刚越过中点$y = y_{min} - 1/2$时所在位置的右上方向的像素。因此，我们只需求线与水平线$y = y_{min} - 1/2$的交点，再对交点的x坐标取整，就能得到第一个像素B的坐标(Round($x_{y_{min}-1/2}$), y_{min})。

78
~
79



a)



b)

图3-10 从裁剪边界开始画线。a) 与一条垂直边相交，b) 与一条水平边相交（灰色的像素表示它们在线上但不在裁剪矩形内）

最后要指出的是，即使端点是在一个浮点数光栅图形包中定义的，增量中点算法亦同样适用，只是增量由一个浮点数表示，算术运算也按浮点数方式进行。

3. 根据斜率变化线的灰度

观察图3-11中所示的两条扫描转换线。对角线B的斜率是1，因此它的长度是水平线A的$\sqrt{2}$倍。而这两条线都是由相同数目的像素（10个）表示的。如果每个像素上的灰度是I，那么线A的单位长度上的灰度就是I，而线B上的就仅是$I/\sqrt{2}$。这种差别很容易被人们觉察出来。尚若显示器只有2种显示状态则无法消除这种差别。但在n位像素的显示系统中，我们可以根据斜率调整灰度以消除这种差别。在3.17节中我们将会看到，反走样技术能生成效果更好的线。

80

⊖ 当$mx_{min} + B$正好位于两条水平栅格线的正中时，我们实际上是向下取整，当$d = 0$时，结果是选择像素E。

它的处理是将线看成一个很薄的矩形，并对每一列上靠近或内含于矩形的多个像素计算出适当的灰度。

将线看成一个矩形也是生成宽线的一种方法。在3.9节中，我们将讨论如何修改基本的扫描转换算法以处理宽的图元以及具有不同线型和笔型等属性的图元。在第19章我们还将讨论多种增强基本算法功能的方法，比如怎样处理端点的形状，怎样连接具有多像素宽度的线等。

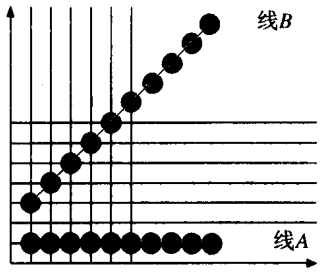


图3-11 光栅线的灰度值随斜率变化

4. 由线构成的轮廓图元

知道如何扫描转换线之后，我们如何扫描转换由线构成的图元呢？对于折线的情形，可以每次扫描转换一条线段。对于矩形和多边形这些定义区域的图元，在扫描转换时可以一次扫描转换一条线段，但这样会画出一些不在图元所定义区域内的像素（参见3.5节和3.6节对这问题进行处理的一些特殊算法）。必须注意，对连接边共享的顶点只能画一次，因为对一个顶点画两次可能会改变色彩，比如当写屏幕的模式是**xor**时，就会变成背景色；若写到胶片上，灰度值就会加倍。实际上，两条靠近或交错的线会共享多个像素。参见19.7节和习题3.8中关于这个问题的讨论，以及关于一条折线和一列连接起来的线段之间的差异的讨论。

3.3 圆的扫描转换

尽管SRGP中不支持圆但它支持椭圆这种图元，因此可以将圆弧当作一种特殊的椭圆弧并根据圆的八重对称性很好地生成圆。这种处理对圆的裁剪和扫描转换都是适用的。圆心在坐标原点的圆方程为$x^2 + y^2 = R^2$。若圆心不在坐标原点，通过平移操作就能将圆心移到坐标原点，但在其后的扫描转换时，像素的位置要做适当的平移修正。对圆进行扫描转换有几种容易实现但效率不高的方法。比如，从圆的隐式方程求解y，我们可得到$y = f(x)$的显示表达式为

$$y = \pm\sqrt{R^2 - x^2}$$

为了画出四分圆的一部分（其余四分圆根据对称性画出），我们可以将x一个单位一个单位地从0递增到R，并在每一步求出$+y$的值。但这样处理要进行乘法和二次开方的运算，其效率不高。特别是当x靠近R时，圆会有较大的间断，因为圆的斜率在此变得无穷大（见图3-12）。另有一种效率差不多的方法，就是将θ一步步地从0°增长到90°，并画点$(R\cos\theta, R\sin\theta)$；该方法能避免出现大的间断。

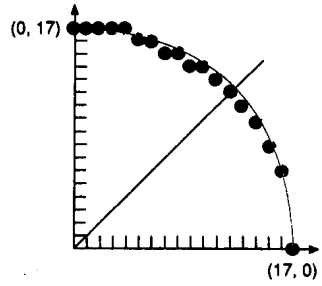


图3-12 x按单位步长变化生成的四分圆，y根据x的值计算并取整。由于相应于每个x值只生成一个y值，因而产生了间断

3.3.1 八方向对称性

利用圆的对称性，我们可以改进上节所介绍的画圆方法。首先分析圆心在坐标原点的圆。如图3-13所示，若点(x, y)在圆上，则根据多种对称性，很容易得到另外7个在圆上的点。为此我们只需要计算0°到45°之间的一段圆弧就能得到整个圆。对于圆心在坐标原点的圆，用程序CirclePoints就能将8个各相对称的点显示出来（这个函数可以很容易地进行修改以处理圆心不在坐标原点的圆）：

```
void CirclePoints (int x, int y, int value)
{
    WritePixel (x, y, value);
    WritePixel (y, x, value);
    WritePixel (y, −x, value);
    WritePixel (x, −y, value);
    WritePixel (−x, −y, value);
    WritePixel (−y, −x, value);
    WritePixel (−y, x, value);
    WritePixel (−x, y, value);
} /* CirclePoints */
```

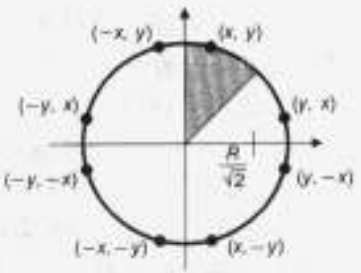


图3-13 圆上的8个各相对称点

在$x = y$时，我们并不想调用CirclePoints程序，因为对于4个这样的点，每个都将被画两次。代码只需稍做修改以处理其边界情况。

81
82

3.3.2 中点圆算法

Bresenham[BRES77]开发了一种增量圆生成器，其效率要高于我们此前所讨论的方法。该算法假设用笔绘图仪的方式递增地沿着圆弧行进，能为圆心在坐标原点的圆产生所有在圆上的点。对于圆心坐标和半径均为整数的情况，我们运用中点准则推导了一个类似的算法，它能产生同样的一组优化的像素。特别是，此方法的程序代码本质上与专利4,371,933[BRES83]中所给出的一致。

我们只考虑角度为45°的一段圆弧，即从$x = 0$到$x = y = R/\sqrt{2}$的第二个八分圆弧，并调用程序CirclePoints画出圆上其余的所有点。与中点线算法相似，这里所用的方法是：在2个像素之间的中点处给出一个评估函数值，并据此在2个像素中选择更靠近圆的那个像素。如图3-14中所示，如果在(x_p, y_p)的像素P是刚被选择的最靠近圆的一个像素，那么下一个像素就要在E和SE之间选择。

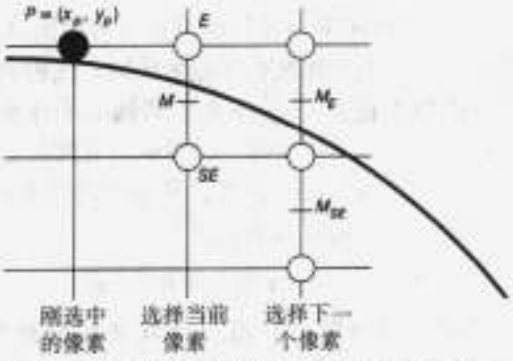


图3-14 中点圆算法的像素栅格，E和SE是两个候选像素，M是它们之间的中点

设函数$F(x, y) = x^2 + y^2 - R^2$。对于圆上的点，此函数的值是0；对于圆内的点，函数值是正的；而对于圆外的点，函数值则是负的。这样，如果像素E和SE之间的中点在圆外，则SE更靠近圆。相反，如果中点在圆内，则E更靠近圆。

与画线类似，我们选择像素是根据判定变量d的值，即函数$F(x, y)$在中点处的值：

$$d_{old} = F(x_p + 1, y_p - \tfrac{1}{2}) = (x_p + 1)^2 + (y_p - \tfrac{1}{2})^2 - R^2$$

如果$d_{old}<0$，就选择像素E，并将当前中点的x坐标增加一个单位，以得到下一个中点，从而得到

$$d_{new} = F(x_p + 2, y_p - \tfrac{1}{2}) = (x_p + 2)^2 + (y_p - \tfrac{1}{2})^2 - R^2$$

由于$d_{new} = d_{old} + (2x_p + 3)$；因此，增量$\Delta_E = 2x_p + 3$。

如果$d_{old} \geq 0$，就选择像素SE[⊖]，而下一个中点是将当前中点的x坐标增加一个单位，y坐标减少一个单位，从而得到

⊖ 当$d = 0$时选择SE，这与我们在画线算法中的选择是不同的。当然如此选择是随意的。读者如果以手工的方式模拟这个算法的运行就会发现，在$R = 17$时，这样选择会改变一个像素。

$$d_{\text{new}} = F(x_P + 2, y_P - \tfrac{3}{2}) = (x_P + 2)^2 + (y_P - \tfrac{3}{2})^2 - R^2$$

由于$d_{\text{new}} = d_{\text{old}} + (2x_p - 2y_p + 5)$，可得增量$\Delta_{SE} = 2x_p - 2y_p + 5$。

回顾画线时，Δ_E和Δ_{SE}都是常量；但在处理圆这样的二次函数时，Δ_E和Δ_{SE}在每一步都要变化，且依赖于上一步所选择的像素P的坐标x_p和y_p。由于计算Δ_E和Δ_{SE}的公式是以(x_p, y_p)为自变量表达的，我们将P称为估值点。在每一步，我们直接将上一步所选择像素的x和y坐标值代入增量计算公式，就得到了新的增量。由于计算公式是线性的，其直接计算的开销并不大。

总之，和画线时一样，此算法的每次循环都做同样的两步：第一是根据上一次循环时所计算的判定变量d值的符号选择一个像素；第二是根据所选择的像素计算增量Δ，并用它修改判定变量d。与画线算法惟一的不同之处是：该算法在修改d时要根据估值点计算一个线性函数。

至此，我们还需讨论的就是计算初始条件。由于限定该算法处理的圆半径是整数，并只画第二个八分圆弧，因此，圆的起始像素是$(0, R)$。由于下一个中点的位置是$(1, R - 1/2)$，因此$F(1, R - 1/2) = 1 + (R^2 - R + 1/4) - R^2 = 5/4 - R$。该算法的实现如图3-15所示。显而易见，这个算法的结构与画线的算法很相似。

```
void MidpointCircle (int radius, int value)
/* 假设圆的中心在原点 */
{
    int x = 0;
    int y = radius;
    double d = 5.0 / 4.0 - radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)              /* 选择E */
            d += 2.0 * x + 3.0;
        else {                  /* 选择SE */
            d += 2.0 * (x - y) + 5.0;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

图3-15 中点圆扫描转换算法

因为d的初始值有小数，所以这个算法必须做实数的算术运算。尽管只需做适当的修改，该算法就能处理圆心不在整数光栅格点的圆和半径不是整数的圆，但是我们还是希望有一个效率更高的只进行整数运算的算法。为此，我们可对这个算法做一些简单的编程变换，以避免小数的出现。

首先，我们定义一个新的判定变量$h = d - 1/4$，即将程序中的d替换为$h + 1/4$。这样在初始化时，$h = 1 - R$。而比较$d<0$变成了$h< - 1/4$。然而，由于h的初始值是一个整数，并且其相应的增量（Δ_E和Δ_{SE}）也是整数，因此我们可将这个比较改为$h<0$。这样算法就变成了关于h的只进行整数运算的形式。为保持与画线算法的一致性，我们将全部的h替换成d，从而成为一个完整的整数型算法，如图3-16所示。

```
void MidpointCircle (int radius, int value)
/* 假设圆心的坐标是在原点。只进行整数算术运算 */
{
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0)              /* 选择E */
            d += 2 * x + 3;
        else {                  /* 选择SE */
            d += 2 * (x - y) + 5;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    } /* while */
} /* MidpointCircle */
```

图3-16 扫描转换圆的整数型中点算法

图3-17显示了用这个算法产生的半径为17的圆的第2个八分圆弧，以及根据对称性产生的第1个八分圆弧（请与图3-12进行比较）。

二阶差分

我们可以对增量计算方法进行进一步的拓展，由此改善画圆的中点算法的效率。我们知道Δ函数是线性方程，可以进行直接计算。然而任一个多项式都可以按增量的方式进行计算，如同我们处理线和圆的判定变量一样。事实上，我们是在计算一阶和二阶的*偏差分*，在第11章和第19章我们还将使用这一有用技术。其根本思想就是计算函数在其两个邻近点上的值及这两个值的差分（对多项式而言，常常是一个低次多项式的值），并且在程序的每一次迭代中运用这个差分值。

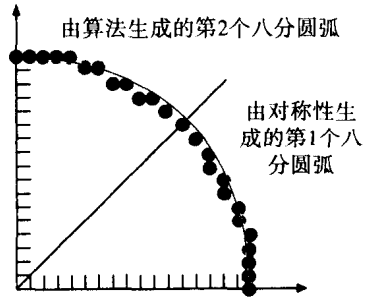


图3-17 运用中点算法生成的第2个八分圆弧，及由对称性生成的第1个八分圆弧

在当前迭代中，如果我们选择了E，则估值点从(x_p, y_p)变化到$(x_p + 1, y_p)$。显然，在点(x_p, y_p)处的一阶差分$\Delta_{E\,old} = 2x_p + 3$。因此，在点$(x_p + 1, y_p)$的$\Delta_{E\,new} = 2(x_p + 1) + 3$，且其二阶差分是$\Delta_{E\,new} - \Delta_{E\,old} = 2$。

类似地，在点(x_p, y_p)处的$\Delta_{SE\,old} = 2x_p - 2y_p + 5$，在点$(x_p + 1, y_p)$处，$\Delta_{SE\,new} = 2(x_p + 1) - 2y_p + 5$，因此二阶差分是$\Delta_{SE\,new} - \Delta_{SE\,old} = 2$。

如果在当前的循环中我们选择了SE，则估值点从(x_p, y_p)变化到$(x_p + 1, y_p - 1)$。因此在点$(x_p + 1, y_p - 1)$处$\Delta_{E\,new} = 2(x_p + 1) + 3$，而二阶差分是$\Delta_{E\,new} - \Delta_{E\,old} = 2$。同样，在点$(x_p + 1, y_p - 1)$处$\Delta_{SE\,new} = 2(x_p + 1) - 2(y_p - 1) + 5$，而相应的二阶差分是$\Delta_{SE\,new} - \Delta_{SE\,old} = 4$。

于是，修改后的算法包括以下几个步骤：(1)根据上一次循环所计算的判定变量d的值的符号选择一个像素；(2)运用上一次循环所计算的相关的Δ，用Δ_E或Δ_{SE}值对判定变量d进行修改；(3)根据像素的移动情况，用前面计算的差分常量修改Δ；(4) 移动像素。Δ_E和Δ_{SE}使用起始像素$(0, R)$进行初始化。这样修改后的程序如图3-18所示。

```
void MidpointCircle (int radius, int value)
/* 这个程序运用二阶偏差分来计算增量 */
/* 假设圆心在坐标原点 */
{
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    int deltaE = 3;
    int deltaSE = -2 * radius + 5;
    CirclePoints (x, y, value);

    while (y > x) {
        if (d < 0) {                /* 选择E */
            d += deltaE;
            deltaE += 2;
            deltaSE += 2;
        } else {
            d += deltaSE;           /* 选择SE */
            deltaE += 2;
            deltaSE += 4;
            y--;
        }
        x++;
        CirclePoints (x, y, value);
    }  /* while */
}  /* MidpointCircle */
```

图3-18 运用二阶差分的中点圆扫描转换算法

83 ~ 87

3.4 椭圆的扫描转换

图3-19所示的是一个圆心在坐标原点的标准椭圆。它的方程如下：

$$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

这里，$2a$是沿x轴的椭圆主轴的长度，而$2b$是沿y轴的椭圆副轴的长度。针对线和圆所讨论的中点技术也可以用来处理一般的圆锥曲线。在这一节我们将讨论SRGP支持的标准椭圆，而在第19章，我们还将讨论各种倾斜的椭圆。为了简化算法，我们在此只画椭圆的第1个四分椭圆，而其他3个四分椭圆可以根据对称性生成。对于圆心不在坐标原点而在一个整数光栅格点的标准椭圆，通过

一个简单的平移处理就能很容易地画出。这里所讨论的算法基于Da Silva的算法[DASI89]，该算法运用二阶偏差分并结合了Pitteway[PITT67]、Van Aken[VANA84]、KAppel[KAPP85]等所用的一些技术。

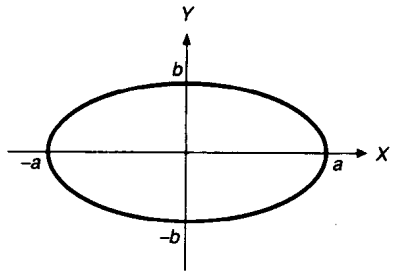


图3-19 圆心在坐标原点的标准椭圆

首先，我们将这个四分椭圆分成两个区域。这两个区域的分界点就是曲线上斜率为 - 1的那个点（见图3-20）。

显然，相比于在圆中的操作，在椭圆中求这个点比较复杂。与曲线上的点P处的切线垂直的向量被称为*梯度*，其定义为：

$$\text{grad } F(x, y) = \partial F/\partial x\,\mathbf{i} + \partial F/\partial y\,\mathbf{j} = 2b^2x\,\mathbf{i} + 2a^2y\,\mathbf{j}$$

两个区域的边界点是该点上曲线上斜率为 - 1的点，并且该点出现在梯度向量的斜率为1时，也就是说，该梯度向量的**i**和**j**两个分量具有相同大小的数值时。显然，在区域1中梯度向量的**j**分量大于**i**分量，而在区域2中则是**i**分量大于**j**分量。因此，在扫描转换过程中，如果在下一个中点$a^2(y_p - 1/2) \leqslant b^2(x_p + 1)$，我们就从区域1切换到区域 2。

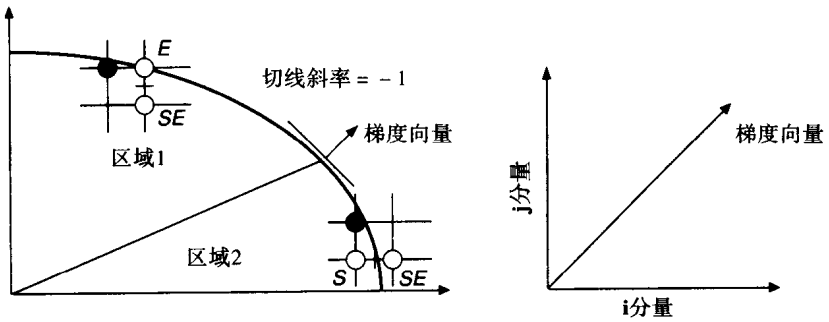


图3-20 由45°的切线定义的椭圆的两个区域

对于任何中点算法，我们都是在两个像素的中点处计算函数的值，并根据该值的符号判定该中点是在椭圆内还是在椭圆外，再由此挑选更靠近椭圆边的像素。因此，在区域1中，如果当前的像素是在(x_p, y_p)处，那么在区域1中的判定变量d_1的值就是$F(x, y)$在E与SE之间的中点（$x_p + 1, y_p - 1/2$）处的值。现在，我们重复为圆求取两个Δ的过程。若选择像素E，则下一个中点是当前中点在x方向增长一步。那么，

$$d_{\text{old}} = F(x_P + 1, y_P - \tfrac{1}{2}) = b^2(x_P + 1)^2 + a^2(y_P - \tfrac{1}{2})^2 - a^2b^2$$

$$d_{\text{new}} = F(x_P + 2, y_P - \tfrac{1}{2}) = b^2(x_P + 2)^2 + a^2(y_P - \tfrac{1}{2})^2 - a^2b^2$$

因为$d_{\text{new}} = d_{\text{old}} + b^2(2x_p + 3)$，所以增量$\Delta_E = b^2(2x_p + 3)$。

若选择像素SE，则下一个中点是当前中点在x方向增长一步，而在y方向下降一步。于是，

$$d_{\text{new}} = F(x_P + 2, y_P - \tfrac{3}{2}) = b^2(x_P + 2)^2 + a^2(y_P - \tfrac{3}{2})^2 - a^2b^2$$

因为$d_{\text{new}} = d_{\text{old}} + b^2(2x_p + 3) + a^2(-2y_p + 2)$，所以增量$\Delta_{SE} = b^2(2x_p + 3) + a^2(-2y_p + 2)$。

在区域2中，若当前像素是在(x_p, y_p)处，那么判定变量d_2的值就是在E与SE之间的中点上的$F(x_p + 1/2, y_p - 1)$。在区域1中所做的运算，在区域2中也进行类似的处理。

当然，我们也必须计算初始条件。假设a和b均为整数，椭圆从$(0, b)$开始画，而第1个要计算的中点是$(1, b - 1/2)$，那么

$$F(1, b - \tfrac{1}{2}) = b^2 + a^2(b - \tfrac{1}{2})^2 - a^2b^2 = b^2 + a^2(-b + \tfrac{1}{4})$$

在区域1中的每一次迭代，我们不仅要测试判定变量d_1和修改Δ函数，而且还要计算E和SE之间的中点处的梯度以判断是否转到了区域2。当中点进入区域2时，我们就要改变像素的选择方式，即由在E和SE之间选择变为在SE和S之间选择，同时还要根据SE和S之间的中点，对区域2的判定变量d_2进行初始化。也就是说，如果在区域1中选择的最后一个像素位于(x_p, y_p)，那么就在点$(x_p + 1/2, y_p - 1)$处初始化d_2。当在区域2中选择的像素的y值等于0时，就停止画像素操作。

88 ~ 89

与画圆算法类似，在循环的每一次迭代中，我们可以直接计算Δ函数，也可以用差分计算它们。Da Silva指出，为Δ所进行的二阶差分计算，实际上也可以用来计算梯度[DASI89]。他还讨论了如何处理旋转过的一般椭圆，以及具有棘手边界条件的很扁的椭圆。在图3-21所示的伪代码算法中，评价函数采用较简单的直接计算方法，而没有采用高效的二阶差分方法。其程序亦省略了许多测试操作（参见习题3.9）。当a和b都是整数时，通过程序变换，我们可以避免小数的出现，而只运用整数算术运算。

```
void MidpointEllipse (int a, int b, int value)
/* 假设椭圆的圆心在坐标原点。注意，在平方操作时， */
/* 16位的整数可能溢出。 */
{
    double d2;

    int x = 0;
    int y = b;
    double d1 = b^2 - (a^2 b) + (0.25 a^2);
    EllipsePoints (x, y, value);              /* 根据4相对称性写像素 */

    /* 检测梯度，判断是否还在第一区域内 */
    while ( a^2 (y - 0.5) > b^2 (x + 1) ) {    /* 区域1 */
        if (d1 < 0)                           /* 选择E */
            d1 += b^2 (2x + 3);
        else {                                /* 选择SE */
            d1 += b^2 (2x + 3) + a^2 (-2y + 2);
            y--;
        }
        x++;
        EllipsePoints (x, y, value);
    }  /* Region 1 */

    d2 = b^2 (x + 0.5)^2 + a^2 (y - 1)^2 - a^2 b^2;
    while (y > 0) {                           /* 区域2 */
        if (d2 < 0) {                         /* 选择SE */
            d2 += b^2 (2x + 2) + a^2 (-2y + 3);
            x++;
        } else
            d2 += a^2 (-2y + 3);              /* 选择S */
        y--;
        EllipsePoints (x, y, value);
    }  /* Region 2 */
}  /* MidpointEllipse */
```

90

图3-21 中点椭圆扫描转换算法的伪代码。

我们已经讨论了如何扫描转换单像素宽的线和不需填充的图元，下面我们将注意力转到如何使用这些算法的修改版以一种颜色或图案来填充定义区域的图元，或者运用线宽和笔型属性

的组合来绘制非填充的图元。

3.5 填充矩形

填充图元的工作可以分成两部分：判定哪些像素要被填充（这取决于连续裁剪后图元的形状），决定用什么值来填充这些像素。我们首先讨论只用一种颜色来填充未被裁剪的图元，而在3.8节中，我们将讨论图案填充。一般而言，决定填充哪些像素包括下面的操作：用连续的扫描线对图元求交，并对由相邻的像素构成的位于图元内的跨段进行从左至右的填充。

在用一种颜色填充矩形时，对每一条扫描线上从矩形的左边到右边的所有像素，我们都给它们置成同样的值，也就是说，填充从x_{min}到x_{max}的每个跨段。跨段反映了图元的*空间相关性*，即对于一个跨段内的所有像素，或者从一条扫描线到另一条扫描线，图元的特征常常是不变的。为尽可能地利用相关性，我们一般只找那些会引起相关性变化的像素。对于只用一种颜色的图元，一个跨段内的像素都置成同样的值，这就有了*跨段相关性*。而只用一种颜色的矩形，则具有很强的*扫描线相关性*，即与矩形相交的那些连续的扫描线都是完全等同的。以后我们还会为一般的多边形运用*边相关性*。我们利用各种相关性，不只是为了扫描转换二维图元，还可以用来绘制三维图元，这将在15.2节中讨论。

能够对一个跨段内的多个像素进行相同的处理是非常重要的，因为这样就可以往帧缓存中一次写一个字，以尽可能地减少费时的访问存储器的操作。对于一个二值显示器来说，我们每次写16个或32个像素。当然，如果跨段不是按字长对齐的，算法必须屏蔽掉字中那些不对应像素的位。在此我们需要特别考虑写存储器的效率，这如同实现copyPixel命令一样，我们将在3.16节中进行简要的讨论。不过在我们的代码中，我们将主要讨论如何确定跨段，而不讨论写进存储器时的效率问题，参见第4章和第19章以及习题3.13。

为此，矩形的扫描转换就只是一个简单的嵌套**for**循环：

```
for (y from ymin to ymax of the rectangle)      /* 扫描线 */
    for (x from xmin to xmax )                  /* 跨段内的像素 */
        WritePixel (x, y, value);
```

在这种简单直接的处理中会出现一个有趣的问题，此问题类似于扫描转换一个由具有共享像素的折线段构成的线集。当两个矩形有一条共享边时，如果我们依次扫描转换每个矩形，那么，共享边上的像素将被写两次。正如前面所讨论的，这不是所希望的。该问题反映了区域定义图元的一个更广泛的问题，即需要确定哪些像素是属于图元的而哪些像素不是。显然，对于这些图元而言，在数学上位于图元内部的像素将肯定属于该图元。但是，在图元边界上的那些像素呢？如果我们只考虑一个矩形（或只是从数学上考虑此问题），边界上的像素将包含在图元内。但因为我们要 91
避免对共享边扫描转换两次，我们就必须确定一些规则，以便惟一地确定边界元素。

一个简单的规则就是：由一条边确定的包含图元的半平面如果位于该边的左方或下方，那么，这条边上的边界像素（即边上的像素）就不属于该图元。因此，左端的边和底端的边上的像素将会画出，但右端的边和上端的边上的像素将不画出。所以，共享的垂直边将“属于”有共享边的两个矩形中靠右的那个。这样，矩形中的一个跨段就是在左边封闭而在右边开放的一个区间。

对此规则有一些情况需要说明。第一，它可用于任意形状的多边形，而不只是矩形。第二，矩形的左下顶点还是会被画两次（对这种特殊的情况，我们还需其他规则来处理，这将在下一节中讨论）。第三，这个规则也可以用于非填充的矩形和多边形。第四，这个规则会导致每个跨段遗失其最右边的像素，以及每个矩形失去其最上端的一行。这些问题表明，对于避免在共

享边（或潜在的共享边）上的像素画两次的问题，没有“完美”的解决方法。但相比于像素不显示或在**xor**模式下将像素置成不想要的色彩，人们一般还是认为省去右边和顶边上的像素要好些（视觉上少一些杂乱）。

3.6 填充多边形

下面要讨论的多边形扫描转换算法处理的多边形包括凸的和凹的，甚至是自相交或有内部空洞的。它一般通过计算位于多边形的左端边和右端边之间的跨段来实现。跨段的端点用增量算法计算，此算法从前一条扫描线与边的交点计算下一条扫描线与边的交点。图3-22中说明了多边形扫描转换算法的基本过程，其中包括一个多边形和一条穿过它的扫描线。扫描线8与边*FA*和*CD*的交点是在整数坐标上，而与边*EF*和*DE*的交点就不是。图中的交点，从 *a* 到 *d*，都用小竖线标志。

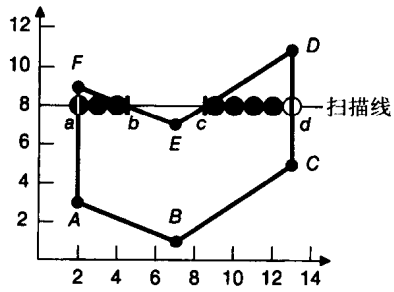


图3-22 多边形和扫描线8。

92 我们必须决定每一条扫描线上的哪些像素在多边形中，并给这些像素置成适当的颜色（此图中的两个跨段分别是从*x*等于2到4和从*x*等于9到13）。为每一条与多边形相交的扫描线重复上述过程，就完成了对整个多边形的扫描转换，图3-23给出了使用这一操作的另一个多边形。

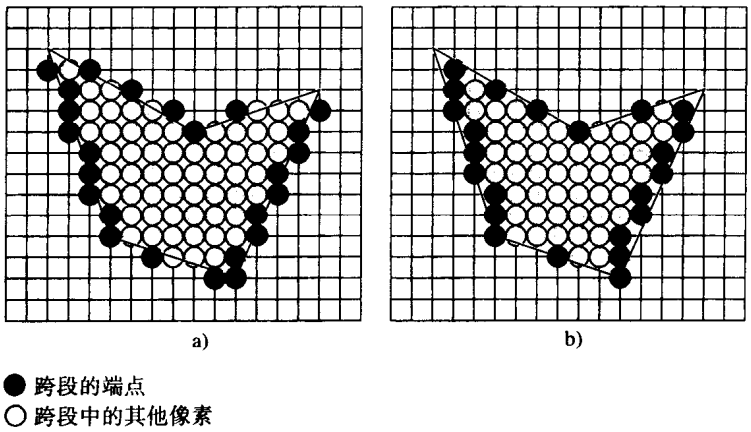


图3-23 一个多边形中的跨段。跨段的端点用黑色表示，跨段内的像素用灰色表示。a) 端点是用中点算法计算的，b) 端点位于多边形内部

在图3-23a中，每个跨段的端点像素用黑色表示，而其间的像素用灰色表示。一种直接计算跨段端点的方法是对每条边运用中点线扫描转换算法，并用一个表记录每条扫描线上的各跨段的端点。当为一条边产生的像素超越了已知的跨段时，就修改记录表。但是，这样会产生一些位于多边形外的端点像素，因为根据扫描转换算法，所选择的像素是最靠近边的，而没有考虑它们是否在多边形的内部——画线的算法没有内部与外部的概念。显然，对于有共享边的多边形，我们并不想画这些共享边以外的像素，因为它们侵入了相邻多边形的区域，当这些多边形是用不同的颜色填充时，这会使图形很难看。所以，最好是只画那些严格地在多边形内部的像素，即便一个外部像素可能更靠近多边形的边。为此，我们必须适当调整多边形扫描转换算

法。比较图3-23a和3-23b我们会发现，在图3-23b中有许多在理想图元外的像素没有画。

使用这种技术，一个多边形就不会侵入别的图元所定义的区域（就是一个像素也不会）。我们可以运用相同的方法来处理非填充的多边形，以保证其一致性。甚至于当多边形没有相同的边界像素时，不管它们是填充的或非填充的，我们都可以用这样的技术在一条线段上同时扫描转换多个矩形和多边形。

同初始的中点算法一样，我们运用增量算法来计算一条扫描线上的跨段的端点，即根据前一条扫描线上的端点进行递推运算，而不是解析地计算扫描线与多边形的每条边的交点。例如，在图3-22中的扫描线8上，有两个在多边形内部的像素跨段。运用下面的3步操作，就能将这些跨段都填上：

93

1) 找到扫描线与多边形所有边相交的交点。

2) 根据x坐标的增序对这些交点进行排序。

3) 运用奇偶规则，填充每对交点间在多边形内部的像素。决定一个点是否在区域内的奇偶规则是：开始时，奇偶性初始化为偶，每当遇到一个交点就变换一次奇偶性的奇偶性；当奇偶性为奇时，就画像素，而当奇偶性为偶时，就不画像素。

在第1步和第2步是找到交点并对它们排序，这将在下一节讨论。现在，我们讨论如何填充跨段。在图3-22中，已排序的x坐标是（2, 4.5, 8.5, 13）。第3步需要考虑以下4个方面：

3.1) 如果交点是一个任意的有小数的x值，如何判定交点的哪一边的像素是在多边形内部？

3.2) 如果交点是在整数像素坐标上，如何处理这种特殊情况？

3.3) 若3.2中的交点是共享顶点，如何进行处理？

3.4) 在3.2的情况中，若顶点定义了一条水平边，该如何处理？

为了处理3.1的情况，如果以往右的方式靠近有小数坐标的交点，并且我们在内部，那么，我们对交点的x坐标进行往下取整的操作，以得到相关的在多边形内部的像素；否则，如果我们在外部，我们就往上取整，以保证所确定的像素在多边形内部。对于3.2的情况，我们运用的准则就是我们用来避免在矩形的共享边处出现冲突的准则：如果一个跨段最左的像素有一个整数的x坐标，我们就认为它在内部；如果其最右的像素有一个整数的x坐标，我们就认为它在外部。对于3.3的情况，在进行奇偶性的计算时，我们只对一条边上y_{min}顶点计数，而不对其y_{max}顶点计数；这样，一个y_{max}顶点，只有当它恰是相邻边的y_{min}顶点时，才有可能画上。例如，图3-22中的顶点A，在奇偶性的计算中，它只计数一次，因为它是边FA的y_{min}顶点，当然它也是边AB的y_{max}顶点。这样，边和跨段就都被当成了在最小值处关闭而在最大值处开放的区间。当然，相反的规则也是可行的。但这里使用的规则似乎更自然一些，因为它将最小值的端点作为进入点，而将最大值的点作为离开点。处理3.4的情况，也就是水平边的情况时，可以希望得到的结果是，像矩形一样，底部边被画上而顶部边则不画。正如我们在下一节将讨论的那样，如果不对这种（水平）边的顶点记数，上述结果将能自动实现，因为这些顶点既不是y_{min}也不是y_{max}顶点。

在图3-22中，扫描线8不与多边形的任何顶点相交。我们将上述的规则运用于该扫描线时，将填充下面这些像素：从a点（即像素(2, 8)）到b点左边的第一个像素（即像素(4, 8)）之间的所有像素，以及c点右边的第一个像素（即像素(9, 8)）到d点左边的第一个像素（即像素(12, 8)）之间的所有像素。对于扫描线3，因为顶点A既是边FA的y_{min}顶点，也是边AB的y_{max}顶点，因此，它被记数一次。这使得奇偶性变为奇，因此我们就从这里开始填充一个跨段的像素，一直到该扫描线与边CB的交点的左边的第一个像素，然后，奇偶性又变为偶。至于扫描线1，它只与顶点B相交，并且该点是边AB和BC的y_{min}顶点，因此，B将被记数2次，而奇偶性还保持为偶。这

94 个点就相当于一个没有长度的跨段——在该点进入，画该点，然后又在该点离开。虽然在这样的局部最低点会画一个像素，但在局部最高点不会画任何像素。比如扫描线9与顶点F的交点。在此，顶点F是边FA和EF的y_{max}顶点，不会引起奇偶性的变化，奇偶性保持为偶。

3.6.1 水平边

考察图3-24中有关水平边的各种情况，我们不必对水平边的顶点进行有关奇偶性的记数就能够恰当地处理水平边的情况。在底端边AB，顶点A是边JA的y_{min}顶点并会引起奇偶性变为奇，而边AB上的顶点不影响奇偶性的变化，因此，奇偶性为奇而跨段AB会画出。由于垂直边BC的y_{min}顶点是B，它会使得奇偶性变为偶，但边AB此时也不会影响奇偶性的变化，因此，该跨段到此结束。在点J，它是边IJ的y_{min}顶点，而不是边JA的y_{min}顶点，因此，奇偶性变为奇，并画一个跨段到边BC。由于C是边BC的y_{max}顶点，不会引起奇偶性的变化，因此，该跨段沿着边CD继续前进，一直到D点为止。因为D是边DE的y_{min}顶点，使得奇偶性变为偶，并终止跨段。在点I处，由于I是边IJ的y_{max}顶点，而HI是水平边，所以，奇偶性保持为偶，而顶端边IH不会画出。但是在点H，由于它是边GH的y_{min}顶点，奇偶性变为奇，并从H开始画一个跨段，一直到该扫描线与边EF的交点的左边的第一个像素为止。最后，由于点G和F都不是y_{min}顶点，因此顶部边FG不会被画上。

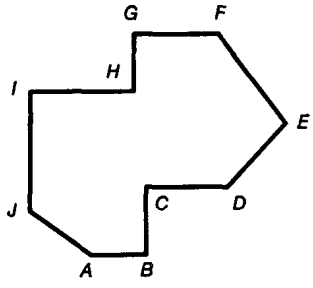


图3-24 一个多边形中的水平边

上面讨论的算法可以处理多边形中的共享顶点、两个相邻多边形的共享边以及水平边。它也可以处理自交的多边形。但是，我们注意到，该算法会遗失一些像素。更糟糕的是，如果不保持填充过程的记录，它会对一些共享像素画许多次，比如由两个以上的多边形共享的边，或者两个不相邻的三角形所共享的y_{min}顶点（见习题3.14）。

3.6.2 狭长条

在我们的扫描转换算法中还有一个问题，它不能像水平边那样得到令人满意的处理，即多边形的边非常靠近，产生了狭长条——其多边形区域太
95 过细窄以致于容纳不下扫描线上可见的最短跨段。例如，图3-25中的三角形，该三角形的三个顶点是(0, 0)，(3, 12)和(5, 12)。如果根据规则，只画出在多边形内部以及多边形的左端边和底端边上的像素，许多扫描线上只有一个像素或根本就没有像素。这种“遗失”像素的问题，是走样问题的一个例子。以离散的方式对连续的信号进行逼近表达时，就会产生走样问题。如果像素是多位表示的，我们就可以应用反走样的技术，如应用在3.17节将介绍的关于线走样的处理方法，以及在19.3节将介绍的关于多边形走样的处理方法。反走样将降低规则“只填充在多边形内部以及多边形的左端边和底端边上的像素”的作用，使边界像素甚至一些外部像素也能以某个亮度值进行填充。当然，这个亮度值是根据像素中心到图元的距离决定的。如此，在一个像素上可以汇聚多个图元的作用。

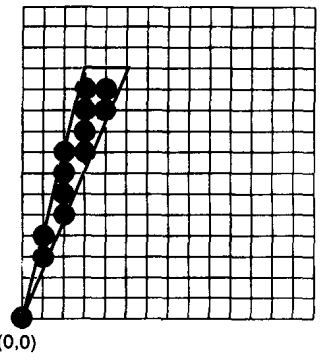


图3-25 扫描转换多边形的狭长条

3.6.3 边相关性和扫描线算法

在算法的第一步，是计算扫描线与边的交点。这必须采用巧妙的方法，否则其算法运行速度将会很慢。特别是，我们必须避免采用直接了当的测试算法，即测试与每一条新扫描线相交

的多边形的边。通常，对于给定的一条扫描线常常只有少数几条边是有价值的。而且我们注意到，与扫描线i相交的许多边，也常常与扫描线$i+1$相交。对一条边而言，它与多少条扫描线相交，这种边相关性就会在这多条扫描线上存在。类似于中点线扫描转换算法，可以根据当前的点递推出下一个点，当从一条扫描线移到下一条扫描线时，我们可以根据上一条扫描线与边的交点的x坐标，很快地计算出新扫描线与该边交点的x坐标。其递推公式是：

$$x_{i+1} = x_i + 1/m$$

在此，m是边的斜率。在中点线扫描转换算法中，我们通过计算一个整型的判定变量避免了小数的算术运算，并且只需根据判定变量的符号就能选择最靠近实际线的像素。在此，我们也将用整型的算术运算进行相关的取整操作来计算最靠近边的在多边形内的像素。

现在，考察斜率大于+1的在左边的线。对于在右边或有其他斜率的线，可以用类似的参数进行处理，当然，对它们要做一些技巧性的处理。在此，注意垂直边是一种要特殊处理的情况（我们已知，水平边可以由跨段进行隐含的处理）。在线的端点(x_{min}, y_{min})，我们需要画一个像素。然后，当y按单位长度递增时，线上相关点的x坐标随之递增，其步长是$1/m$，在此，$m=(y_{max}-y_{min})/(x_{max}-x_{min})$是线的斜率。这样的增长使得$x$除了有一个整数部分外，还有一个小数部分，这可以表示为以$(y_{max}-y_{min})$为分母的一个分数。当我们重复这一过程时，小数部分将会溢出，而整数部分会增加。例如，如果斜率是5/2，并且x_{min}是3，那么x值的变化序列是 $3, 3\frac{2}{5}, 3\frac{4}{5}, 3\frac{6}{5}=4\frac{1}{5}$ 等等。当x的小数部分是零时，我们就画上位于线上的像素（x, y）。当x的小数部分不是零时，我们需要进行取整操作来求得一个严格地在多边形内的一个像素。当x的小数部分变得大于1时，我们将x的整数部分加1，并将它的小数部分减1，同时将像素的位置往右移一位。如果增量运算到某处恰好落在一个像素上，就画出这个像素，并从x的小数部分中减去1，以保证小数部分小于1。

96

显然，当分子大于分母时，小数部分就大于1。因此，只需记录分数中分子的变化，就能避免使用小数。在图3-26所示的算法中，我们实现了该技术。我们用一个变量*increment*来记录分子连续加时的变化。一旦分子大于分母，就从分子中减去分母，并将x坐标增加一个单位。

```
void LeftEdgeScan (int xmin, int ymin, int xmax, int ymax, int value)
{
    int y;

    int x = xmin;
    int numerator = xmax - xmin;
    int denominator = ymax - ymin;
    int increment = denominator;

    for (y = ymin; y <= ymax; y++) {
        WritePixel (x, y, value);
        increment += numerator;
        if (increment > denominator) {
            /* 溢出，则向上取整以移到下一个像素，然后从分子中减去分母的值 */
            x++;
            increment -= denominator;
        }
    }
} /* LeftEdgeScan */
```

图3-26 扫描转换多边形的一条左边的边

现在我们可设计*扫描线算法*，利用边相关性，对每条扫描线跟踪记录它所相交的边及其交点，存于一个称为*活动边表*（AET）的数据结构中。AET中的边是根据它们与扫描线交点的x坐标值进行排序的。这样，我们就能根据这些交点，按顺序依次两两成对地形成跨段进行填充，这些交点（经过取整处理）就是各个跨段的端点。当我们移到下一条扫描线$y+1$时，对AET要进行修改。首先，目前在AET中但不会与下一条扫描线相交的边（比如那些$y_{max}=y$的边）会从AET中去掉。然后，与下一条扫描线相交的新边（比如那些$y_{min}=y+1$的边）将被加入到AET中。最后，对那些在AET中并且还将与下一条扫描线相交的边，我们将运用前面介绍的增量边算法计算新交点的x坐标。

97

为了高效地将边加入到AET中，我们在初始化时建立一个全局的*边表*(ET)，它包含多边形的所有边，并且这些边按照它们各自y坐标较小值排序。一般地，ET是基于桶排序的方式建立的，有多少条扫描线就有多少个桶。在每个桶中，根据边的较低的端点的x坐标，按照增序的方式排列各条边。ET中的每一项包含下列信息：边的y_{max}坐标值，边的下端端点的x坐标(x_{min})，随着扫描线递变到下一条线时的x坐标的增量$1/m$。图3-27中显示了图3-22中的多边形的6条边是如何排序的，而图3-28则显示了该多边形在扫描线9和10处的活动边表。（实际实现时，我们很可能会加一个标志，以区分是左端边还是右端边。）

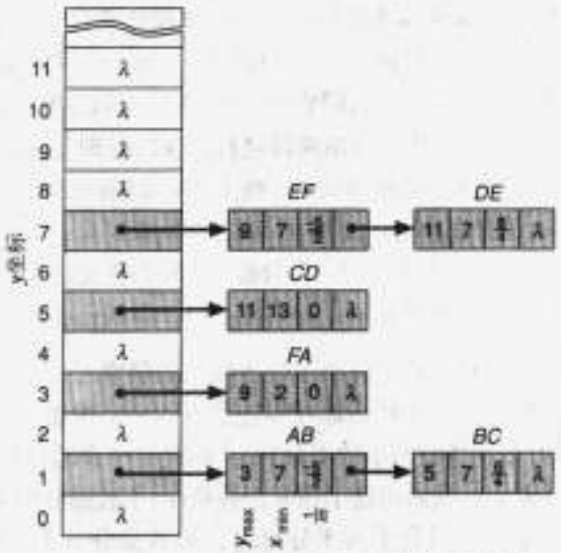


图3-27 为图3-22中的多边形建立的桶排序的边表

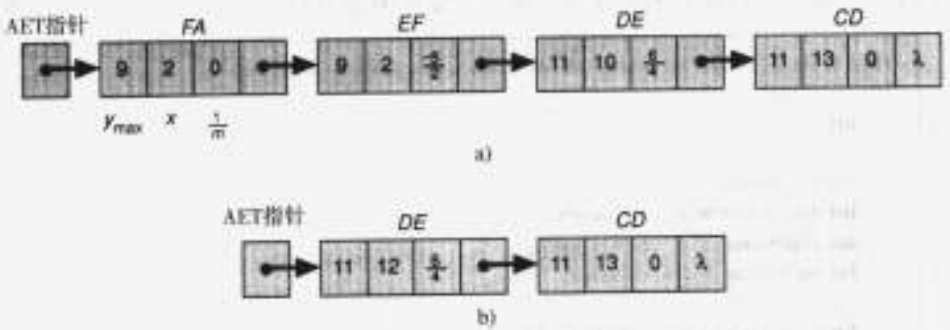


图3-28 为图3-22中的多边形建立的活动边表。a) 扫描线9，b) 扫描线10。（注意：b)中边DE的x坐标是向上取整的，因为它是左端边。）

一旦生成了ET，扫描线算法就按下列步骤进行：

1) 将y置为边表ET中最小的y坐标值，即第一个非空的桶的y值。

2) 将AET初始化为空。

3) 重复以下运算，直至AET和ET都为空：

3.1) 从ET的y桶中选择那些$y_{min}=y$的边加入到AET中（进入边）。

3.2) 除去AET中那些$y=y_{max}$的项（即与下一条扫描线不再相交的边），然后在AET中

根据x坐标值进行排序（这很容易实现，因为ET是已排好序的）。

3.3) 在扫描线y上，根据AET中的x坐标，两两成对地构成跨段，并对其中的像素以所希望的颜色进行填充。

3.4) y坐标增加1（即移到下一条扫描线的位置）。

3.5) 对于每一条留在AET中的非垂直的边，根据新的y值修改x坐标。 98

这个算法运用了边相关性来计算交点的x坐标，还利用了扫描线相关性（以及排序）来计算跨段。因为只对少量的边排序，并且在3.2步的重新排序操作是在一个差不多排好序或已排好序的链上进行，所以，无论是用插入排序的方法，还是用复杂度为$O(N)$的冒泡排序方法，在此都是可以接受的。在第15章和第16章中，我们将看到如何拓展这个算法用于处理多个多边形的可见性问题，包括处理透明多边形的情况；以及在第17章，如何在同一个像素点上合成多个多边形的颜色效果。

在扫描转换图元的操作中，三角形和梯形可以作为特殊的多边形来处理，因为它们在每一条扫描线上只有两条边（水平边不进行显式的扫描转换）。实际上，因为任意一个多边形都可以剖分成有共享顶点和共享边的三角形网格（见习题3.17），因此对任意形状的多边形的扫描转换，我们都可以先将其多边形剖分成三角形网格，然后再扫描转换这些三角形。在计算几何中，这种三角剖分的处理是一个经典的问题[PREP85]，并且很容易应用于凸多边形；然而对于非凸的多边形，要进行有效的三角剖分是比较困难的。

注意跨段的计算是累积进行的。也就是说，在扫描转换算法的当前循环中运行到第3.5步，也就是在生成了同一条扫描线上的多个像素后，跨段的端点要进行适当的修改。（对于交叉边和狭长条的情况，在进行跨段的计算时要进行一些特殊的处理。）我们可以一次性地计算出所有的跨段，然后再对这些跨段进行填充，也可以边生成一个跨段就边填充一个跨段，直至处理完所有跨段。运用跨段的另一个好处是裁剪操作可以在计算跨段时同时进行：每个跨段可以独自地相对裁剪矩形的左边或右边的坐标进行裁剪。注意，在15.10.3节中，我们将用一个稍微有点不同的跨段计算方法，用“光线跟踪”绘制多个三维实体。

3.7 填充椭圆弧区域

为每条扫描线计算跨段的方法也同样可用来处理圆和椭圆。在该算法的每一次循环中，我们都是对跨段的端点进行累积的变化，且对每个端点坐标进行取整处理，以保证像素都在区域内。如同扫描转换非填充图元那样，我们可以根据对称性只扫描转换一段弧。在此，注意区域的变化， 99
特别是处理椭圆时。在算法的每一次迭代中，要么是在同一条扫描线上产生一个新的像素，要么是在下一条扫描线上产生一个新的像素。若是前者，就调整跨段的端点，若是后者，就开始形成下一个跨段（见图3-29）。判断一个像素P是否在区域内，我们只需简单地检测一下判定函数$F(P)$的符号，如果是正的，表明该像素是在区域内，并继续考察下一个像素。显然，我们并不想直接计算这个函数，而只要计算此函数在中点的值$F(M)$；这个值可以根据判定变量的值推导出来。

我们知道，对椭圆而言，扫描线只与其边界交叉两次，因此，我们不必使用边表这样的结构，而只需保留一个当前的跨段。正如在处理多边形时所做的，我们可以先生成各条扫描线与图元相交的所有跨段，然后再填充；也可以边生成一个跨度就边填充一个跨段，比如说，y值增加一次，就填充一个跨段。

对于锲形区域的填充，必须特别注意。第一个问题是如何计算由起始角和终止角定义的射线与图元边界的交点，以及如何根据这些交点设置判定变量的起始值和终止值。对于圆，通过将其方程变为三角函数的形式就可以方便地进行这种计算，然后将(Round($R\cos\theta$), Round($R\sin\theta$))

用于中点计算公式。那么，要进行扫描转换的区域的边界由两条射线和它们之间的边界弧段围成。由于角度的不同，一条扫描线在开始和结束时都可能落在射线或弧段上，因此，必须采用经过相应修改了的增量算法，以保证只选择在锲形内部的像素（见习题3.19）。

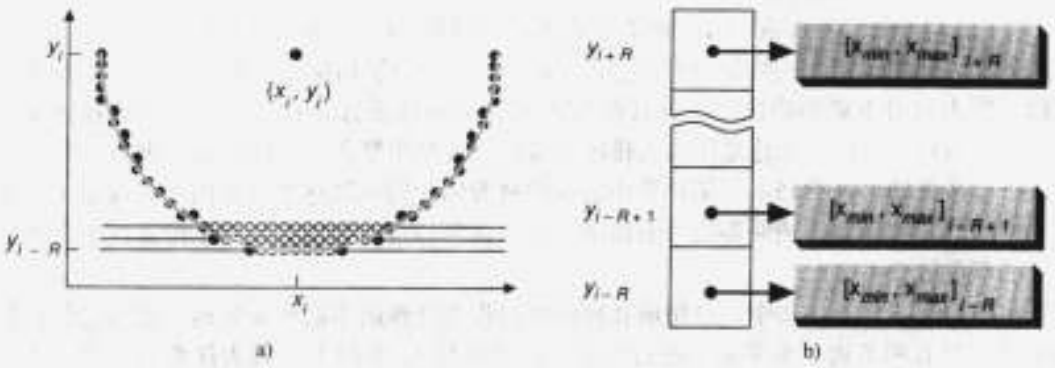


图3-29 运用跨度填充一个圆。a) 3个跨度，b) 跨度表。每个跨度只存储它的端点

3.8 图案填充

在前面几节，我们对SRGP中定义区域的图元进行填充时，只用一种颜色，其实现是将该颜色赋给WritePixel程序中的*value*参数。本节我们将讨论用图案进行填充，其实现方法是在扫描转换算法中最终写像素的时候增加一些控制操作。对于像素图图案，这种控制操作就是从图案中的适当位置选择出颜色，这将在下面讨论。当位图图案以透明方式填充时，我们执行WritePixel，对于图案中是1的位用前景颜色填充像素，而对于是0的位则不用WritePixel。这就像有关线型的处理一样。另一方面，如果位图图案以不透明方式填充，则对于1或0的位，就要分别选择前景颜色或背景颜色。

100

用图案进行填充的最主要的问题是在图案区域和图元区域之间建立联系。换句话说，我们要决定图案“固定”在哪里。这样，我们才能知道图案中的哪个像素相应于图元中的当前像素。

第一种方法是通过放置图案第一行最左的像素来将模式固定在多边形的一个顶点。这样，当图元移动时，图案可以跟着移动。对于具有较强几何结构的图案，比如在绘图应用中常用的交叉阴影线图案，该种处理可以产生理想的视觉效果。但是，一个多边形中并没有一个特别的点，可以很显然地建立这样的关联；至于圆和椭圆这样平滑变化的图元，就根本没有这样的点。因此，程序员必须确定建立这种联系的固定点是在图元的边界上还是在图元中。在有些系统中，一个固定点甚至会应用于一组图元。

第二种方法就是将整个屏幕用图案完全覆盖，而将图元当成是一个带透明性的轮廓或填充区域，以允许图案显示出来。在SRGP中使用了这种方法。这样处理，固定点的标准位置就是屏幕的原点。于是，图元的像素均被当成1，并同图案进行“与”的操作。这种方法的一个副作用是：当图元轻微移动时，图案不会“粘着”图元进行变化。相反，图元的移动就像是在固定的图案背景上进行剪切一样，因此，它的外观会随着其移动而变化。对于不带明显几何倾向性信息的规则图案，用户可能注意不到这种效果。除了计算效率高以外，这种绝对固定点的处理可以保证图元无缝地相互覆盖和邻接。

将图案应用于图元时，我们要根据当前像素的坐标(x, y)对图案进行索引以做对应的处理。因为一个图案是定义成一个$M \cdot N$的较小的位图或像素图，因此，我们需用“取模”的算术运算来循环使用该图案。例如，如果将图案中的像素*pattern*[0, 0]与屏幕的原点重合[⊖]，我们就可以以透明的方式用下列的语句来写一个位图图案：

```
if (pattern[x % M][y % N])
    WritePixel (x, y, value);
```

如果我们以**replace**（替换）的方式填充整个跨段，则当底层有一个copyPixel命令可以用于写多个像素时，我们就可以一次性地复制图案中的一整行。例如，假设图案是一个8×8矩阵，那么对于长度是8个像素的跨段，就可以反复地进行这种操作。如果一个跨段的最左的点是字节对齐的（也就是这个像素的x坐标值对8取模后的值是0），那么，用copyPixel命令操作一个1×8的数组就可以将图案中整个第一行都写出。这种操作可以重复多次，直至填完跨段。如果跨段的两个端点都不是字节对齐的，则那些不在跨段内的像素必须用掩码屏蔽掉。程序员有时花费许多时间以使光栅算法在处理一些特殊情况时特别有效。例如，可以通过提前检测消除一 101
些内循环，或者为内循环写一些汇编语言代码，以发挥某些专门硬件的长处，比如使用已介绍过的指令高速缓存或者一些特别有效的循环指令。这种优化将在第19章中讨论。

不用重复扫描转换的图案填充

至此，我们已经讨论了在扫描转换过程中进行填充的操作。另有一种方法，是先将图元扫描转换到一个矩形工作区，然后从该工作区的位图中将每个像素写到画布中的适当位置。这种写到画布的所谓*矩形写*的操作是一个简单的嵌套式**for**循环，此时，遇到1时，就用当前的色彩写；而遇到0时，就不写（透明时）或用背景颜色来写（不透明时）。与在扫描转换过程中进行填充相比，这种两步方法的工作量要多一倍。因此，对于只需被扫描转换一次的图元，这种方法是不合适的。但是，对于要被反复扫描转换多次的图元，这就比较合适。比如，给定了字体和大小的字符，就是这样的一种情况，可以提前对它们的轮廓进行扫描转换。对于只定义成位图字体的字符，或者别的什么作为位图作画或扫描的图元，如图标和一些应用符号，在任何情况下都不用扫描转换，而惟一要用的就是对位图实施矩形写操作。预先扫描转换位图的优点是，对一个矩形内的所有像素进行写的操作，不必做任何裁剪或有关跨段的算术运算，因此，与每次都从头开始进行扫描转换图元并做裁剪的操作相比，它显然要快得多[⊜]。

既然我们要将一个矩形位图写到画布上，为什么不直接用copyPixel来复制位图，而是要一次只写一个像素？对于二值显示，用当前的颜色1进行写的时候，copyPixel可以很好地工作：对于透明的方式，我们用**or**（或）的写模式；对于不透明的方式，我们用**replace**（替换）的写模式。对于多值显示，我们不能用一个像素对应1位的方式直接写位图，而必须将位图中的每个位转换成一个完整的n位颜色值，然后再写。

一些系统有功能更强的copyPixel命令，它的拷贝操作可接受一个或多个关于“源读”或“目标写”的掩码的影响。如果我们可以将位图定义成一个“目标写”的掩码，将源定义成是（当前的）常量颜色的一个数组，我们就可以将这种功能用于透明的方式（用于SRGP中的字符处理）。那么，只有当位图的写掩码有1时，像素才被写以当前的颜色；此时，位图的写掩码就相当于一个具有任意形状的裁剪区。从某种意义上说，在一个n位像素的系统中以嵌套的**for**循环显式地实现矩形写的操作，就是模仿这种功能很强的“带有写掩码的copyPixel命令”工具。

⊖ 在窗口系统中，图案经常是固定在窗口坐标系的原点。

⊜ 在反走样时，情况要复杂一些，这将在第19章中讨论。

现在考虑另外一种变形。我们希望画一个填充字母，或者别的什么形状，但是在其内部不是填实，而是用一种图案进行填充。例如，我们要用黑白均匀间杂的针点图案来填充一个粗体字符“P”，字形用一个50%灰度的图案，或者用一个具有两种色调的砖-灰泥图案来生成一个房屋的图标。那么，我们怎样才能用不透明的方式来写这样的一个形体而不进行反复的扫描转换呢？这个问题就是，位图中的0对应的区域内部的“洞”应当写以背景颜色，而区域外的洞
102 （比如“P”中的空穴）还要以透明的方式写，以避免影响它后面的图像。换句话说，对于形体内部的0，我们要以背景颜色来标识，而对于其外部的0，包括空穴，就要根据写掩码保护形体外的像素。如果我们进行快速扫描转换，就不会出现位图中不同区域的0代表不同含义这种问题，因为，我们永远不会遇到形体边界外的像素。

我们可用一个四步过程来避免重复的扫描转换，在此，我们用图3-30中所示的山的场景为例。运用图3-30b中图标的轮廓，第一步是生成一个将用作写掩码/裁剪区域的“填实”的位图，即图形内部的像素都设置为1，而其外部的像素都设置为0。这在图3-30c中已表达出来了，此时白色代表背景像素(0)，而黑色代表1。这种扫描转换只做一次。在第二步，一旦需要对形体进行图案化的拷贝时，我们将用背景颜色填实的位图以透明的方式写到画布上。这样，我们就以背景颜色清理出一块具有形体形状的区域，如图3-30d所示，在此，在已经有山的图像中，有一块房屋形状的区域被置成白色的背景颜色。第三步，用copyPixel命令，将一个矩形图案（图3-30e）按照**and**（与）模式作用于填实的位图，以生成其填实的形体的位图的图案化结果。这样，形体形状内部的一些像素就从1变到了0（图3-30f）。这可以看作是用形体的形状在任意大的图案中裁剪出一小块。最后，我们再将这个新的位图以透明的方式写到画布的同一位置。当然，这次是用当前颜色，即前景颜色，如图3-30g所示。这次与第一次写画布的时候一样，形体外部区域中的像素都是0，

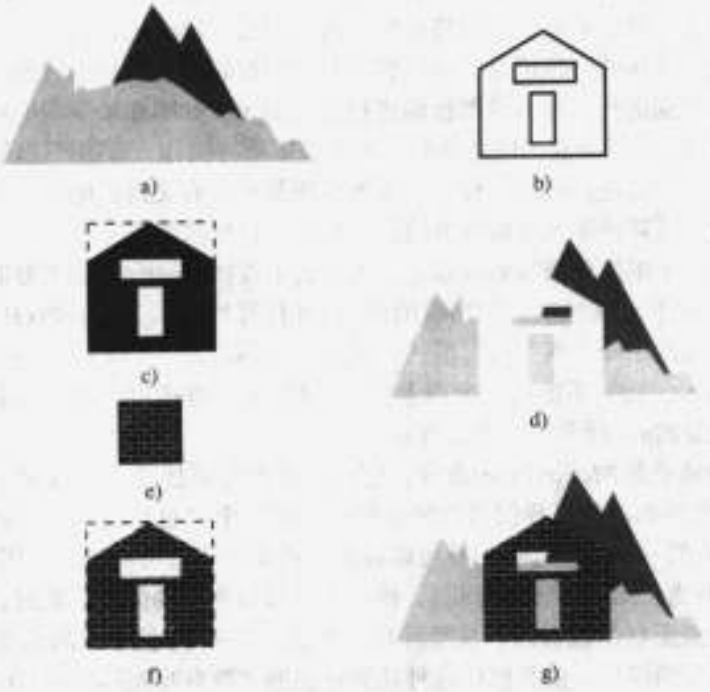


图3-30 运用两个以透明方式写的操作，以不透明的方式写一个图案化的形体。a) 山的场景，b) 房屋图标的轮廓，c) 房屋图标填实状态的位图，d) 通过写“背景”，清理出来的场景，e) 砖的图案，f) 砖的图案应用于房屋图标，g) 将图案化的房屋图标透明地写到屏幕上

以保护区域外的像素。在此，区域内的0不影响以前写的（白色）背景；只有在有1的地方，才写（黑色）前景。如果想用一种带灰泥的红砖图案来对房子进行写的操作，我们可用灰色来写前面提到的填实位图，而用红色来写图案化的位图；在图案位图中，除了表示灰泥的一些狭带上是0外，大部分都是1。就效率而言，在此我们简化了矩形写的过程，该过程原本要在写掩码的作用下用两种颜色做图案填充，而现在则是在透明方式下做两次写操作或者带掩码的copyPixel操作。

3.9 宽图元

从概念上讲，仿照扫描转换单个像素宽的轮廓图元，我们就能生成宽图元。这就是将有某种横截面形状的画笔的中心（或者别的容易辨别的点，比如一个矩形画笔的左上角点）放在扫描转换算法所选择的像素上。单个像素宽的线可以看成是用大小只是单个像素的画笔所画出来的。当然，这样简单的描述掩盖了许多复杂的问题。首先，画笔的形状是怎样的？一般常用的画笔是圆形和矩形的。第二，非圆的画笔的朝向该是怎样的？矩形画笔是否总是直立的，以保持画笔固定的宽度？画笔的朝向是否能随着图元进行变化，使得画笔的垂直轴总是与图元相切？宽线的端点实际上该是什么样的形状？在整数栅格上又该是什么形状？一个宽多边形的顶点会怎样变化？线型和笔型怎样相互作用？在本节，我们将讨论一些比较简单的问题，其余的问题将在第19章中讨论。

画宽图元有4种基本的方法，如图3-31至图3-36所示。对于这些图元实际的情况，我们以白中夹黑的轮廓线表示；对图元进行单个像素宽的扫描转换所产生的像素，是用黑色表示的；为形成宽图元而增加的像素是用灰色表示的。旁边用黑色像素显示了缩小了的宽图元，其图像看上去好像是用颇低的分辨率生成的。第一种方法是粗糙的逼近，即在扫描转换过程中在每一列（或行）应用一个以上的像素。第二种方法是沿着图元单个像素宽的轮廓运动画笔的横截面。第三种方法是生成图元的两个副本，它们的间距是宽度t，然后对这两个副本所形成的内外边界之间的跨段进行填充。第四种方法是用折线逼近所有的图元，然后再用宽线画折线的每条线段。

我们简要地考察一下这些方法，并分析它们各自的优缺点。对于许多应用来说，即使不是绝大部分，至少在屏幕上显示的时候，所有的方法都能得到满意的效果。但是在用于印刷时，应当尽量使用高分辨率来达到好的效果，因为在印刷中，对于算法速度的要求不像实时生成图元那样高。这样，我们可以用一些比较复杂的算法来产生好看的结果。软件包甚至可能对不同的图元使用不同的技术。例如，QuickDraw在画线的时候是移动直立的矩形画笔，而在处理椭圆的时候，则是填充同心的两个椭圆边界之间的跨段。

103 / 104

3.9.1 复制像素

我们可将扫描转换中的内循环扩展一下，使它在每个计算出来的像素处写多个像素以生成宽图元。此法用于画线时效果尚好；在此，对于斜率在 -1和1之间的线，就在每列上复制多个像素，而对于其他斜率的线就在每行上复制多个像素。当然，这样处理，线的端点总是垂直的或水平的，在画相当宽的线的时候，其效果是不大令人满意的，如图3-31所示。

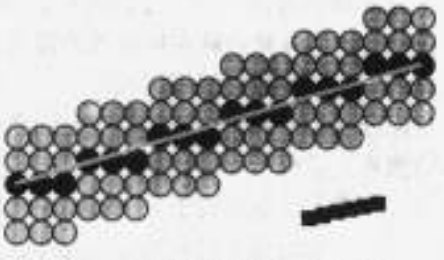


图3-31 通过列复制所画的宽线

复制像素的算法在几个线段以一个角度相接的地方还会产生明显的缝隙，因为当斜率发生变化，需要从水平复制变到垂直复制，此时，这种算法会遗漏一些像素。对于椭圆，当其弧

在45°的地方时，就会生成特别细的边界，如图3-32所示。

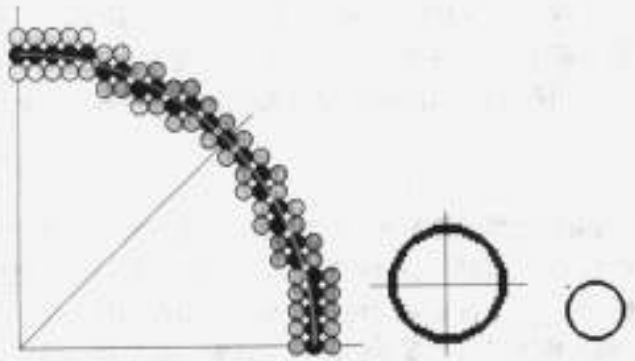


图3-32 通过列复制所画的宽圆

特别是，当图元的宽度定义为图元的内外边界之间垂直于图元切向的间距时，水平线和垂直线的宽度与倾斜的线的宽度是不同的。例如，如果宽度参数是t，水平线或垂直线的宽度是t，而倾斜角度是45°的线的平均宽度是$t/\sqrt{2}$。这又是一种倾斜线的像素比较少的情况，这种情况在3.2.3节中已经提过；与同样宽的水平线和垂直线相比，倾斜线的亮度就低。对于复制像素，还有一个普遍性的问题，即宽度为偶数时出现的情况。此时，我们无法将复制的列或行的中心置于所选择的像素上，这样，图元的一边比另一边就要“多出”一个像素。总之，像素复制的方法是一种有效但是略显粗糙的逼近方法，在画不是很宽的图元的时候，它的效果最好。

3.9.2 移动画笔

选择一个矩形画笔，使它的中心或角点沿着图元的单个像素宽的轮廓运动。对于线的生成，如图3-33所示，这样处理的效果相当好。注意，这条线与用像素复制的方法所生成的线很相似，但在端点处要宽一些。同像素复制的方法一样，因为笔是垂直对齐的，所画的图元的宽度是随着图元的角度变化的，但方式正相反：水平线段的宽度最细，而斜率为±1的线段最宽。例如，椭圆弧的宽度在它的整个轨迹上都是变化的，在其切线几乎是垂直或水平的弧段时，宽度正好是所定义的，而在切线的角度大约是±45°的时候，宽度变宽的因子是$\sqrt{2}$（见图3-34）。如果这个方块随着路径而转动，就可以解决这个问题。当然，最好是采用一个圆形的横截面，这样，宽度就与线倾斜的角度无关。

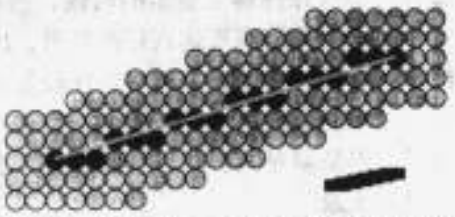


图3-33 通过跟踪一个矩形画笔所画的宽线

现在，对于一些简单的情形，如直立矩形或圆形的横截面，我们讨论如何实现这种移动画笔算法。最简单的方法是用命令copyPixel拷贝所要求的填实的或图案化的横截面（也称为*足迹*），以确保它的中心或角点就在所选择的像素上；对于一个圆形足迹或用不透明方式所画的图案，我们必须再用掩码屏蔽掉圆形区域外的一些位。然而这并不容易做到，除非底层的copyPixel有一个针对目标区域的写掩码。使用copyPixel完成任务的直接方式是对像素写多次，因为笔的足迹会在相邻的像素上重叠。一种比较好的方法是运用足迹的跨度来计算在相邻的像素上连续的足迹所形成的跨度，这方法也适宜于处理有关圆形横截面的问题。如同填充定义区域的图元，在一条光栅扫描线上跨段的组合不过是线段的合并，这只需要在每条光栅线上跟踪记录增长的跨段的最小和最大的x值。从图3-35中可以看到一个矩形足迹移动时相邻的两个位置，以及一部分临时的相关

数据结构，它们为每条光栅线保存了跨段的端点。当一个宽多边形或宽椭圆弧可能与一条扫描线相交多次时，每个扫描线桶可能有一列跨段，这很像扫描转换多边形的活动边表。

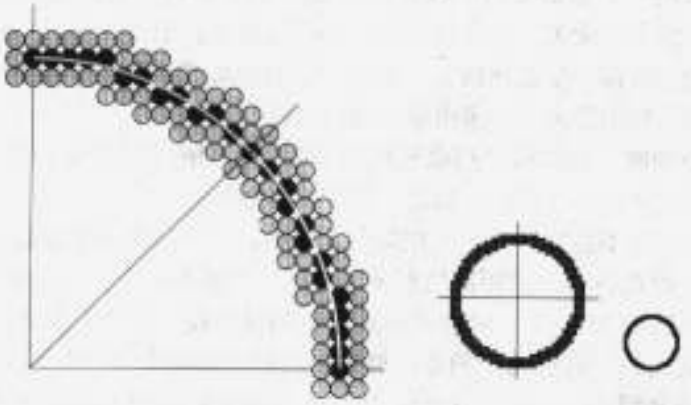


图3-34 通过跟踪一个矩形画笔所画的宽圆

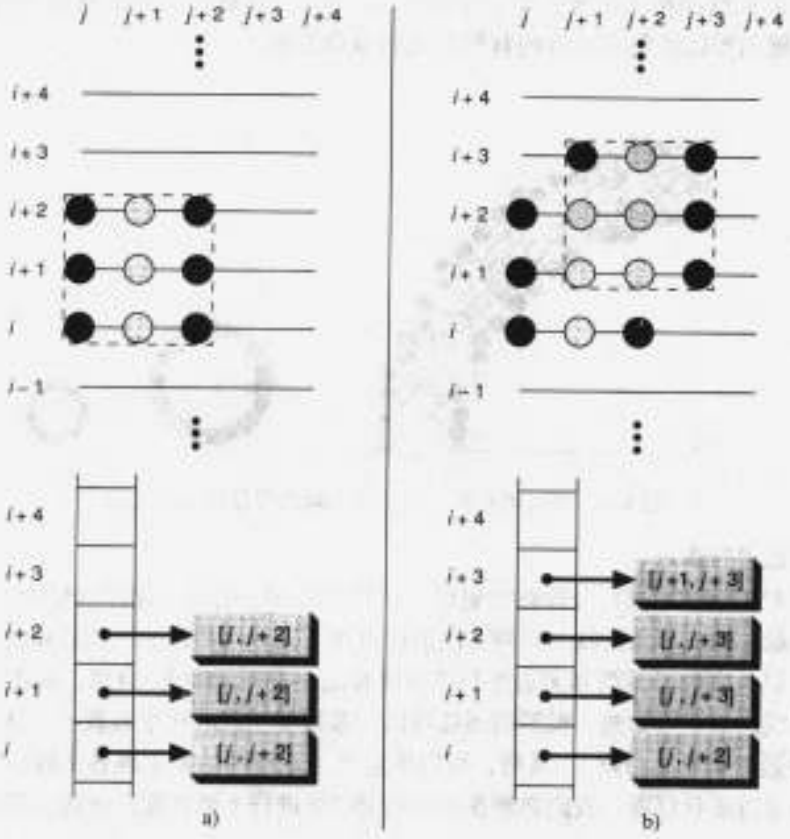


图3-35 为矩形画笔所记录的跨度：a) 在$x=j+1$处的足迹，b) $x=j+2$

3.9.3 填充边界之间的区域

绘制一个宽图元的第三种方法就是构造出图元的内外两个边界，它们分别位于理想（单像素）图元的轨迹的两边，其距离为$t/2$。或者，对于定义区域的图元，我们就让原来的边界作为外部边界，然后往里收缩以画出里面的边界。这种填充技术的优点是，对于奇数和偶数的宽度

都能处理，不必管图元变粗时图元所增大的程度。但是，此方法也有缺点，一个定义区域的图元实际上会“收缩”一点，并且它的“中心线”（即原来单个像素宽的轮廓）会发生偏移。

一条宽线可以用一个矩形来画，该矩形的宽度为t，而长度就是原来线的长度。这样，矩形的宽度与线的倾斜角度无关，并且矩形在两端的边是垂直于该线的。一般地，旋转后倾斜的矩形，其顶点不会正好位于整数栅格上。如此，它们必须进行取整操作将其变换到最靠近的像素上，然后再将这个矩形作为一个多边形进行扫描转换。

若要生成宽边的圆，我们可以扫描转换两个圆，外圆的半径是$R + t/2$，内圆的半径是$R - t/2$，然后填充它们之间的一个或两个跨段，如图3-36所示。

对于椭圆，情况就不是这样简单。在微分几何中，一个经典的结论是：沿着垂直椭圆弧的方向，将此椭圆上的点移动$t/2$的距离所形成的曲线，并不是一个同心的椭圆，而是由一个8次方程所描述的曲线[SALM96]⊖。对这些函数进行扫描转换，计算量很大，因此，我们一般采用逼近的方法。我们扫描转换两个同心的椭圆，内椭圆的两个半径分别为$a - t/2$和$b - t/2$，而外椭圆的两个半径分别为$a + t/2$和$b + t/2$。然后，计算它们之间的跨段并填充它们。填充工作可以在完成了所有跨段的运算后进行，也可实时地进行。当然，绘制细的椭圆所遇到的一些典型问题，在此也依然存在（这将在第19章讨论）。另外，在此提到的为椭圆生成内部边界的问题，在处理光栅图形软件包所支持的其他图元时也会发生。

107
|
108

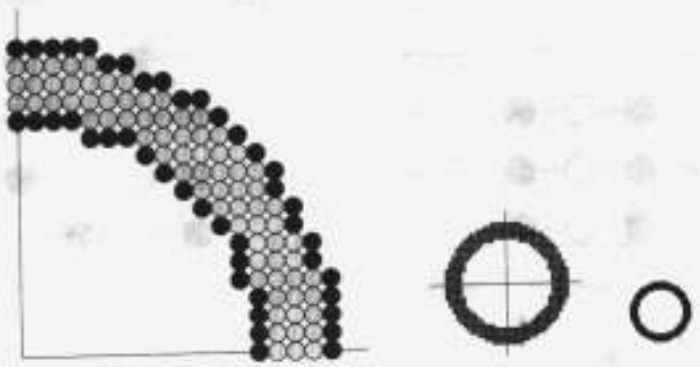


图3-36 通过填充两个同心圆之间的部分画出的宽圆

3.9.4 用宽折线进行逼近

我们可以对任何图元进行分段线性逼近，即计算边界上的点（用浮点坐标），然后再将这些点用线段连接起来形成一条折线。这种方法的优点是可以利用高效的线裁剪和线扫描转换的算法（细图元的），以及多边形裁剪和多边形扫描转换算法（宽图元的）。自然，在图元中朝向剧烈变化的地方，其线段必须非常短。椭圆的圆弧可以用参数多项式的比率来表示，这样，它们就容易进行分段线性逼近（见第11章）。其后，可以用定义了宽度的矩形来画各个线段。当然，为了使这种宽图元的逼近形状好看，我们必须在宽线的交接处进行平滑处理，这将在第19章中讨论。

3.10 线型和笔型

SRGP的线型属性可以影响各种轮廓图元。一般地，我们必须用条件逻辑来检测是否写一个像素，当然，只有为1的时候才写。我们将图案的写掩码当作一列16个布尔值（例如，一个

⊖ 这样生成的8次曲线可能会有自交点或尖点，亲手绘制这些法线就能看出这一点。

16位的整数）来存储；因此，每隔16个像素就会重复。实现时我们将线扫描转换算法中无条件WritePixel语句修改成如下面的条件语句：

```
if (bitstring[i % 16])
    WritePixel (x, y, value);
```

在此，序号*i*是内循环中一个新的递增变量，它是为了反复使用掩码而设的。但是，这种方法有一个缺点。因为掩码中的每一位对应于循环中的一次迭代操作，而不是对应于线上的一个单位距离，因此，短划线的长度是随着线的倾斜角度变化的；与水平短划线和垂直短划线相比，有一定角度的短划线要长一些。对于工程绘图，这种差异是不能允许的。短划线必须作为独立的线段来计算和扫描转换，其长度不能随角度变化。生成宽线时，则运用一列交替变换为填实的或透明的矩形来完成。这些矩形的顶点是根据所选择的线型计算出来的。然后，这些矩形就可以独立地进行扫描转换；对于水平线和垂直线，程序可以用copyPixel来拷贝矩形。

109

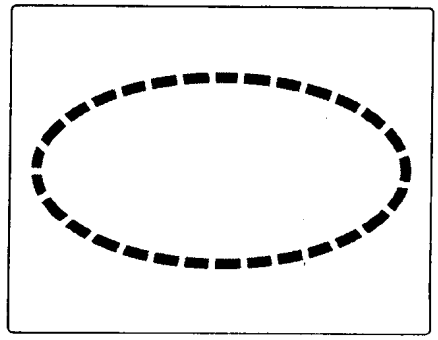


图3-37 综合笔图案和线型的作用

在处理宽轮廓图元时，线型和笔型会相互作用。线型用于计算每条短划线的矩形，而每个矩形用所选择的笔图案来填充（图3-37）。

3.11 光栅空间的裁剪操作

正如我们在本章概论中提到的那样，尽可能快地进行裁剪和扫描转换是非常重要的，以便在应用模型有所变化时能够很快地进行更新。裁剪可以解析地进行；在扫描转换过程中实时地进行；或者作为实际裁剪矩形的copyPixel命令的一部分，在将存储了未经裁剪的图元的画布复制到另一个画布上去的时候进行。上述第三种方法在以下情况下很有用：预先生成一个大的画布，然后用户根据需要移动裁剪矩形来检查这画布的一些部分，但不修改画布上的内容。

将裁剪和扫描转换结合起来，有时称为*截剪*，这在填充图元和画宽图元时容易做到，因为这可以作为跨段操作的一部分：只有端点需要裁剪，而内部点不需要检测。截剪也显示了跨段相关性的另一个优点。再者，如果一个轮廓图元并不比裁剪矩形大许多，即相对而言，图元落在裁剪区域外的像素不多。对于这样的情况，与提前进行解析的裁剪相比，在生成每个像素的同时实施裁剪的操作可能要快许多（即对其进行有条件限制的写操作）。特别是，尽管区间检测是在内循环中进行的，但大量的对外部像素的写存储器的操作可以节省，并且增量计算和检测操作可以完全在一个高速存储器中（如CPU的指令高速缓存或者显示控制器的微码存储器中）进行。

110

另外，还有一些技巧可能是有用的。例如，在运用标准的中点扫描转换算法时，每次都挑选当前像素后面第*i*个像素进行下一次操作，将这个像素与矩形的边界进行比较，直至找到第一个在裁剪区内的像素，由此可知线已经通过它与一条裁剪边的交点进入了裁剪区域。然后，逐个像素地回溯以找到进入裁剪区域的第一个像素，再进行一般的扫描转换。同理，也可以类似地找到线在裁剪区内的最后一个像素，或者将检测每个像素作为扫描转换的循环操作中的一部分，并且一旦检测不成立就停止扫描转换。一般地，其间隔选择为8是比较合适的，因为在进行检测的次数和回溯的像素个数之间，这是一个比较好的折中（见习题3.26）。

对于用浮点数进行操作的图形包来说，最好是在浮点坐标系中进行解析的裁剪，然后再对

裁剪后的图元进行扫描转换。此时，要注意对判定变量进行正确的初始化，如在3.2.3节中我们处理线时所做的那样。对于整型的图形包，如SRGP，要选择是预先裁剪再扫描转换还是在扫描转换中进行裁剪。因为对线和多边形进行解析的裁剪相对要容易一些，因此对它们常常是先裁剪再扫描转换。但是对其他图元，在扫描转换过程中进行裁剪就要快一些。而在一个浮点型的图形包中，最常见的是在浮点数坐标系中进行解析的裁剪，然后再调用下层的扫描转换软件来生成裁剪后的图元。这种整型的图形软件随后可以相对于矩形（甚至任意形状）窗口边界在光栅上再进行一次裁剪。因为解析地裁剪图元对整型图形包以及2D和3D的浮点型图形包都是可用的，我们就在本章讨论一些基本的解析裁剪算法。

3.12 线段裁剪

本节讨论用矩形裁剪线的解析方法[⊖]；裁剪其他图元的算法，将在后面的几节中讨论。尽管有一些关于矩形裁剪和多边形裁剪的特别算法，但是值得注意的是，对SRGP的由线条组成的图元（例如，折线、非填充的矩形和多边形）的裁剪，可以通过反复运用裁剪线的方法来实现。特别是，因为圆和椭圆可以用一系列很短的线进行分段的线性逼近，因此，它们的边界可以当作是一个折线或多边形来进行裁剪和扫描转换。在一些系统中，二次曲线是由参数多项式的比率表示的（见第11章），这种表示很适于递增的分段线性逼近，这样，它们可方便地运用裁剪线的算法。用一个矩形裁剪一个矩形的结果，最多是一个矩形。用一个矩形裁剪一个凸多边形的结果，最多也是一个凸多边形。但是，裁剪一个凹多边形就可能会得到多个凹多边形。用一个矩形裁剪一个圆或椭圆的结果，最多是4个弧段。

线与矩形裁剪区域（或者任意凸多边形的裁剪域）的交，通常是被裁剪成一条线段；线若与
111 矩形裁剪框的边重合，也认为是在裁剪框内并且被显示出来。图3-38列出了裁剪线的几种情况。

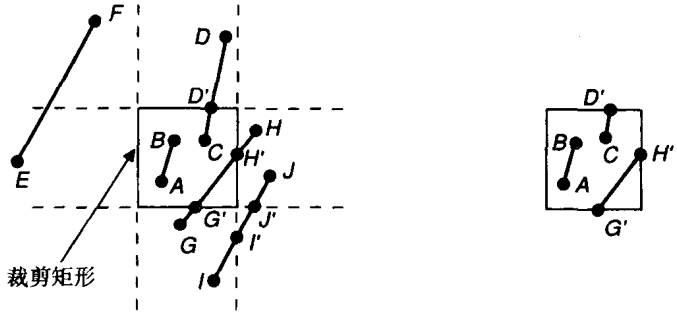


图3-38 裁剪线的几种情况

3.12.1 裁剪端点

在讨论裁剪线以前，我们先看看简单的裁剪单个点的问题。如果裁剪矩形的x坐标的边界是x_{min}和x_{max}，而y坐标的边界是y_{min}和y_{max}，那么一个位置为(x, y)的点在裁剪矩形内，就必须满足下面的4个不等关系：

$$x_{min} \leqslant x \leqslant x_{max},\ y_{min} \leqslant y \leqslant y_{max}$$

⊖ 本节不讨论图元相对于多个矩形的裁剪（比如，在窗口系统中窗口重叠的情况），也不讨论非矩形裁剪域的情况。在19.7节，将简单地讨论一下非矩形裁剪域的情况。

如果不能满足这4个不等关系中的任意一个，这个点就在裁剪矩形外。

3.12.2 利用求解联立方程组的线段裁剪

裁剪一条线，我们只需考虑它的端点，而不必关心它无穷的内部点。如果一条直线的两个端点都在裁剪矩形内（例如，图3-38中的线*AB*），那么该线完全在裁剪矩形内，因此就“简单接受”。如果一个端点在外，而另一个在内（比如图中的线*CD*），则线与裁剪矩形相交，我们必须计算出交点。如果两个端点都在外，则线可能与裁剪矩形相交，也可能不相交（图中的线*EF*, *GH*, *IJ*），此时，我们必须做进一步的计算以决定是否有相交的部分，如果有，则需决定它们在什么位置。

一种简单的裁剪线的方法是将裁剪矩形的4条边都与线求交，看是否有交点在这些边上。如果有，这条线就与裁剪矩形相交，且有一部分在裁剪矩形内。所以，对于每一条线和每一条裁剪矩形的边，我们选取两条在数学上具有无穷长度的线，它们分别与被检测的线和裁剪矩形的边重合。然后，求这两条无穷长度的线的交点，并判断交点是否在“里面”，也就是说，它是否既在被检测的线上，又在裁剪矩形的边上。如果是，则被检测线与裁剪矩形相交。在图3-38中，交点*G'*和*H'*在里面，而*I'*和*J'*在外面。

我们运用这个方法时，对每一个 <边，线> 对，要用乘法和除法求解两个联立方程。尽管可 112
以利用解析几何中关于线的方程表达式，但那是描述无穷长度的线的，而在图形学和裁剪中，我们处理的是有穷长度的线（在数学上，它们叫线段）。再说，线的斜距式不能处理垂直线这种特殊情况，比如裁剪矩形中直立的边。线段的参数式表达式可以解决这两个问题：

$$x = x_0 + t(x_1 - x_0),\ y = y_0 + t(y_1 - y_0)$$

此方程组刻画了从（x_0, y_0）到（x_1, y_1）的有向线段上的点（x, y），其中，参数t的变化范围是[0, 1]。只需要简单地将参数t替换掉，就能回到常见的线性方程。对关于边和线段的两组联立方程组的参数t_{edge}和t_{line}进行求解，并检查t_{edge}和t_{line}是否都在[0, 1]之中。如果是，则其交点位于这条边和这条线段中，它便是线与裁剪矩形的交点。特别是，在解联立方程前，必须检测是否有线与裁剪矩形的边平行这种特殊情况。总之，这种直接计算的方法要做许多计算和检测，它的效率不高。

3.12.3 Cohen-Sutherland 线裁剪算法

Cohen-Sutherland算法是一种很有效的裁剪算法，它在初始化时，对线做一些检测，以决定是否可以不进行求交计算。该方法首先检测线的两个端点，看它们是否都在裁剪矩形内。若是，则该线便完全在裁剪矩形内；否则就进行下一步的区域检测。例如，在图3-38中的线*EF*，只需对它做两次简单的x坐标的比较，就知道它的两个端点的x坐标均小于x_{min}，这样，它就在裁剪矩形左边的区域（即位于左端的边所定义的外半平面内）。于是，线*EF*就能被简单地拒绝，而不必进行裁剪和显示了。类似地，对于x_{max}所定义的右边的区域、y_{min}定义的下边的区域和y_{max}定义的上边的区域，只要线段的两个端点都同在某一个区域，就知道这条线必不在裁剪矩形内，可以简单地裁剪掉。

若通过上面的步骤不能将线段简单地接受或拒绝，就由一条裁剪边将该线段分成两个部分，其中有一个部分可以简单地拒绝。如此循环地对线段进行裁剪和检测是否简单地接受或拒绝的操作，直至剩余的部分完全在裁剪矩形内或能被简单地拒绝。这个算法对两种常见的情况特别有效。第一种情况是裁剪矩形非常大，它几乎覆盖了整个显示区域，此时，绝大部分图元都能简单地接受。第二种情况是裁剪矩形很小，几乎可以简单地拒绝所有的图元，这在运用鼠标进行图元选择的工作时会遇到，此时，环绕鼠标的一个小矩形（称为拾取窗口）用来裁剪图元，以判断哪些图元位于鼠标选择点附近的小范围（矩形）内（见7.12.2节）。

在执行简单地接受或拒绝的检测操作时，我们将裁剪矩形的边延长并由此将平面划分成9个区域（见图3-39）。每个区域分配一个4位的代码，这个代码是根据裁剪矩形的边所定义的外半平面和各个区域的相对关系来决定的。外码中的每一位被置成1（真）或0（假）；这样的4
113 位代码对应以下情况：

第1位，上端边所定义的外半平面，在上端边上方，$y>y_{max}$

第2位，下端边所定义的外半平面，在下端边下方，$y<y_{min}$

第3位，右端边所定义的外半平面，在右端边右方，$x>x_{max}$

第4位，左端边所定义的外半平面，在左端边左方，$x>x_{min}$

例如，对既在裁剪矩形的上方又在它的左方的区域，它位于上端边和左端边所定义的两个外半平面内，它的代码就是1001。基于下述的原理，我们可以得到一个计算外码的很有效的方法，即各位的值可分别根据$(y_{max}-y)$、$(y-y_{min})$、$(x_{max}-x)$、$(x-x_{min})$的符号决定。然后，线段的每一个端点根据它所在的区域被赋予该区域的代码。根据线段的两个端点的代码，我们可以判断线段是完全在裁剪矩形里面还是在某条边所定义的外半平面内。如果这两个端点的代码中的每一位都是0，那么这条线段完全在裁剪矩形内。当然，如果两个端点都位于某条边所定义的外半平面内，比如图3-38中的线EF，则它的两个端点的代码在相应于这条边的位上都会被置成1，以表示这些点位于这条边所定义的外半平面内。对于线EF，它的外码分别是0001和1001，由第4位上的值可知该线段位于左端边所定义的外半平面内。因此，对这两个代码进行按位“与”的逻辑操作，如果结果不为0，则这条线段可以被简单拒绝。

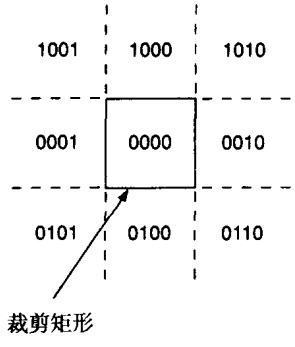


图3-39 区域的外码

如果一条线段不能简单接受或拒绝，我们必须将它分成两段，再进行判断以去掉其中的一段或二段。此时，我们用与这条线相交的边来划分这条线，并将位于这条边所定义的外半平面内的那一段去掉。在检测哪条边与这条线相交时，边的选择次序可以是任意的，但在整个算法的运行过程中，这个次序必须保持不变。下面，我们将采用生成其外码的次序：由上到下，再由右到左。外码的一个重要特点是它的非零位对应于这条线会相交的边：如果一个端点位于一条边所定义的外半平面，但这条线又不能简单拒绝，则它的另一个端点必定位于这条边所定义的内半平面内，因此，这条线一定与这条边相交。所以，算法就选择一个在外面的点，并根据该点的外码中非零的位来决定一条裁剪边；所选择的边是在“由上到下，再由右到左”次序中遇到的第一条边，即外码中最左的非零位。

114 算法按下列方式运行。计算线段的两个端点的外码，并检测它们是否能简单接受或拒绝。如果不能，我们就选择一个在外面（至少有一个点在外面）的点，然后检测它的外码找到一条会与该线相交的边，并求出交点。随后，我们去掉从这个外面点到交点的这一段，并将交点作为裁剪后的线段的一个新的端点。最后为这个新的端点计算其外码，并准备进行下一次循环操作。

例如，考察图3-40中的线段AD。点A的外码是0000，点D的外码是1001。这条线既不能简单接受也不能简单拒绝。因此，算法选择外部点D，因为它的外码显示这条线会与裁剪矩形的上端边和左端边相交。根据我们的检测次序，我们首先选择上端边将线AD裁剪成线AB，并计算出B的外码是0000。在下一次循环操作时，我们运用简单接受或简单拒绝的测试，就能简单

接受*AB*并显示它。

裁剪线*EI*需要多次循环操作。第一个端点*E*的外码是0100，因此算法选择它作为外部点，并根据其外码知道这条线首先相交的边是下端边，由此边将*EI*裁剪为*FI*。在第二次循环操作时，不能简单接受或拒绝*FI*。因为端点*F*的外码是0000，所以算法选择外码为1010的外部点*I*。此时所选择的相交边是上端边，由此边裁剪出线*FH*。*H*的外码确定为0010，所以，第三次循环操作所用的裁剪边是右端边，并裁剪出线*FG*。在第四次循环操作时，这条线可以简单接受，因此循环结束，并显示这条线。如果开始时我们选择*I*作为外部点，则会得到另一种选取裁剪边的次序：根据它的外码，我们将首先裁剪上端边，然后是右端边，最后是下端边。

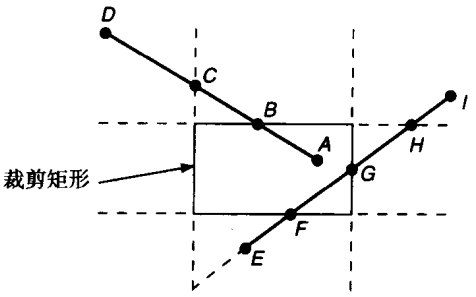


图3-40 裁剪线的Cohen-Sutherland算法示意图

在图3-41的代码中，我们用常整数和按位算术来表示其外码，这比用一个数组表示其外码更自然一些。在此，我们用一个子程序对这些外码进行二进制的合成运算。为改善性能，我们当然要对这些代码进行排序。

通过避免重复计算斜率，我们可以将算法的效率稍微提高一些（见习题3.28）。不过即使有此改进，这个算法也不是效率最高的。因为检测和裁剪是按照一个固定的次序运行的，该算法有时会做一些不必要的裁剪。当线段与矩形的边的交点是“外部交点”时就是这种情况：也就是说，交点不在裁剪矩形的边界上（比如，图3-40中在线*EI*上的点*H*）。相比较而言， 115
Nicholl、Lee和Nicholl[NICH87]的算法就可以避免计算外部交点，其方法是将平面划分成许多的区域，这将在第19章中讨论。在此介绍的简单的Cohen-Sutherland算法的一个优点是：它可以直接拓展应用于三维正交视见体，这将在6.5.3节中讨论。

```
typedef unsigned int outcode;
enum {TOP = 0x1, BOTTOM = 0x2, RIGHT = 0x4, LEFT = 0x8};

void CohenSutherlandLineClipAndDraw (
    double x0, double y0, double x1, double y1, double xmin, double xmax,
    double ymin, double ymax, int value)
/* 裁剪从(x0, y0)到(x1, y1)的线的Cohen-Sutherland裁剪算法 */
/* 其裁剪矩形的对角顶点为(xmin, ymin)和(xmax, ymax)。 */
{
    /* P0和P1，以及位于裁剪矩形外的任何点的外码。 */
    outcode outcode0, outcode1, outcodeOut;
    boolean accept = FALSE, done = FALSE;
    outcode0 = CompOutCode (x0, y0, xmin, xmax, ymin, ymax);
    outcode1 = CompOutCode (x1, y1, xmin, xmax, ymin, ymax);
    do {
        if (!(outcode0 | outcode1)) {              /* 简单接受并结束算法 */
            accept = TRUE; done = TRUE;
        } else if (outcode0 & outcode1)            /* 逻辑“与”为真，所以简单拒绝，并结束算法 */
            done = TRUE;
        else {
            /* 两种检测失败，所以计算线段被剪掉的部分： */
            /* 从一个外部点到线与一条裁剪边的交点 */
```

图3-41 Cohen-Sutherland线裁剪算法

```
        double x, y;
        /* 至少有一个外部点，拾取它 */
        outcodeOut = outcode0 ? outcode0 : outcode1;
        /* 现在，找交点 */
        /* 所用公式是：y = y0 + slope*(x - x0), x = x0 + (1/slope)*(y - y0) */
        if (outcodeOut & TOP) {                    /* 在裁剪矩形的上端边划分线段 */
            x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
            y = ymax;
        } else if (outcodeOut & BOTTOM) {          /* 在裁剪矩形的下端边划分线段 */
            x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
            y = ymin;
        } else if (outcodeOut & RIGHT) {           /* 在裁剪矩形的右端边划分线段 */
            y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
            x = xmax;
        } else {                                   /* 在裁剪矩形的左端边划分线段 */
            y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
            x = xmin;
        }
        /* 现在，我们将外部点移到交点进行裁剪 */
        /* 并准备下一次循环操作 */
        if (outcodeOut == outcode0) {
            x0 = x; y0 = y; outcode0 = CompOutCode (x0, y0, xmin, xmax, ymin, ymax);
        } else {
            x1 = x; y1 = y; outcode1 = CompOutCode (x1, y1, xmin, xmax, ymin, ymax);
        }
    }  /* 细分 */
} while (done == FALSE);

    if (accept)
        MidpointLineReal (x0, y0, x1, y1, value);    /* 双精度坐标画线过程 */
}  /* CohenSutherlandLineClipAndDraw */

outcode CompOutCode (
      double x, double y, double xmin, double xmax, double ymin, double ymax);
{
    outcode code = 0;
    if (y > ymax)
        code |= TOP;
    else if (y < ymin)
        code |= BOTTOM;
    if (x > xmax)
        code |= RIGHT;
    else if (x < xmin)
        code |= LEFT;
    return code;
}  /* CompOutCode */
```

图3-41 （续）

3.12.4 参数化的线裁剪算法

Cohen-Sutherland算法可能至今仍是最常用的线裁剪算法，因为它提出得很早并且得到了广泛的传播。1978年，Cyrus和Beck提出了一个完全不同但更有效的线裁剪算法[CYRU78]。Cyrus-Beck算法可以用来在平面上由一个矩形或一个任意形状的凸多边形对一条二维的线进行裁剪，或者在空间由一个任意的凸多面体对一条三维的线进行裁剪。后来梁友栋和Barsky独立

地提出了一个更有效的参数化线裁剪算法，特别是，该算法在处理二维或三维的直立矩形裁剪区域时非常快[LIAN84]。除了发掘简单的裁剪边界的优点外，他们还为一般的裁剪区域引入了更有效的简单拒绝的检测。在此，我们从初始的Cyrus-Beck算法开始来介绍参数化的裁剪操作。但是，因为我们关注的只是直立的矩形裁剪矩形，在讨论的最后，我们将Cyrus-Beck算法归约为更有效的梁友栋–Barsky算法。

在Cohen-Sutherland算法中，对于不能简单接受或拒绝的线都要计算这条线与一条裁剪边的交点(x, y)，在求解过程中，是将垂直或水平的裁剪边的x或y坐标值代入线的方程进行计算。然而，在线的参数方程中，求取的是参数t，它对应于裁剪边所在的无穷长的线与被裁剪线的交点。广义地说，4条裁剪边都会与被裁剪线相交，这样，就需要算出4个t值。然而只需经过一系列的简单比较，以判断这4个t值中的哪些（如果存在）对应于真正的交点。随后，只对一个或两个真正的交点坐标(x, y)进行计算。显然，与Cohen-Sutherland中的求交算法相比，该算法会节约许多时间，因为它不必求线段与各个裁剪边的交点，可减少循环操作的次数。再说，在一维参数空间中的计算要比三维坐标空间中的计算简单。在Cyrus-Beck算法的基础上，梁友栋和Barsky做了进一步的改进，一旦一个t值计算出来就进行检测，并裁剪掉一部分线段，而不必等到4个t值都计算出来后再进行检测。

116 ~ 117

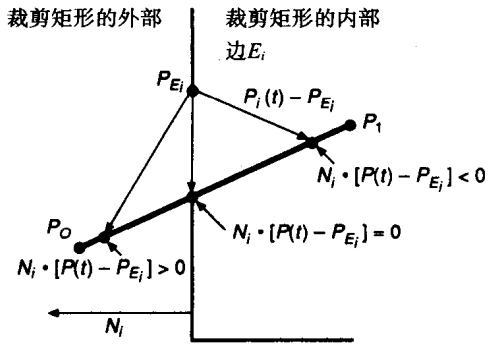


图3-42 外部点、内部点和裁剪框边界上的点的点积

Cyrus-Beck算法的基础是下面的两条线求交的公式。图3-42中显示了一条裁剪边E_i和这条边向外的法向N_i（也就是指向裁剪矩形外的方向[⊖]），以及从P_0到P_1这条将被该边裁剪的线段。在求交点时，可能要对这条裁剪边或线段进行延长。

如同前面所介绍的，这条线的参数化表达式是：

$$P(t) = P_0 + (P_1 - P_0)t$$

其中，在P_0处$t = 0$，而在P_1处$t = 1$。现在，我们在边E_i上任取一点P_{Ei}，并考察从P_{Ei}到线段P_0P_1上3个点的3个向量，这3个点是：将被确定的交点、位于裁剪边所定义的内半平面内的端点和位于裁剪边所定义的外半平面内的端点。通过计算点积$N_i \cdot [P(t) - P_{Ei}]$的值，我们可以知道各个点位于哪个区域。对于在内半平面内的点，其点积是负数；对于在边上的点，点积是零；而对于在外半平面内的点，其点积是正数。关于一条边的内外半平面的定义对应于沿裁剪区域边的逆时针顺序。这是我们在本书中将始终遵循的一条常规。现在，我们可以用下面的方程来求解线与边相交时的参数t的值：

$$N_i \cdot [P(t) - P_{E_i}] = 0$$

首先，替换掉$P(t)$后得到

$$N_i \cdot [P_0 + (P_1 - P_0)t - P_{E_i}] = 0$$

然后，合并同类项，并运用点积的分配定律，我们得到

⊖ Cyrus和Beck用的是向内的法线，但我们倾向于用向外的法线，这是为了与三维空间中定义平面的法线保持一致，因为它们是向外的。因此，我们这里所用的方式与Cyrus和Beck的差别只是符号的检测不同。

$$N_i \cdot [P_0 - P_{E_i}] + N_i \cdot [P_1 - P_0]t = 0$$

设从P_0到P_1的向量为$D = (P_1 - P_0)$，于是t的解为:

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D} \tag{3-1}$$

注意，只有在表达式中除数不为零时，我们才能得到一个有效的t值。

为保证这一点正确，算法要做如下检测：

$N_i \neq 0$（即法线不能为0；只有出现错误时，才会发生这种情况），

$D \neq 0$（即$P_0 \neq P_1$），

$N_i \cdot D \neq 0$（也就是，边E_i和从P_0到P_1的线段不平行。如果它们平行，就不会与这条边有单个交点，如此，算法就得对这种情况进行进一步的讨论）。

式(3-1)可以用来计算线段P_0P_1与裁剪矩形的每一条边的交点。在做这种计算时，我们要为每条边确定法向和边上的任意一个点P_{E_i}（比如说，这条边的一个端点），随后，这些值可用来求该边与所有线段的交点。假设为一条线段已求出了4个参数t值，下一步就是要判断其中哪些值确实相应于裁剪框的边与线段的真正的交点。首先，在区间[0, 1]之外的t值可以去掉，因为，它位于线段P_0P_1之外。然后，我们需要判断交点是否在裁剪边界上。

从图3-43中线1的情况得到启发，我们可以简单地对余下的t值进行排序，并选择中间的t值来求取交点。但我们怎样将线1的情况与线2的情况区别开来呢？在线2的情况中，这条线与裁剪矩形没有任何相交的部分，而中间的t值相应的点也不在裁剪边上。再者，线3上的4个交点，哪些又在边界上呢？

根据下面的原则，图3-43中所示的交点可以相对于裁剪矩形，形象地分为“可能进入点”（PE）和“可能离开点”（PL）。这些原则是：从P_0往P_1移动时，如果跨过一条边就进入该边所定义的内半平面，这个交点就是一个PE；否则，如果是离开该边所定义的内半平面，则该交点是一个PL。我们注意到，根据这样的区分，一条线与裁剪矩形的两个靠里面的交点会有不同的标志。

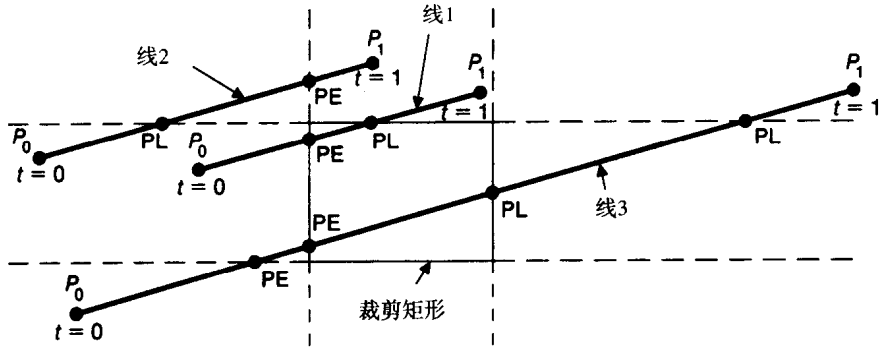


图3-43 沿着裁剪矩形的对角方向分布的一些线段

正式地说，根据线P_0P_1与N_i的夹角可以将交点分成PE和PL两类：如果夹角小于90°，交点就是PL的；夹角大于90°，交点就是PE的。这些信息都隐含在N_i和P_0P_1的点积的符号里了：

$$N_i \cdot D < 0 \Rightarrow \text{PE}（\text{夹角大于90°}），$$

$$N_i \cdot D > 0 \Rightarrow \text{PL}（\text{夹角小于90°}）。$$

注意$N_i \cdot D$只是式(3-1)中的除数，这意味着，在计算t的过程中，我们就能很容易地知道交点是什么类型的。

根据这样的分类，由图3-43中的线3可以得到算法的最后步骤。也就是，我们必须选择一个能确定最后被裁剪的线的（PE, PL）对。对于穿过P_0P_1的无穷长的线，其在裁剪矩形内的线段是由一个PE点和一个PL点确定的，其中，PE点有最大t值t_E，而PL点有最小t值t_L。于是，裁剪出的线段就由（t_E，t_L）范围确定了。但是，因为我们关心的是线段P_0P_1上被裁剪出来的部分，而不是无穷长的线，因此，我们要对这个范围做进一步的修改使得于t_E的下界必是$t=0$，而t_L的上界是$t=1$。如果$t_E > t_L$会怎样呢？这正是线2的情况。这时意味着线段P_0P_1与裁剪矩形没有相交部分，因此，整条线段都被拒绝。真实交点的t_E和t_L的值将用来计算相应的x和y的坐标值。

图3-44给出了关于直立矩形的完整裁剪算法的伪代码，而在表3-1中，为每条边列出了其上的法向N_i、边上的一个典型的点P_{Ei}、向量$P_0 - P_{Ei}$和参数t。有趣的是，每个法向量中总有一个坐标量是0，这样，对P_{Ei}中相应的这个坐标值，我们就不必限定了（由一个不确定的x或y表示）。实际上，裁剪边都是水平或垂直的，因此，我们自然地利用许多简化操作。所以从表中我们可以看到，被除数，也就是决定端点P_0是在一个特定边的里面还是外面的点积$N_i \cdot (P_0 - P_{Ei})$，可以简化成从该点到这条边的有向水平距离或垂直距离。这正好是为其Cohen-Sutherland算法中的代码相应的部分所计算的量。除数，即判断交点是“可能进入点”还是“可能离开点”的点积$N_i \cdot D$，可以简化成$\pm dx$或dy：如果dx是正数，则线段从左往右移动，且与左端边的交点是PE，而与右端边的交点是PL，等等。最后，参数t，也就是被除数和除数的比值，可以简化成到边的距离除以dx或dy，dx或dy是可以直接从线的参数方程中计算出来的比例常量。值得注意的是，保留除数和被除数的符号而不是取消减号是很重要的，因为除数和被除数作为带符号的距离量附带了许多有用的信息，这些都要在算法中使用。

118
~
120

```
预计算Ni，并为每条边选择一个PEi点；

for（每条要被裁剪的线段）          {
    if (P1 == P0)
        退化成一个点的线被作为一个点进行裁剪；
    else {
        tE = 0; tL = 1;
        for（与一条裁剪边相关的每个可能的交点）        {
            if (Ni · D != 0) {      /* 现在，不考虑与裁剪边平行的边 */
                计算t；
                用Ni · D的符号来区分PE和PL；
                if (PE) tE = max (tE, t);
                if (PL) tL = min (tL, t);
            }
        }
        if (tE > tL)
            return NULL;
        else
            return P(tE)和P(tL)作为真正的裁剪交点；
    }
}
```

图3-44 Cyrus-Beck参数化线裁剪算法的伪代码

图3-45给出了[LIAN84]算法改编后的完整的代码。程序中调用了一个内部函数CLIPt()来计算交点处的参数值，而且，因为新的t_E或t_L值肯定要分别越过旧的t_L或t_E值，在此，可以检测能否简单拒绝。该内部函数是用除数的符号来判断线段与边的交点是PE点还是PL点。如果线段平行于裁

剪边并且在外部，则肯定简单拒绝它，也就是说，它根本不可见。这样，主程序通过将端点移到最新计算的t_E或t_L处来完成实际的裁剪工作。当然，这只有线段确实有片段在裁剪矩形内时才会执行。判断一个线段是否被拒绝的函数所返回的标志，由一个有4层**if**嵌套语句的条件检测来测试。

表3-1 参数化线裁剪算法中的计算

裁剪边i	法向量N_i	P_{Ei}	$P_0 - P_{Ei}$	$t = \frac{N_i \cdot (P_0 - P_{Ei})}{-N_i \cdot D}$
左端：$x = x_{min}$	$(-1, 0)$	(x_{min}, y)	$(x_0 - x_{min}, y_0 - y)$	$\frac{-(x_0 - x_{min})}{(x_1 - x_0)}$
右端：$x = x_{max}$	$(1, 0)$	(x_{max}, y)	$(x_0 - x_{max}, y_0 - y)$	$\frac{(x_0 - x_{max})}{-(x_1 - x_0)}$
下端：$y = y_{min}$	$(0, -1)$	(x, y_{min})	$(x_0 - x, y_0 - y_{min})$	$\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$
上端：$y = y_{max}$	$(0, 1)$	(x, y_{max})	$(x_0 - x, y_0 - y_{max})$	$\frac{(y_0 - y_{max})}{-(y_1 - y_0)}$

注：每条边上点P_{Ei}的准确坐标对于计算无关紧要，因此它们可以用不定变量x和y来表示。比如对于左端边上的一个点，$x = x_{min}$，就如表中第一行第三个量所示。

```
void Clip2D (double *x0, double *y0, double *x1, double *y1, boolean *visible)
/* 用一个对角顶点为（xmin, ymin）和（xmax, ymax）的裁剪矩形， */
/* 来裁剪从（x0, y0）到（x1, y1）的二维线段。它们是一些全局量， */
/* 当然也可以作为参数来传递。如果以端点参数的形式返回一条线段 */
/* 的裁剪结果，标志变量visible就设置成TRUE。如果一条线段被拒绝， */
/* 端点不会被改变，并且visible被设置成FALSE。 */
{
    double dx = *x1 - *x0;
    double dy = *y1 - *y0;
    /* 只有当线段位于所有4条边的内部时，才生成输出结果。 */
    *visible = FALSE;
    /* 首先，检测退化的线，并裁剪这个点； */
    /* 如果这个点在裁剪矩形内，ClipPoint就返回TRUE。 */
    if (dx == 0 && dy == 0 && ClipPoint (*x0, *y0))
        *visible = TRUE;
    else {
        double tE = 0.0;
        double tL = 1.0;
        if (CLIPt (dx, xmin - *x0, &tE, &tL))                    /* 位于wrt的左端边的里面 */
            if (CLIPt (-dx, *x0 - xmax, &tE, &tL))               /* 位于wrt的右端边的里面 */
                if (CLIPt (dy, ymin - *y0, &tE, &tL))            /* 位于wrt的下端边的里面 */
                    if (CLIPt (-dy, *y0 - ymax, &tE, &tL)) {     /* 位于wrt的上端边的里面 */
                        *visible = TRUE;
                        /* 如果tL移动了，计算PL交点。 */
                        if (tL < 1) {
                            *x1 = *x0 + tL * dx;
                            *y1 = *y0 + tL * dy;
                        }
                        /* 如果tE移动了，计算PE交点。 */
                        if (tE > 0) {
                            *x0 += tE * dx;
```

图3-45 梁友栋-Barsky参数化线裁剪算法的代码

```
                    *y0 += tE * dy;
                }
            }
        }
    } /* Clip2D */

boolean CLIPt (double denom, double num, double *tE, double *tL)
/* 这个函数是为一条线和一条边的一个内部交点计算 */
/* 一个新的tE或tL的值。参数denom是 - (Ni · D)，对 */
/* 于直立的裁剪矩形，它被简化成 ± Δx, Δy（如表3-1 */
/* 中所示）。它的符号可以判定交点是PE还是PL。对 */
/* 于一种特别的线/边组合，参数num是Ni · (P0 - PEi)， */
/* 它可以简化成从P0到一条边的有向垂直距离或水平 */
/* 距离。它的符号可以判定P0是否可见，并用来简 */
/* 单拒绝水平线或垂直线。如果线段可被简单拒绝， */
/* 就返回FALSE；否则，就返回TRUE，并且如果需要， */
/* 就为线段在边的里面的部分调整tE或tL的值。 */
{
    double t;

    if (denom > 0) {          /* PE交点 */
        t = num / denom;      /* 在交点的t的值 */
        if (t > tL)           /* 越过了tE和tL */
            return FALSE;     /* 所以，准备拒绝线段 */
        else if (t > tE)      /* 找到一个新的tE */
            tE = t;
    } else if (denom < 0) {   /* PL交点 */
        t = num / denom;      /* 在交点的t的值 */
        if (t < tE)           /* 越过了tE和tL */
            return FALSE;     /* 所以，准备拒绝线段 */
        else                  /* 找到一个新的tL */
            tL = t;
    } else if (num > 0)       /* 线段在边的外面 */
        return FALSE;
    return TRUE;
} /* CLIPt */
```

图3-45（续）

概括地说，如果对于外码检测的开销不大（比如，在汇编语言中运行位操作），并且对大部分线段可以简单拒绝或接受，Cohen-Sutherland算法是很有效的。当要对很多线进行裁剪时，参数化的线裁剪方法更好一些，因为它尽量避免了求交点坐标的计算，并且是基于参数值进行检测的。但是，在Cohen-Sutherland算法中本可以简单接受的端点，在参数化算法中却要进行参数计算。梁友栋-Barsky算法比Cyrus-Beck算法更有效的原因是，对于不会与裁剪矩形相交的线段，它所附加的简单拒绝的检测可以避免计算所有4个参数值的工作。对于不在非可见半平面内的线段，Cohen-Sutherland算法不能简单地拒绝，而必须重复多次裁剪才能判断出来，而梁友栋-Barsky算法中关于拒绝的检测就可以做到这一点。一般地说，19.1.1节中Nicholl等人提出的算法要比Cohen-Sutherland算法和梁友栋-Barsky算法都好，但它不能像参数化裁剪那样推广到三维。对Cohen-Sutherland算法的加速处理在[DUVA90]中讨论。对于本节所讨论的这两个算法，习题3.29涉及到它们运行时执行的指令次数，这也是比较它们在各种条件下工作效率的一种手段。

121
~
123

3.13 圆和椭圆的裁剪

用一个矩形裁剪一个圆，我们首先要用下一节将介绍的多边形裁剪算法做一个简单的拒绝或接受的检测，即求圆所在的方形区域（即边长为圆的直径的正方形）与裁剪矩形的交。如果圆与裁剪矩形相交，我们就将圆均分成4个部分并对每个部分进行简单拒绝/接受的检测。这些检测可能会使得圆要被继续分成一些八分之一部分并对这些部分进行检测。然后，通过并行地求解圆和边的方程组，我们可以解析地求得圆和边的交点。最后，扫描转换所计算的圆弧。在此，对算法要进行适当的初始化，所用的起始点和结束点就是所求的交点（进行了适当的取整处理）。如果扫描转换很快，或者圆不是太大，将圆边界上的像素一个一个地相对于裁剪矩形的边界进行检测并裁剪的操作，在写像素之前完成，可能更加有效。在任何情况下，范围检测都是很有用的。如果是对圆进行填充，可以对每个跨段进行裁剪再填充其中的像素，这样，就不必将跨段中的像素相对于裁剪边界进行逐个的检测了，这在3.7节中已讨论过。

为了裁剪椭圆，我们至少要将它分成4个部分进行范围检测，就如同关于圆的处理一样。然后，我们解析地求取椭圆和裁剪矩形的交点，对这些交点进行适当的取整处理，再用适当初始化了的扫描转换算法进行运算就得到了裁剪结果（相关的扫描转换算法将在下一节介绍）。当然，我们也可以在扫描转换过程中进行裁剪。

3.14 多边形裁剪

如图3-46所示，裁剪一个多边形的算法必须处理很多不同的情况。特别值得注意的是图3-46a中的情况，一个凹多边形被裁剪成了两个独立的多边形。总之，裁剪工作是相当复杂的。多边形的每一条边必须相对于裁剪矩形的每一条边进行检测。裁剪过程中，可能要增加新的边，而对于已有的边，要判断是舍弃、保留还是剖分。裁剪一个多边形可能会得到多个多边形。我们需要一种结构很好的方法来处理所有这些情况。

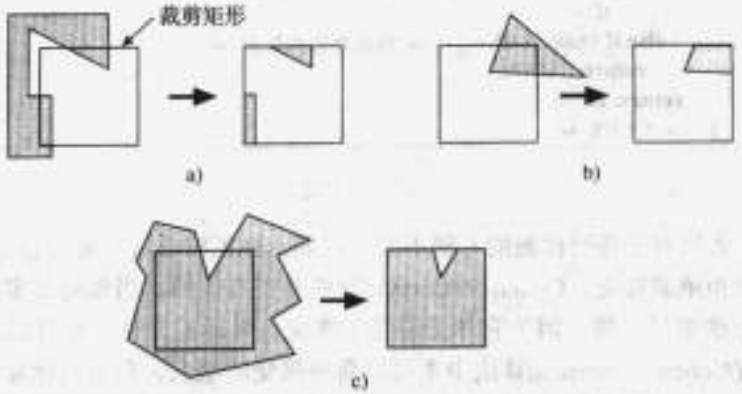


图3-46 多边形裁剪的例子。a) 产生了多个单元，b) 简单的凸多边形情况，c) 有多条外凹边的情况

Sutherland-Hodgman多边形裁剪算法

Sutherland-Hodgman多边形裁剪算法采用了分而治之的策略：它先解决一些简单而明确的问题，然后，综合起来就可以解决全部的问题。这个简单问题就是用一条无穷长的裁剪边来裁剪一个多边形。对于裁剪矩形，就是用其4条边对多边形进行连续的裁剪操作（见图3-47）。

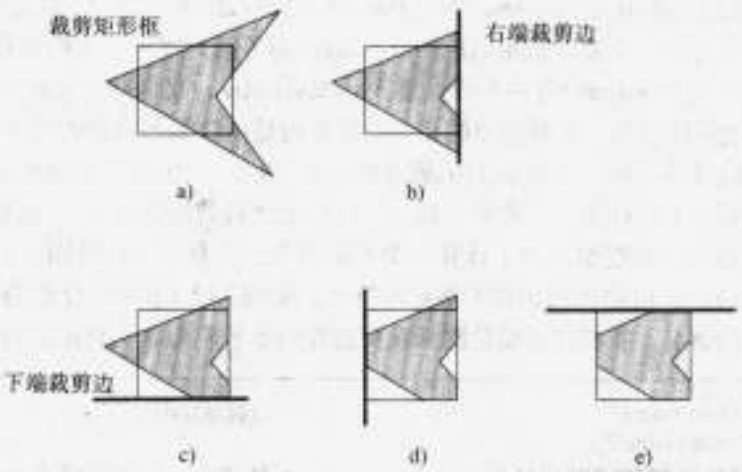


图3-47 多边形裁剪沿着裁剪边依次进行。a) 裁剪前；b) 在右边裁剪；c) 在底边裁剪；d) 在左边裁剪；e) 在顶边裁剪，多边形被裁剪完毕

值得注意的是：裁剪多边形的策略与Cohen-Sutherland裁剪线的算法是不同的。裁剪多边形时，要连续地对4条边进行裁剪，而裁剪线时，是检测外码以判断线段跨越了哪条边，并且只在需要时才进行裁剪。实际上，Sutherland-Hodgman算法的应用范围很广：任何一个凸的或凹的多边形都可以相对于一个凸的裁剪多边形进行裁剪；在三维空间中，多边形可以相对于由平面片构成的多面体进行裁剪。该算法的输入参数是多边形的一串顶点$v_1, v_2, \cdots, v_n$。在二维空间，多边形的边是根据其顶点依次连接生成的，即从v_i到v_{i+1}是一条边，而最后一条边是从v_n到v_1。将多边形相对于一条无穷长的裁剪边进行操作，其输出结果是关于裁剪后的多边形的一串顶点。随后，对刚得到的多边形，相对于第二条裁剪边进行裁剪。如此继续，直至相对于所有裁剪边进行了裁剪操作。

该算法的操作过程是沿着多边形的边从顶点v_n移动到v_1，再顺序移动回到v_n，在每一次移动时，都检测连续的两个顶点与裁剪边的相互关系。在每一步，对于裁剪后的多边形的顶点序列，可能会增加一个顶点或两个顶点，也可能不会增加顶点。此时，要分析4种情况，如图3-48所示。

124
↓
125

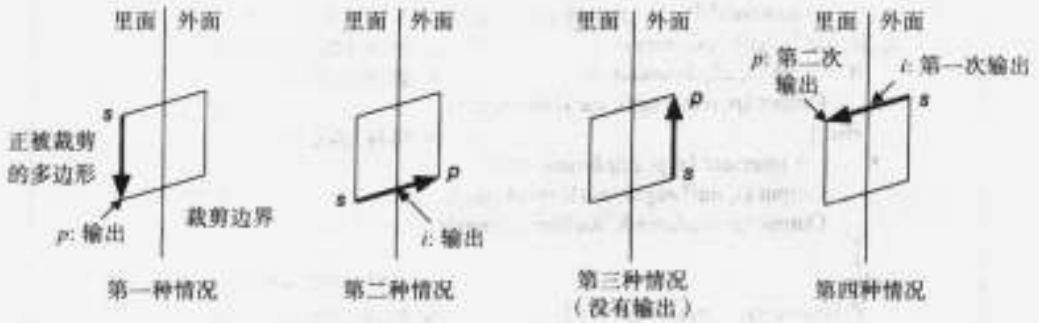


图3-48 多边形裁剪的4种情况

在图3-48中，考察多边形的从顶点s到p的边。假设在上一次循环操作中已经处理了起始点s。在第一种情况时，多边形的边完全在裁剪边的里面，所以，顶点p被加到输出的顶点序列中。在第二种情况时，因为多边形的边与裁剪边界相交，所以将交点i输出。在第三种情况时，因为两个顶点都在边界的外面，所以没有顶点输出。在第四种情况时，交点i和顶点p都加到输出的顶点序列中。

图3-49中的函数SutherlandHodgmanPolygonClip()的输入是顶点的一个数组*inVertexArray*，而其输出的是所产生的另一个顶点数组*outVertexArray*。为了简化程序，我们没有对数组进行越界检测，同时我们用函数Output()将一个顶点放入*outVertexArray*数组中。函数Intersect()计算由顶点*s*和*p*确定的多边形的边与一条裁剪边的交点，裁剪边是由裁剪多边形边界上两个顶点定义的。如果顶点在裁剪边界的里面，函数Inside()就返回true。在此，“里面”的意思是“当从裁剪多边形的一条边上的第一个点往第二个点看去时，顶点位于该裁剪边的左方”。这相当于以逆时针方向沿着裁剪多边形的边界巡游。为了计算一个点是否在一条裁剪边的外面，我们可以计算裁剪边上的法向与多边形的边的点积，并考察它的符号，如同3.12.4节中所介绍的。(对于直立的裁剪矩形这种简单的情况，我们只需要检测到裁剪边界的水平距离或垂直距离的符号。)

```
typedef point vertex;                          /* 点保存双精度x, y */
typedef vertex edge[2];
typedef vertex vertexArray[MAX];               /* MAX 是一个声明的常量 */

static void Output (vertex, int *, vertexArray);
static boolean Inside (vertex, edge);
static vertex Intersect (vertex, vertex, edge);

void SutherlandHodgmanPolygonClip (
      vertexArray inVertexArray,               /* 输入顶点的数组 */
      vertexArray outVertexArray,              /* 输出顶点的数组 */
      int inLength,                            /* inVertexArray数组中的元素数目 */
      int *outLength,                          /* out VertexArray数组中的元素数目 */
      edge clipBoundary)                       /* 裁剪多边形的边 */
{
   vertex s, p,                                /* 当前多边形边的始点和终点 */
   i;                                          /* 与一条裁剪边界的交点 */
   int j;                                      /* 顶点循环的记数器 */

   *outLength = 0;   /* 从inVertexArray中最后的一个顶点开始 */
   s = inVertexArray[inLength - 1];
   for (j = 0; j < inLength; j++) {
      p = inVertexArray[j];   /* 此时的 s 和 p 相应于图3-48中的顶点 */
      if (Inside (p, clipBoundary)) {          /* 第1种和第4种情况 */
         if (Inside (s, clipBoundary))         /* 第1种情况 */
            Output (p, outLength, outVertexArray);
         else {                                /* 第4种情况 */
            i = Intersect (s, p, clipBoundary);
            Output (i, outLength, outVertexArray);
            Output (p, outLength, outVertexArray);
         }
      } else                                   /* 第2种和第3种情况 */
         if (Inside (s, clipBoundary)) {       /* 第2种情况 */
            i = Intersect (s, p, clipBoundary);
            Output (i, outLength, outVertexArray);
         }                                     /* 在第3种情况下，没有操作 */
      s = p;                                   /* 转到下一对顶点继续运行 */
   }  /* for */
}  /* SutherlandHodgmanPolygonClip */
```

图3-49 Sutherland-Hodgman多边形裁剪算法

```
/* 将newVertex加到outVertexArray中，然后修改outLength */
static void Output (vertex newVertex, int *outLength, vertexArray outVertexArray)
{
...
}
/* 检查顶点是在裁剪边的里面还是外面 */
static boolean Inside (vertex testVertex, edge clipBoundary)

{
...
}

/* 相对于clipBoundary（裁剪边界）裁剪多边形的边(first, second)，并输出新的点 */
static vertex Intersect (vertex first, vertex second, edge clipBoundary)
{
...
}
```

图3-49 （续）

Sutherland和Hodgman指出了对算法如何进行组织就能保证算法能够反复操作。一旦输出一个点，算法就用此点来调用自身。再相对于下一条裁剪边界进行裁剪操作。这样，对于已经裁剪了一部分的多边形，就不需要临时的存储空间了：实际上，这个多边形是在裁剪算法的操作“流水线”中穿行。每一步都可以作为一种不需要附加缓冲空间的特殊硬件来实现。这种特点（以及它的普适性）使得这个算法能够适用于当前的硬件实现。但是，在算法执行中可能会在裁剪矩形的边界上引入新的边。考察图3-46a中的情况，连接三角形的左上点和矩形的左上点就引入了一条新的边。在后处理时，要删除这些边，这将在第19章讨论。在19.1节中讨论多边形裁剪多边形的Weiler算法的同时，也将讨论一个基于参数线表示的多边形裁剪算法，该算法是相对于一个直立裁剪矩形进行裁剪的。

3.15 生成字符

3.15.1 定义和裁剪字符

定义字符有两种基本的方法。最一般而又计算开销很大的方法是将每个字符定义为一条曲线或多边形的轮廓，然后在需要时进行扫描转换。在此，我们先讨论另一种较简单的方法：对于给定某种字体的每一个字符，生成一个小型的矩形位图。然后，在产生字符时，只需简单地用copyPixel将字符的图像从一个称为*字体高速缓存*的屏幕外画布中复制到目标位置的帧缓存中。

如下所述，字体高速缓存实际上可以在帧缓存中。对于大多数用一个专用的帧缓存来刷新屏幕的图形系统来说，其帧缓存的空间是大于存储一幅显示图像所要的空间的。例如，一个矩形屏幕上的像素可以存储在一个方形的存储空间中，但该空间中有一条矩形状的区域是在屏幕上“不可见的”。或者，当存储空间比较大，足以容纳两个屏幕时，存储空间就分成两个部分，一部分用来对当前屏幕进行刷新，而另一部分用来写入图像，即对图像进行双缓冲的操作。因为显示控制器的copyPixel在本地图像空间中运行最快，因此，为当前显示字体服务的字体高速缓存往往是存放在这样的不可见的屏幕存储区中。有关这种不可见存储区的一种应用是，在弹出如窗口、菜单或者表格的图像时，对于临时被遮挡的屏幕内容进行保留。

字体高速缓存中存的位图往往是对印刷字体进行各种程度的放大，然后再进行扫描生成

的；那么，字型设计师在需要的时候可以用一个绘图程序对每个字符位图中的各个像素进行修改。或者，字型设计师可以从一开始就用一个绘图程序来为屏幕或低分辨率的打印机专门设计字体。由于小位图在缩放时质量不是太好，因此，对于给定字体的每一个字符要定义多种尺寸大小的位图，以便提供各种大小的标准字体。而且，每个字型有它自己的一套位图。因此，在实际应用时，要为每种字体分配一个单独的字体高速缓存。

在SRGP中，对位图型的字符自动进行裁剪，这是作为copyPixel命令的一部分功能实现的。每个字符是按照逐个像素的方式裁剪到目标矩形中去的，这样我们就可以在字符位图的任何列或行上进行裁剪。若一个系统中的copyPixel命令运行比较慢，一种比较简单但快速的方法是对字符甚至一个字符串进行一种“要么全部接受，要么全部舍弃”的裁剪。只有当字符或字符串的区域可以完全接受的时候，才用copyPixel命令去写字符或字符串。即便系统中的copyPixel命令运行比较快，预先对字符串的范围进行简单的接受/拒绝检测依然是很有用的，这有利于在copyPixel操作中删除一些字符。

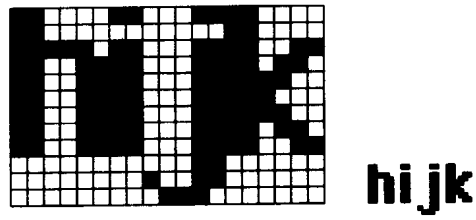


图3-50 一个字体高速缓存例子的一部分

SRGP中是运用简单的位图字体高速缓存技术将字符一个靠一个地存储在画布上。当然这个画布很宽，但它的高度只是最高字符的高度。图3-50中显示了这个高速缓存的一部分以及几个离散的低分辨率字符的例子。每一种被调用的字体都由一个结构（在图3-51中声明的）来描述，该结构包含一个指向存储字符图像的画布的指针、字符的高度信息以及在字符串中相邻字符之间的间距。（在有些软件包中，将字符间距作为字符宽度的一部分保存，由此字符间可以有不同间距）。

```
typedef struct {
    int leftX, width;                    /* 在字体高速缓存中图像的水平位置和宽度 */
} charLocation;

typedef struct {
    canvasID cache;
    int descenderHeight, totalHeight;    /* 高度是常量，宽度是变化的 */
    int interCharacterSpacing;           /* 按照像素的个数进行度量 */
    charLocation locationTable[128];     /* 在文本中解释 */
} fontCacheDescriptor;
```

图3-51 字体高速缓存的类型声明

如2.1.5节中所介绍的，对于一个给定的字体，其下行高度和整个高度是不变的——前者是只为字符的下行部分所用的字体高速缓存中下端的像素行数，而后者则是字体高速缓存画布的高度。但另一方面，字符的宽度并不作为一个常量；这样，字符就能根据自身的大小来占有空间，而不必硬性塞进固定宽度的字符框里。画一个字符串时，SRGP是设置一个固定的字符间距，这个间距是在每种字体的描述器中给出的。用SRGP来显示文本中的各个字符，字处理程序可以显示多行文本，并且可以通过变化字符间距来适当调整各行的情况，以及在标点结束后继续填完各行，以保证各行中最右的字符在右端对齐。这包括用探知文本大小的工具来决定每个单词右边的位置，以便计算下一个单词从何处开始。显然，对于复杂的字处理应用，SRGP的字处理工具是很粗糙的，它不能用于排版程序，因为排版程序要求对单个字母的范围进行更

精细的规划，以便处理一些不能进行水平对齐的情况，如上标、下标以及对文本中的一些字母进行缩放和变化的情况。

3.15.2 一种文本输出图元的实现

在图3-52的程序中，我们揭示了SRGP文本在内部是如何实现的：给定的字符串中的每个字符都是一个一个单独放置的，而字符间距是由字体描述器中适当的项表示的。注意：对于复杂的字符处理，如在字符串中混有多种字体的情况，必须由应用程序来处理。

```
void SRGP_characterText (
        point origin,                          /* 在当前的画布中，在何处放置字母 */
        char *stringToPrint,
        fontCacheDescriptor fontInfo)
{
    int i;

    /* 由应用程序定义的原点为基准，同时不包括下降部分 */
    origin.y -= fontInfo.descenderHeight;

    for (i = 0; i < strlen (stringToPrint); i++) {
        rectangle fontCacheRectangle;
        char charToPrint = stringToPrint[i];
        /* 在高速缓存中寻找字符所在的矩形区域 */
        charLocation *fip = &fontInfo.locationTable[charToPrint];

        fontCacheRectangle.bottomLeft = SRGP_defPoint (fip->leftX, 0);
        fontCacheRectangle.topRight = SRGP_defPoint (fip->leftX + fip->width - 1,
                                                     fontInfo.totalHeight - 1);

        SRGP_copyPixel (fontInfo.cache, fontCacheRectangle, origin);
        /* 修改原点，使它越过刚生成的字符和一个字符间距 */
        origin.x += fip->width + interCharacterSpacing;
    }
} /* SRGP_characterText */
```

图3-52 SRGP文本图元的字符定位的实现

126 ~ 130

我们曾提到，对于显示设备或输出设备的各种不同的分辨率，利用位图技术时其字体、尺寸和字型的每一种组合都要求有一个独立的字体高速缓存。若一种字体有8种不同大小的点阵和4种字型（常规的、加粗的、斜的和加粗兼斜的）就需要32个字体高速缓存！图3-53a中给出一种用一个字体高速缓存来支持多种不同字型的一般方法：斜体字型的模拟可以先将字体图像划分成许多水平状的区域，然后调用一系列的SRGP_copyPixel对各个区域进行偏移。

然而这种粗糙的模拟难以产生令人满意的字符；例如，字母“i”上的点将不会是圆的。对联机交互的用户而言，一个更大的问题是这样处理会扭歪字符间距，使得交互时选取字母非常困难。生成粗体时，我们可以用类似的技巧，就是在水平方向上稍微移动一下将图像复制两次，并用**or**模式合成（见图3-53b）。然而，这些技术同样并不是很令人满意，因为它们生成的字母可能是难以辨认的，特别是在处理上下标这种情况时。

解决存储问题的一个较好的方法是用一种抽象的与设备无关的方式来存储字符，即保存用浮点参数定义的刻画字符轮廓的多边形或曲线，然后对这些轮廓线进行适当的变换，就能得到所需要的字符。一种称为样条的多项式函数可以提供一阶和高阶连续的光滑曲线（见第11章），

因此，它常用来生成文字的轮廓线。尽管定义一个字符所要的空间多于在字体高速缓存中表达该字符所需的空间，但对一个这样所保存的字符进行适当的缩放就可以得到各种大小的字符。同理，对轮廓进行适当的错切变换就可以很快地拟合斜体字型。用完全与设备无关的方式存储字符的另一个重要优点是：对轮廓可以进行任意的平移、旋转、缩放和裁剪（或者将其本身当成一个裁剪区域）。

用样条函数刻画的字符在节省空间方面并不像所期望的那样好。例如，对一个字符而言，并不是其所有尺寸的点都能通过缩放一个抽象的形状来得到，因为字体形状的好看与否与点的大小是有关的，因此，一种字型只对有限大小的一些点是最有效的。再者，对样条曲线的文字进行扫描转换，比简单地用copyPixel命令来实现需要更多的操作，因为，与设备无关的形式必须根据当前的大小、字型和变换属性等来变换到像素坐标系。所以，字体高速缓存技术在微机上应用得很普遍，甚至在一些工作站上也使用。为同时利用这两种方法各自的优点，一种策略是以轮廓的形式存储字体，但在具体应用时，就将字体变换到它们对应的位图。例如，实时地生成字体高速缓存。在19.4节中，我们将详细讨论如何处理样条型文本。

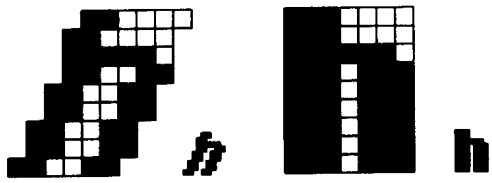


图3-53 为字体产生不同字型的技巧。
a) 斜体，b) 粗体

131

3.16 SRGP_copyPixel

如果只有WritePixel（写像素）和ReadPixel（读像素）这样的底层程序，SRGP_copyPixel函数可以用一个双层嵌套的**for**循环来实现对每个像素的操作。为简便起见，首先假设我们是在二值显示器上工作，并且对于不能字对齐的填写位，不必考虑底层的处理；而在19.6节，我们将讨论一些更接近实际情况的处理方法，它们考虑了硬件存储器的组织结构。在我们简单的SRGP_copyPixel命令的内循环中，我们对源像素和目标像素执行一次ReadPixel操作，然后根据SRGP的写模式对它们进行逻辑组合，最后用WritePixel写出这个结果。对于**replace**这种很常用的写模式，它可以作为一种特例用一个简单的内循环来实现，即只是简单地用ReadPixel读取源像素并用WritePixel写到目标像素，而不必进行逻辑操作。在确定位置的时候，运用裁剪矩形来限制要写的目标像素的区域。

3.17 反走样

3.17.1 增加分辨率

至今所画的图元有一个共同的问题：它们有锯齿形的边。这种称为*锯齿*图或*梯形*图的不好效果，是因为在扫描转换时采取了绝对的“是/否”原则，即对每个像素要么填以图元的颜色，要么就完全不改变。锯齿是“走样”现象的一种情况。减轻或除去走样情况的应用技术，称为*反走样*；运用反走样技术生成的图元或图像被称为是*已反走样*了的。在第14章，我们将从信号处理的角度来讨论一些基本理论，以解释为什么叫走样，走样为什么会发生，以及在生成图像时如何减少或去掉走样。在此，我们只是很直观地解释SRGP的图元为什么会出现走样，并且讨论如何修改本章所介绍的线扫描转换算法以生成已反走样的线。

现在，考察运用中点算法在白色背景上画一条斜率在0和1之间单个像素宽的线。在线穿过的每一列中，该算法都选择最靠近线的一个像素并赋以该线的颜色。每当线所经过的两个相邻列上

的像素不在同一行上时，在画布上所画的线就会出现一次剧烈的跳变，在图3-54a中已清晰地显示出了这一点。对于其他只能给像素赋予两种亮度值的图元，在扫描转换时也会出现同样的情况。

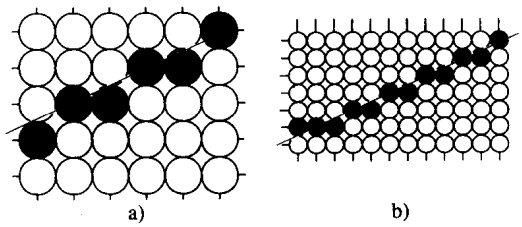


图3-54 a) 二值显示器上的标准的中点线算法，b) 同样的线画在双倍线分辨率的显示器上

假设我们现在用的显示设备在水平和垂直方向上具有双倍的分辨率。如图3-54b所示，这条线穿过了两倍的列数，因此其跳变的次数也翻了一番，但每次跳变的大小，无论在x方向还是y方向上，都减少了一半。尽管这样处理后的图像要好看一些，但这种改进的代价是：空间开销、存储器的带宽以及扫描转换时间都增长到了4倍。提高分辨率的方法是一种开销很大的方法，并且，它只是淡化了锯齿问题，对此问题并没有彻底解决。在下面的几节，我们将探讨一些开销不大的反走样技术，它们依然能生成很好的图像。

3.17.2 未加权的区域采样

改善图像质量的第一种方法是基于以下的认识提出的：尽管理想的图元（如线）是没有宽度的，但我们所画的图元是有宽度的。一个扫描转换的图元在屏幕上占有有限的区域——甚至在一个显示面上最细窄的水平线或垂直线也有一个像素宽，而其他斜率的线的宽度是随着图元的不同而变化的。因此，我们可以将任何线当成一个有一定宽度的矩形，它盖住了一部分栅格，如图3-55所示。然后，根据以下准则进行操作：一条线不应该只在它所经过的每一列上选一个像素来变黑，它应该对它在每一列上所经过的每一个像素赋以一定量的灰度值。（当然，灰度值的这种变化只能在多位像素的显示器上才能反映出来。）这样，对于单个像素宽的线，只有水平线和垂直线才会在它们所属的行或列中每次只涉及一个像素；而对于其他斜率的线，在每一行或每一列中，会有多个像素被赋以各自适当的灰度值。

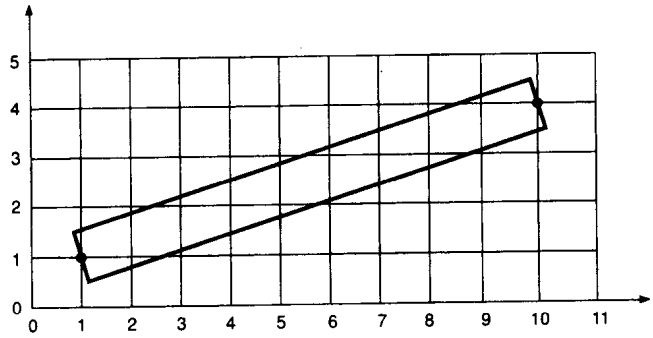


图3-55 宽度不为零的从点(1, 1)到点(10, 4)的线

但是，一个像素的几何特征是怎样的呢？它有多大？对于一条线穿过的一个像素，其灰度值该是多少？一种方便计算的假设是像素是相互间没有重叠的方形的片，可以一片挨着一片地覆盖屏幕，每一片的中心位于栅格点上。（当我们说一个图元盖住了一个像素或像素的一部分时，我们是指它盖住了该像素的整片或一部分；之所以强调这个，是因为我们有时将这样的正方形当作*由像素表现的区域*。）我们还假设一条线赋给一个像素的灰度值决定于该像素的片被线覆盖的区域的大小。在黑白显示的情况下，一个被完全覆盖的像素将被赋以黑色，而被部分覆盖的像素所着的颜色为灰色，其灰度值决定于该像素被线所覆盖的范围。这种技术运用于图3-55中所示的线的情况，如图3-56所示。

132
~
133

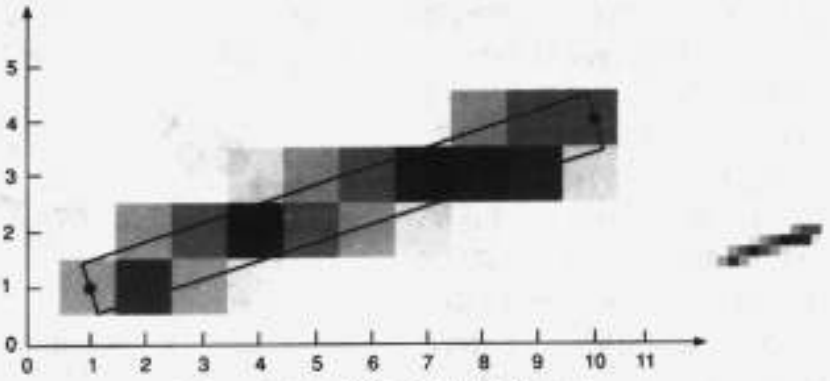


图3-56 与所覆盖的面积成比例的灰度

对于一条在白色背景上所画的黑线，像素（2, 1）上大约70%是黑的，而像素（2, 2）上只有大约25%是黑的。与线没有相交的像素如（2, 3）就完全是白的。根据一个像素被图元覆盖的区域按比例地赋予该像素相应灰度值，将淡化图元边界上刺眼的锯齿特征，得以在完全填充和完全不填充之间很平滑地过渡。这种模糊处理使得一条线从远处看质量很好，尽管它是将填充/不填充的过渡过程扩展到一行或一列上的多个像素。对覆盖区域的一种粗略计算的方法，是将像素划分成更细的矩形的子像素栅格，然后计算表达线的矩形所覆盖的子像素的个数。例如，在线的上端边界和下端边界之间的部分（见习题3.32）。

对这种根据覆盖面积的多少来决定灰度值的方法，我们称之为*未加权的区域采样*。相比于将像素设置成完全填充或者为零的方法，这种方法明显地提高了图像质量。当然，还有一种称为*加权区域采样*的更有效的方法。为解释这两种不同形式的区域采样方法的差异，我们注意到不加权的区域采样有以下3个特点。第一个特点是，与一条线相交的像素的灰度值是随着它的中心与边的距离的增大而降低的：离图元越远，则图元对该像素灰度的影响就越小。这种关系是显然存在的，因为覆盖面积降低，灰度也降低，而当线远离像素的中心而靠近像素的边界时，覆盖面积是递减的。当线完全覆盖了像素时，灰度值将达到一个最大值；而当图元的边只在像素的边界上相切时，覆盖面积为零，因此，灰度值为零。

第二个特点是，如果图元与一个像素不相交，则图元对该像素的灰度根本就没有影响，这
134 也就是说，图元与反映像素的正方形片不相交。第三个特点是，不管像素的中心到覆盖区域的距离如何，相同的面积大小就有相同的灰度值，也就是，只有面积的大小起作用。这样，在像素的角上的一小块面积同靠近像素中心的同样大小的面积所起的作用是一样的。

3.17.3 加权区域采样

在加权区域采样中，我们保留了未加权区域采样的前二个特点（覆盖面积降低，灰度就降低；只有当图元覆盖了像素所表达的区域，图元才能发挥作用），但我们改变了第3个特点。我们让同样面积大小的区域发挥不同大小的作用：靠近像素中心的小块区域的作用大于远离中心的同样大小区域的作用。这种变化的理论基础将在第14章给出，那时，我们将在介绍滤波原理时讨论加权区域采样。

为保留第二个特点，我们必须对像素的几何特征做以下改变。在未加权的区域采样中，对于反映一个像素的正方形片，如果图元的一条边非常靠近它的边界但与边界又不相交，则该图元对该像素的灰度没有什么影响。在新方法中，像素表示一个比正方形片稍大的圆形区域，因此，图元将与这个较大圆形区域相交，并影响像素的灰度值。

为了解释名称中“未加权”和“加权”这两个形容词的缘由，我们定义一个权值函数，它决定图元中给定的一小块区域dA对一个像素的灰度的影响值，它是关于dA到像素中心的距离的一个函数。对于未加权的区域采样，该函数的值是常量；而对于加权的区域采样，距离越远，则函数值越小。将权值函数当成平面上的一个函数$W(x, y)$，则对在（x, y）处的面积dA，其权值就是在(x, y)处的高度。对于未加权的区域采样，当像素表示为正方形贴片时，W的图形就是一个方盒，如图3-57 所示。

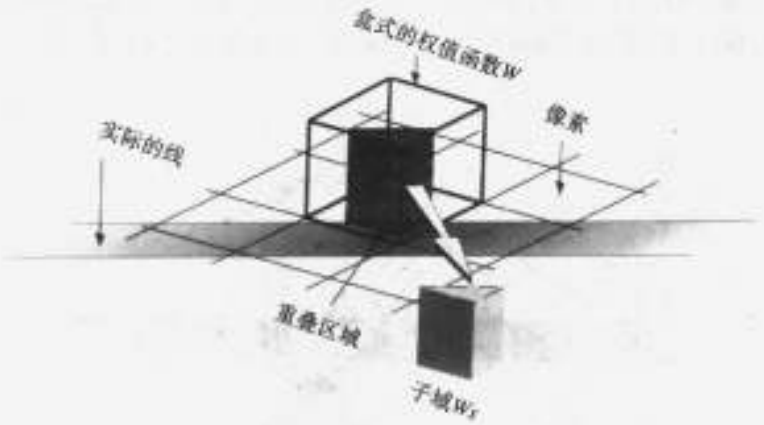


图3-57 针对正方形像素的盒式滤波器

135

图中显示了正方形的像素，其中心是由栅格线的交点表达的；权值函数表示成一个方块，其底就是当前的像素区域。根据图元覆盖像素的面积所决定的灰度是图元与该像素重叠区域的所有小块面积所决定的灰度的总和。每一小块面积所决定的灰度，是与其面积乘以权值的结果成比例的。所以，总的灰度是权值函数在重叠区域上的积分。由此积分W_s表示的体域，总是0和1之间的一个小数，而像素的灰度值I为$I_{max} \cdot W_s$。在图3-57中，W_s是方块中的一个楔形。权值函数也称为*过滤函数*，而此处的方块亦称为*盒式滤波器*。对于未加权的区域采样，方块的高度都会规格化变成1，因此，方块的体积是1，这样，如果一条宽线盖住了整个像素，则该像素的灰度值$I = I_{max} \cdot 1 = I_{max}$。

现在，我们考察怎样为加权区域采样构造一个权值函数；它给远离像素中心的区域的权值一定要小于给靠近像素中心的区域的权值。我们在此挑选的权值函数是最简单的随着距离增大而递减的函数；例如，我们所选择的函数在像素中心有一个最大值，而随着远离中心的距离线性地递减。因为是中心对称的，这个函数的图形就形成了一个圆锥体。锥体的圆形基底（常被称为滤波器的支集）的半径应该比较大；第14章中的滤波理论表明：对此半径的一种较好的选择是整数栅格的单位长度。这样，离像素中心相当远的图元对像素的灰度依然有影响；并且，相邻像素的支集也会有重叠，由此，即使图元的一小部分也可能会对几个不同的像素产生影响（见图3-58）。这种重叠也确保了栅格上不会有不能被像素覆盖的区域。当圆形像素的半径只是栅格的单位距离的一半时，就可能会出现不能被像素覆盖的区域[⊖]。

同关于盒式滤波器的计算一样，为锥形滤波器计算的所有灰度的总和就是在锥顶的下方和锥体的

⊖ 在3.2.1节中已提到，在一个CRT上显示的像素的横截面大致是圆形的，并且相邻的像素一般会有重叠；但是，在加权区域采样中所用的有重叠的圆形模型与这种情况并不直接相关，甚至对显示技术而言也是这样，比如等离子体平板，其中物理像素实际上是没有重叠的正方形片。

基底与图元相交部分的上方之间的体域；体域W_s是锥体中垂直于基底的一个部分，如图3-58所示。像盒式滤波器中一样，锥体的高度首先被规格化，由此，整个锥体下的体积是1；这样，如果一个像素的支集被一个图元完全覆盖，则它可以以最大的灰度值显示。对于图元中远离像素的中心但仍与像素的支集相交的区域，尽管其对像素灰度的贡献很小，但如果像素的中心离线足够近，则它还是能从线获得一定的灰度贡献。相反，在像素被定义为方形几何的模型中，被一条有单位宽度[⊖]的线完全覆盖
136 的方形像素，就不会显示得像它应该的那样亮。加权区域采样的作用其实就是降低相邻像素间的灰度对比，以便像素间的灰度可以平滑过渡。特别是，运用加权区域采样，单位宽度的水平线或垂直线在每一列或每一行中会有1个以上的像素具有亮度。这种情形，在未加权区域采样中不会出现。

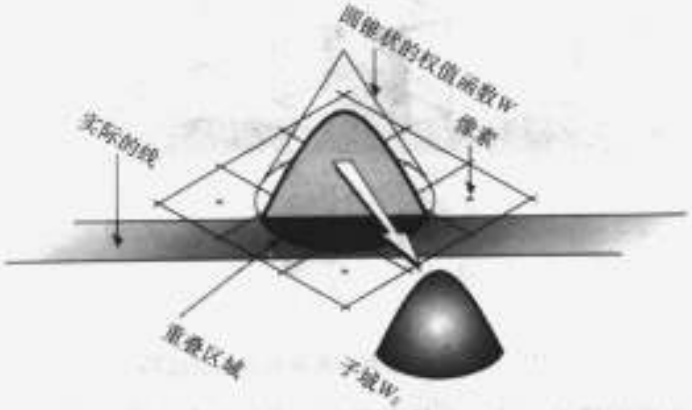


图3-58 针对圆形像素的锥形滤波器，其直径是两个栅格单位

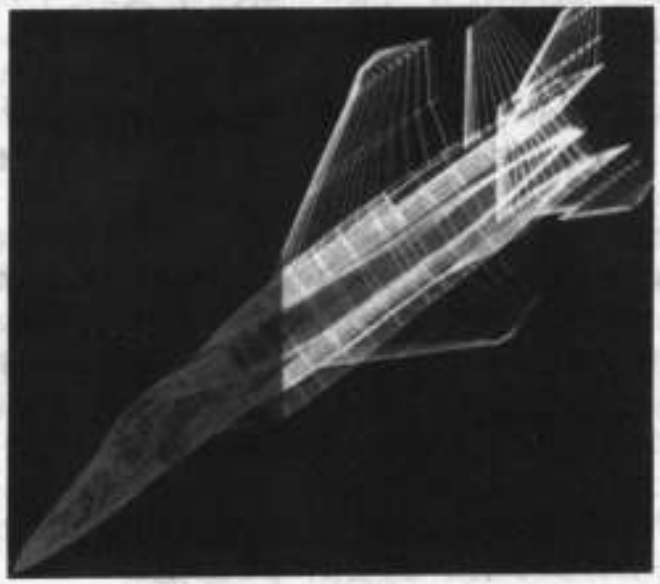


图3-59 画出滤波的线。左半部分是没有滤波的；右半部分是滤波了的。（经数字设备公司Branko Gerovac许可供。）

⊖ 我们现在是说“单位宽度的一条线”，而不是“单个像素宽的一条线”。这是为了明确地表达：在此线宽的一个单位依然是SRGP栅格的单位长，但像素的支集已变大，其直径有两个单位长。

锥形滤波器有两个有用的特点：中心对称，以及函数值随着半径的线性递减性。中心对称是很有用的，它不但使得有关区域的计算与线的倾斜角度无关，而且在理论上是最优的，在第14章中将对此进行介绍。那里，我们还将指出，尽管锥形滤波器要优于盒式滤波器，但它的线性倾斜面（以及它的半径）只是对最优滤波器函数的一种逼近。最优的滤波器在计算上的开销是很大的，而方形滤波器的开销则是最少的，因此，在开销和质量两方面进行比较，圆锥滤波器是一个比较好的在代价与质量间的折中选择。在画线时，滤波和不滤波有很大的不同，这从图3-59即可看出。我们清楚地看到，模糊的线和有颤抖的图案是如何通过滤波来改善质量的。下面，我们将要把锥形滤波器方法应用到扫描转换算法中去。

3.17.4 Gupta-Sproull反走样线扫描算法

本节所讨论的Gupta-Sproull扫描转换线的算法[GUPT81a]预先根据线离像素中心的各种有向距离的情况，计算规格化的滤波器函数的各个相关子域，并将结果存放在一个表中。假设我们所用的像素的半径等于一个栅格单位；也就是说，等于相邻的两个像素中心的间距，这样，斜率小于1的单位宽度的一条线，一般会与一纵列上的三个像素的支集相交，最少两个，最多五个，如图3-60中所示。因为半径为1，每个圆都会有一部分盖住相邻像素的圆。

图3-61中显示了线与像素重叠的几何情况，并据此对滤波器函数Filter(D, t)所建立的表进行查找。在此，D是像素与线中心的距离（与线的倾斜角度无关），t是关于给定宽度的线的一个常量，而函数Filter(D, t)则有赖于滤波器函数的形状。在Gupta和Sproull的文章中为用于4位像素显示器的锥形滤波器给出了一个表；当t = 1时，对范围在0和1.5之间的D的每一次同等增长，该表都记录了滤波器函数Filter(D, t)的各个小数。根据定义，在支集范围以外，也就是$D \geq 1 + 1/2 = 24/16$时，函数值为0。此时，距离的精度只是1/16，因为在4位像素的情况下，灰度值不会大于16。

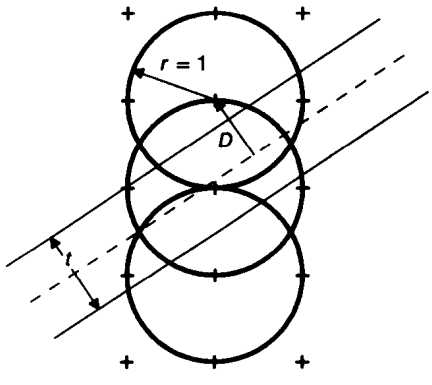


图3-60 一条单位宽的线与3个像素的支集相交

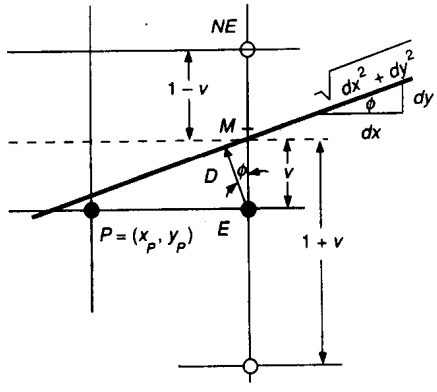


图3-61 中点算法中计算到线的距离

现在，我们可以修改扫描转换线的中点算法。如同前述，我们运用判定变量d来选择像素E或NE，但随后必须对被选择的像素及其垂直方向上的两个相邻的像素赋以适当的灰度值，即根据这些像素到线的距离来计算灰度。图3-61显示了相关的几何特性；运用简单的三角函数，我们可以根据竖直方向距离v算出实际的像素到线的垂直距离D。

我们知道线的斜率是dy/dx，根据图中所示的相似三角形，我们得到

$$D = v\cos\phi = \frac{v\,dx}{\sqrt{dx^2 + dy^2}} \tag{3-2}$$

线上与被选择的像素有相同x坐标的点到被选择的像素之间的垂直距离v就是它们的y坐标

的差值。值得注意的是，这个距离是一个有符号的值。也就是说，如果线从所选择的像素的下方经过，v是负数；如果线从所选择的像素的上方经过，v是正数。因此，在进行滤波函数的计算时，我们必须用该距离的绝对值。所选择的像素也是要赋予灰度的3个像素的中间那个。所选择的像素上方的那个像素到线的竖直距离是$1-v$，而其下方的像素到线的竖直距离是$1+v$。不管线和像素的相对位置如何，你都要核实一下这些距离的有效性，因为距离v是有符号的值。

我们并不直接计算v，而是对$d=F(M)=F(x_p+1, y_p+\frac{1}{2})$运用增量计算来求取。一般地，如果我们知道线上一个点的x坐标，运用3.2.2节所讨论的关系$F(x, y)=2(ax+by+c)=0$，我们就能算出该点的y坐标：

$$y=(ax+c)/-b$$

对于像素E，$x=x_p+1, y=y_p, v=y-y_p$；所以

$$v=((a(x_P+1)+c)/-b)-y_P$$

现在，在等式两边乘以$-b$，并合并同类项，得到

$$-bv=a(x_P+1)+by_P+c=F(x_P+1, y_P)/2$$

但是$b=-dx$。所以，$vdx=F(x_p+1, y_p)/2$。注意，在计算D的式(3-2)中，vdx是分子，而分母是一个可预先计算的常数。在计算vdx时，我们希望根据前面关于$d=F(M)$的计算来递增地计算vdx，并且希望避免除以2的操作，以保持整数运算。因此，对于像素E，

137 ~ 139

$$\begin{aligned}2vdx=F(x_P+1, y_P)&=2a(x_P+1)+2by_P+2c\\&=2a(x_P+1)+2b(y_P+\tfrac{1}{2})-2b/2+2c\\&=d+dx\end{aligned}$$

所以

$$D=\frac{d+dx}{2\sqrt{x^2+y^2}}$$

而常量分母是$1/(2\sqrt{x^2+y^2})$。这样，与y_p+1和y_p-1的像素所相关的分子就可以很容易地得到，它们分别是$2(1-v)dx=2dx-2vdx$和$2(1+v)dx=2dx+2vdx$。

同理，对于像素NE，

$$\begin{aligned}2vdx&=F(x_P+1, y_P+1)=2a(x_P+1)+2b(y_P+\tfrac{1}{2})+2b/2+2c\\&=d-dx\end{aligned}$$

而对于在y_p+2和y_p的像素，其相关的分子依然分别是$2(1-v)dx=2dx-2vdx$和$2(1+v)dx=2dx+2vdx$。

对于3.2.2节中所介绍的中点算法，我们增加了一些操作（其语句前面都加了一个黑点），如图3-62所示。在此，关于E或NE的WritePixel命令被替换成为所选择的像素和其竖直方向上相邻的像素调用IntensifyPixel。IntensifyPixel的作用是通过查找表将距离的绝对值转换到相关的加权区域，得到相对于最大灰度值的一个小数。当然，在实际实现时，这些简单语句是嵌入在程序中的。

对于线的反走样，尽管Gupta-Sproull算法要进行一些小数运算，它依然是一个有效的增量方法。对于线端点的反走样，19.3.5节将讨论用一个不同的查找表来处理。既然查找表的方法可用于与线的边相交的情况，它也可以用来处理任意多边形的边。该方法的缺点是一个查找表只能用于一种宽度的线。19.3.1节还讨论了处理任意线的更一般的技术，它是将线看成间距可以任意宽的两条平行边。我们也能对字符进行反走样处理，即运用滤波的方法（见19.4节），或者简单一些，调用已扫描转换的字符位图并对其边上的像素进行手工的平滑处理。

```
static void IntensifyPixel (int, int, double)

void AntiAliasedLineMidpoint (int x0, int y0, int x1, int y1)
/* 本算法以IntensifyPixel函数的圆形支集上的覆盖区域 */
/* 为自变量，将Gupta-Sproull的灰度表 */
/* 作为相应覆盖区域的函数。注意，在计算 */
/* 16位整数的分母时，因为要做乘方运算，可能会出现溢出。 */
{
    int dx = x1 - x0;
    int dy = y1 - y0;
    int d = 2 * dy - dx;                          /* 如同以前一样，d_start的初始值 */
    int incrE = 2 * dy;                           /* 移动到E时所用的增量 */
    int incrNE = 2 * (dy - dx);                   /* 移动到NE时所用的增量 */
•   int two_v_dx = 0;                             /* 分子；对于起始像素v = 0 */
•   double invDenom = 1.0 /
        (2.0 * sqrt (dx * dx + dy * dy));         /* 预先计算分母的倒数 */
•   double two_dx_invDenom =                      /* 预先计算常量 */
        2.0 * dx * invDenom;
    int x = x0;
    int y = y0;
•   IntensifyPixel (x, y, 0);                     /* 起始像素 */
•   IntensifyPixel (x, y + 1, two_dx_invDenom);   /* 相邻的 */
•   IntensifyPixel (x, y - 1, two_dx_invDenom);   /* 像素 */
    while (x < x1) {
        if (d < 0) {                              /* 选择E */
•           two_v_dx = d + dx;
            d += incrE;
            x++;
        } else {                                  /* 选择NE */
•           two_v_dx = d - dx;
            d += incrNE;
            x++;
            y++;
        }
        /* 现在，设置所选择的像素及与它相邻的像素 */
•       IntensifyPixel (x, y, two_v_dx * invDenom);
•       IntensifyPixel (x, y + 1, two_v_dx_invDenom - two_v_dx * invDenom);
•       IntensifyPixel (x, y - 1, two_v_dx_invDenom + two_v_dx * invDenom);
    }
}  /* AntiAliasedLineMidpoint */

void IntensifyPixel (int x, int y, double distance)
{
    double intensity = Filter (Round (fabs (distance)));
    /* 根据一个整数序号在表中进行查找；宽度为1 */
    WritePixel (x, y, intensity);
}  /* IntensifyPixel */
```

图3-62 Gupta-Sproull的反走样扫描转换线算法

3.18 小结

本章，我们先对基本的裁剪和扫描转换算法进行了讨论，它们是生成光栅图形软件包的基

础。在此，我们只介绍了一些基本的内容；若要高效地实现程序，还必须考虑一些特殊情况，并做一些细致的改进。第19章将讨论一些这方面的问题，以及一些有关任意形状的区域和区域填充的问题。有关位图和像素图操作的另一些算法，将在第17章讨论。在第14章和第19章，对反走样的理论和应用将进行全面的探讨。

[140 ~ 141]

内循环只进行整数运算的增量扫描转换算法，一般是最好的方法。这是本章最重要的思想，因为在交互式光栅图形学中，速度是最根本的要求。这些基本算法可以拓展来处理宽图元的情况，以及运用图案处理边界或填充区域的情况。对图元进行单个像素宽的扫描转换算法，都要尽可能地减少笛卡儿栅格上被选择的像素与定义在平面上实际的图元之间的误差，但那些处理宽图元的算法，基于速度的考虑，往往在质量方面和“正确性”方面要做一些牺牲。到目前为止，尽管许多二维的光栅图形学操作仍使用1位像素的图元，甚至在彩色显示器上也是这样，但我们希望实时反走样的技术能够很快地得到应用和普及。

习题

3.1 编写程序，以扫描转换水平线、垂直线和斜率为 ± 1的线。

3.2 修改扫描转换线的中点算法（图3-8），以便处理任意角度的线。

3.3 说明在扫描转换线的中点算法中，为什么点到线的误差总是小于等于1/2。

3.4 根据3.2.3节中所讨论的内容，修改习题3.2中的扫描转换线的中点算法，以处理端点排序以及与裁剪边相交的情况。

3.5 修改习题3.2中的扫描转换线的中点算法，使得写像素时的灰度是随着线的斜率而变化的。

3.6 修改习题3.2中的扫描转换线的中点算法，以处理不是整数坐标的端点的情况——如果在算法实现时，你用的全部是浮点数，这是非常容易做到的。但是，只用整数来处理端点是有理数的线就是一个比较难的问题。

3.7 对于习题3.2中的扫描转换线的中点算法，分析它能否根据对称性运用判定变量d来从线的两端开始同时向中间画线。如果dx和dy的最大公约数c使得dx/c是偶数而dy/c是奇数（$0<dy<dx$），那么，你的算法能对这种情况进行一致性的处理吗？即不管怎样选择像素，所得的误差是相等的，比如点（0, 0）到点（24, 9）的线。它能否处理下面这种情况呢？即dx是$2dy$的整数倍，比如点（0, 0）到点（16, 4）之间的线？（由J. Bresenham提供。）

3.8 说明在什么情况下折线可以不只共享顶点像素。对此，给出一个不必对像素写两遍的算法。提示：将扫描转换和以**xor**模式写画布的操作分解成独立的过程。

3.9 扩展图3-21中扫描转换椭圆的中点算法的伪代码，使它能正确地检测各种可能出现的情况。

3.10 运用3.3.2节中讨论圆时所用的前差分技术来开发用于标准椭圆的扫描转换的2阶前差分技术。写出实现该项技术的代码。

3.11 开发一种不同于3.3.2节中扫描转换圆的中点算法的算法，以便用折线来对圆进行分段的线性逼近并以此来扫描转换圆。

[142] 3.12 设计一个算法，以扫描转换空心的圆角矩形，其角都是给定半径的四分圆弧。

3.13 写一个扫描转换程序，可以在屏幕上的任意位置填实直立的矩形，并且能高效地写二值帧缓存，一次要写一个字的像素。

3.14 运用3.6节的规则构造一些例子，使得一些像素被“遗失”或被写许多次。试提出另一套

可能更复杂的规则，使得共享边上的共享像素不会画两次，也不会导致像素的漏画。这些规则与所增加的开销相比，是否值得？

3.15 实现3.6节中有关多边形扫描转换的伪代码，在跨度的记录中要考虑可能出现的狭长多边形。

3.16 利用三角形和梯形简单属性给出扫描转换它们的算法。在硬件中，这样的算法是很多的。

3.17 分析可以将任意的（可能是凹的或自交的）多边形分解成顶点共享的三角形网格的三角剖分方法。这是否有助于限制多边形的形状变化，使得多边形最坏就是凹的，但绝不会有自交情况和内部空洞情况的出现？（也可参见[PREP85]。）

3.18 运用跨度表对扫描转换圆的中点算法（图3-16）进行扩展，以便能填充圆和圆状锲形（如圆饼状的图）。

3.19 运用跨度表对扫描转换椭圆的中点算法（图3-21）进行扩展，以便能填充椭圆锲形。

3.20 对于多边形的图案填充，实现绝对定位和相对定位两个算法，这在3.9节中已讨论过。然后，比较它们在视觉效果和计算效率方面的差异。

3.21 运用图3-30中所示的技术，以不透明的方式对字符进行图案填充。对于这种问题的处理，讨论如何充分利用附带写掩码的copyPixel命令。

3.22 实现一种可以画各种符号的技术，比如用小型位图表示的光标图标，使得它们无论写到什么样的背景上都能看得见。提示：为每一种符号定义一个“包含”它的掩码（也就是，掩码要比符号能覆盖更多的像素），然后，分别画掩码和符号。

3.23 用3.9节中介绍的技术实现画宽线的算法。比较它们所得结果的质量和效率。

3.24 扩展扫描转换圆的中点算法（图3-16），以处理宽的圆。

3.25 实现一个可以提供线型以及笔型和图案的画宽线的算法。

3.26 运用每次处理后继第i个像素的快速扫描再回朔的技术，将截剪功能作为线和非填充多边形进行扫描转换的一部分实现。将该技术用于填充的宽线，以及填充的多边形。对这些图元，比较进行实时裁剪的效率和对它们进行解析裁剪的效率。

3.27 将截剪作为扫描转换非填充和填充的圆和椭圆的一部分实现。对这些图元，比较对它们进行这种实时裁剪和解析裁剪的可行性和效率。

3.28 修改图3-41中的Cohen-Sutherland线裁剪算法，以避免在连续的裁剪操作中反复计算斜率。

3.29 考察几种典型及非典型的情况，并用指令计数，以比较Sutherland-Cohen算法和Cyrus-Beck算法的效率。对于水平线和垂直线，能否进行最优的处理？

3.30 考虑一个有n个顶点的凸多边形被一个裁剪矩形裁剪的情况。裁剪后的多边形，顶点数最多是多少？最少是多少？对凹多边形，也考虑同样的问题。所得结果是多少个多边形？如果只得到一个多边形，它最大的顶点数可能是多少？ 143

3.31 解释为什么Sutherland-Hodgman多边形裁剪算法只对凸裁剪区域有效。

3.32 设计一个方法将像素进行细分，并计数被线覆盖（至少要有比较明显的部分被覆盖）的子像素个数，并以此作为采用未加权区域采样的画线算法的一部分。

3.33 基于像素中心到线中心的距离，为各种递减函数建立灰度变化表。将这些表运用于图3-62所示的关于线的反走样算法中。对这些结果与用盒式滤波器生成线的方法所得的结果进行比较。

3.34 扩展处理线的反走样技术，以处理多边形的走样问题。考虑怎样处理曲线型图元和字符的非多边形形状的边界？ 144

第4章 图形硬件

本章将描述计算机图形显示系统中主要硬件的工作原理。4.1节覆盖了硬拷贝设备：打印机、笔式绘图仪、静电绘图仪、激光打印机、喷墨打印机、热传导绘图仪和胶片记录器。简要地叙述了每一类设备的基本技术概念，然后用一个结论小节比较了这些不同的设备。4.2节是有关显示设备的，讨论了单色和彩色荫罩式阴极射线管（CRT）、直视存储管（DVST）、液晶显示器（LCD）、等离子平板、电致发光显示器以及其他更多的一些特定的技术，又用一个结论小节讨论了不同显示设备的优点和缺点。

光栅显示系统可以使用这里讨论的任何一种显示技术，它将在4.3节讨论。我们先介绍一个简单直接的光栅系统，然后再讨论跟图形功能有关的方面以及光栅处理器和通用处理器如何集成到系统地址空间。4.4节讨论了在图像显示、颜色控制、动画和图像合成中要用到的查找表和视频控制器的功能。4.5节简要讨论几乎已经过时了的向量（也叫随机、书法、笔划）显示系统。接下来的4.6节讨论用户交互设备，如手写板、鼠标、触摸板等等。除了技术细节以外，对运作原理也进行了叙述。4.7节简要讨论了图像输入设备，如图像扫描仪，利用图像输入设备可以把现有的图像输入到计算机中去。

图4-1显示了这些硬件设备之间的关系。关键元素是集成的CPU和显示处理器，通常被称为图形

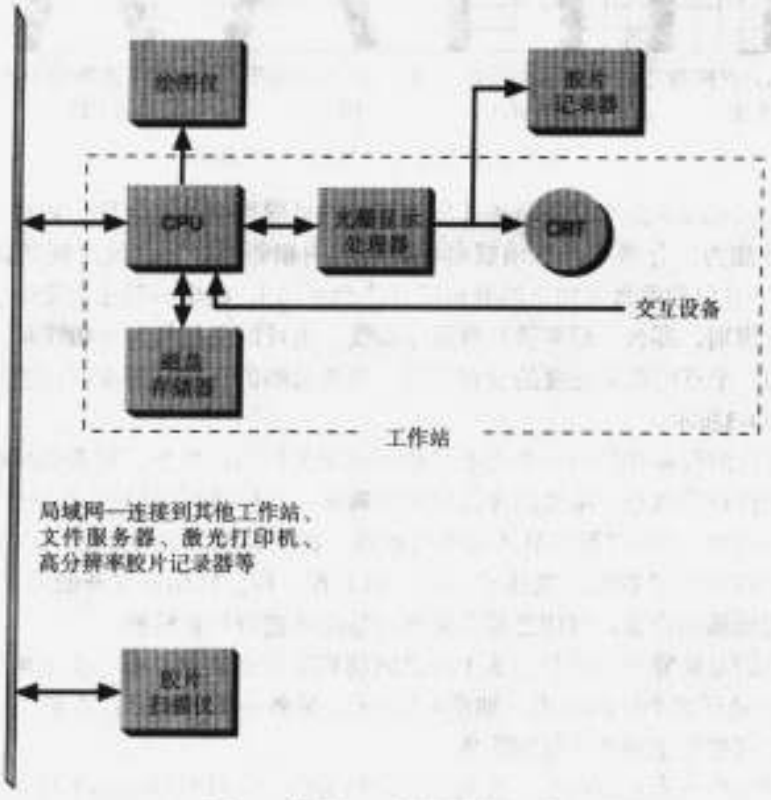


图4-1 典型交互式图形系统组成

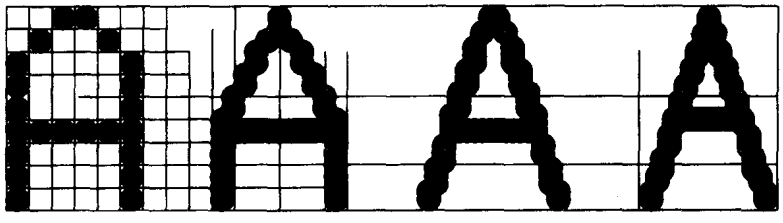
工作站，一般由一个每秒至少执行几百万条指令的CPU、一个大硬盘以及一个分辨率至少为1000×800的显示器组成。局域网把多个工作站连接起来，这样可以共享文件、使用电子邮件，并且可以使用其他的共享外设，如高品质的绘图仪、大硬盘还有连接其他网络的网关和更高性能的计算机。

4.1 硬拷贝技术

在这一节中我们将要讨论各种硬拷贝技术，然后概括它们的特点。首先要定义几个重要的术语。

显示设备所能获得的图像品质与设备的寻址能力（addressability）以及点尺寸（dot size，也称为 spot size）有关。*点尺寸*是设备输出上一个点的直径。*寻址能力*是指每英寸所能创建的点的数目（不要求可辨别），水平方向和垂直方向上的寻址能力可能不同。在x处的寻址能力是（x, y）坐标和（$x+1$, y）坐标中心点距离的倒数，在y处的寻址能力也类似定义。*点间距离*
146 （interdot distance）是寻址能力的倒数。

我们总是希望点尺寸比点间距离大一些，这样可以获得平滑的形状。图4-2演示了其原理。这里进行了权衡：点尺寸是点间距离的几倍时，可以得到很平滑的打印形状，而较小的点尺寸则可以得到更好的细部特征。



a) 点距等于点大小　b) 点距等于点大小的1/2　c) 点距等于点大小的1/3　d) 点距等于点大小的1/4

图4-2 点距与点大小各种比率的效果

分辨率（resolution）是指设备每英寸所能创建的可辨别线条的数目，它和点尺寸有关，小于或者等于寻址能力。分辨率可以用观察者能辨别的相邻黑线和白线之间的最小距离来定义（这又意味着水平方向和垂直方向上的分辨率可能会不同），如果一英寸上交织着的40条黑线和40条白线能够被辨别，那么分辨率就是每英寸80线，也可以说是每英寸40线对。

分辨率也和一个点的截面亮度的分布有关，边界清晰的点的分辨率要比边界不清晰的点的分辨率高，如图4-3所示。

很多将要讨论的设备在任何一个点上只能创建很少的几种颜色，更多的颜色可以通过抖动模式来得到，其代价是减小了结果图像的空间分辨率。我们将在第13章讨论这个问题。

*点阵打印机*利用一个有7到24针（细小的硬线）的打印头，每颗针都可以单独地触发，对着纸击打色带。打印头沿着纸每次移动一趟，纸上卷一行，打印头又开始另一趟打印，因此，这种打印机是光栅输出设备，打印之前需要将向量图像进行扫描转换。

点阵打印机的寻址能力不受打印头上针之间物理距离的限制，打印头上可以有两列针，它们在垂直方向上错开半个针间距离，如图4-4所示，另外一种方法是：在纸上打印两趟，在第一趟和第二趟之间把纸上卷半个针间距离。

彩色色带可以生成彩色硬拷贝。有两种可能的方法，一种是使用多个打印头，每个打印头使用一种颜色色带，另外一种也是更普遍的方法是使用一个打印头、多种颜色的色带。

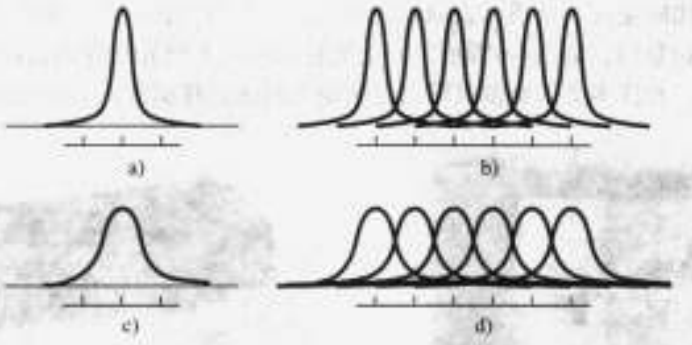


图4-3 点的截面亮度对分辨率的影响。a) 一个具有明显边沿的点；b) 几个重叠起来的a) 中的点；c) 比a) 中点宽但高度稍低的点，因为能量分布在一个较大的区域中，它的边沿不明显，其高度小于a) 中的点；d) 几个重叠的c) 中的点。b) 中尖峰之间的区别要比d) 中的明显。实际图像的亮度是每一个点亮度的总和

比实际色带上颜色更多的颜色可以通过在纸上同一点重复打印两种不同的颜色来得到。在上层的颜色会比下层的颜色要强一些，在任一点上用三种颜色——通常是青色、紫色和黄色，重复打印后可以得到多达8种的颜色，但是，通过打印三种颜色而得到的黑色很模糊，所以经常在色带上加入真正的黑色。

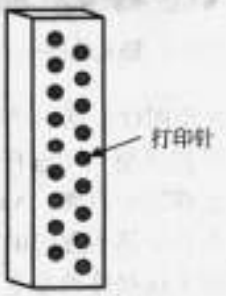


图4-4 具有两列打印针的点阵式打印头，两列打印针在垂直方向上错开半个针距以增加分辨率

就像有随机显示器和光栅显示器一样，也有随机绘图仪和光栅绘图仪。笔式绘图仪（pen plotter）在一张纸上以随机的、向量的方式移动笔，绘制直线的时候，笔定位在直线的起点，然后放下到纸上，沿着一条笔直的路径移向直线的终点，抬起，又移动到下一条直线的起点。有两种基本的笔式绘图仪，桌面平板绘图仪在纸上沿x方向和y方向移动，纸是平铺在桌面上，由静电吸引力、真空吸引力或者就是简单地拉紧固定（图4-5）。一个机械车架在桌面上沿纵向移动，在车架上有一支笔沿车架横向移动，笔可以抬起和放下。平板绘图仪尺寸大小从12英寸×18英寸到6英尺×10英尺或者更大。在某些情况下，笔可能是一个使照相底片曝光的光源，或者划刻的刀刃，一般情况下，使用的是多种颜色，多种宽度的笔。

147
↓
148

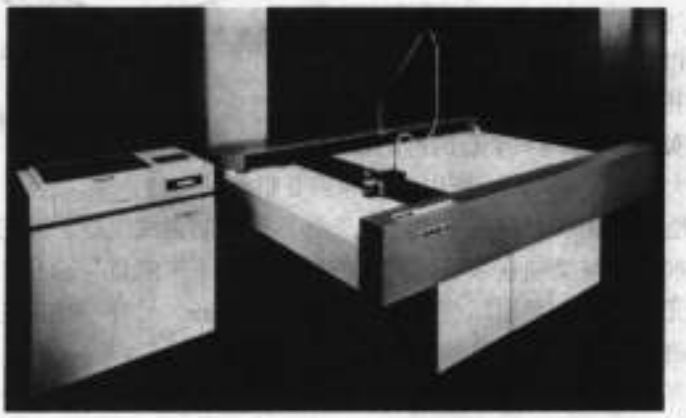


图4-5 平板绘图仪（由Calcomp-California计算机产品公司提供）

和上面形成对比的是鼓式绘图仪，鼓式绘图仪沿一个方向移动纸，沿另外一个方向移动笔，通常，纸是在鼓上拉紧的，如图4-6所示，鼓上的针扣着纸上预先打好的孔以防止纸滑动。鼓可以向前和向后转动，而许多桌面绘图仪可以在滚筒之间前后移动纸，笔则在纸上移动（图4-7）。

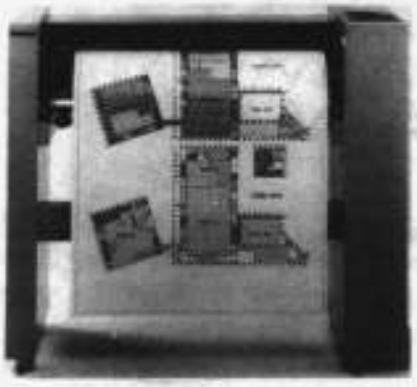


图4-6 鼓式绘图仪（惠普公司提供）



图4-7 桌面绘图仪（惠普公司提供）

笔式绘图仪包括一个微处理器，它可以接受如“绘制直线”、“移动”、“绘制圆”、“绘制文字”、“设置线型”、“选择笔”等类似的命令。（微处理器有时候也对笔的移动进行优化，以减少笔抬起移动的距离。Anderson[ANDE83]为此开发了一种有效的算法。）微处理器把输出图元分解成在八个基本方向中任何一个方向上笔移动的增量。由位置传感器和伺服马达组成的反馈系统实现了这些移动命令，并且有一个电磁铁用于抬起和放下笔，绘图的速度跟笔移动的速度和加速度有关。笔的加速度是笔头质量的函数，所以许多多笔绘图仪把除了当前活动的笔之外的其他笔都放在一边，以减少这个质量。

与笔式绘图仪相对比，静电绘图仪把负电加在白纸上要画成黑色的地方，然后纸从加了正电的黑色调色剂下面通过（图4-8），调色剂微粒就附着在纸上带了负电的地方，在现有的系统里，可以每次一行地给宽达72英寸的纸加电。纸移动的速度可以达到每秒3英寸，纸上的每一英寸上可以水平分布100到400个电触点。每个电触点的状态是开（加上负电）或者关（不加电）。每个在电触点上的点是黑色或者白色的，灰度级必须用抖动模式来获得。

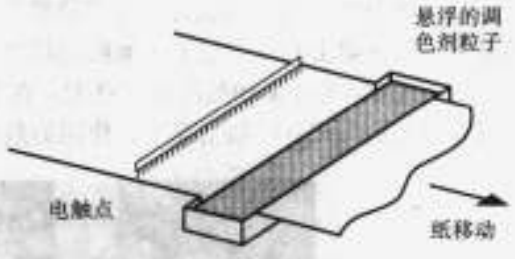


图4-8 静电绘图仪组成

静电绘图仪可以包含扫描转换的功能，也可以由CPU来做扫描转换的工作。在后一种情况下，因为每平方英寸400×400个点的静电绘图仪的信息密度很大（见习题4.1），所以相应地需要很高的传输速率。

一些彩色静电绘图仪在纸上进行多趟绘制，每一趟绘制完成之后，又回卷到图的开始，第一趟时，黑色的校准标记被绘制在靠近纸边缘的地方，接下来的每一趟分别用黑色、青色、紫色和黄色调色剂来完成绘图，利用那些校准标记来保持对齐。另外一些彩色静电绘图仪则是利用多个绘制头在一趟中完成所有颜色的绘制。

静电绘图仪通常比笔式绘图仪快，也是高品质打印机速度的两倍。另一方面，笔式绘图仪绘制出来的图像的对比度要高于静电绘图仪的，因为静电绘图仪甚至会在没有加上负电的地方

沉积调色剂。

激光打印机用一束激光扫描带正电的、其表面覆盖着一层硒的旋转鼓，被激光束击中的区域就失去正电，只有那些将要拷贝成黑色的地方才保留着正电，带负电的调色剂粘着到硒鼓上带正电的区域，然后转移到空白的纸上形成拷贝。在彩色激光打印机中，这个过程要重复三次，每次打印一种主色。图4-9是单色激光打印机的部分结构示意图。



图4-9 激光打印机组成（色剂应用机制和送纸系统未表示）

和静电绘图仪类似，硒鼓上的任一个点要么带正电，要么不带电，在拷贝上相应的点要么就是黑色要么不是黑色，因此激光打印机是两级的单色设备或者八色的彩色设备。 149 - 151

激光打印机有一个微处理器来做扫描转换和控制打印机，越来越多的激光打印机把PostScript文档和图像描述语言作为事实上的标准[ADOB85a]。PostScript提供了要打印的图像的过程描述，PostScript也可以用于存储图像描述(第19章)。大部分激光打印机使用8.5英寸×11英寸大小或者8.5英寸×14英寸大小的纸。但在工程绘制和地图绘制应用中也会用到相当宽（30英寸）的打印机。

喷墨打印机把青色、紫色、黄色，有时候还有黑色的墨水喷洒在纸上。在大多数情况下，墨水喷射器是安装在一个类似打印机结构的头上，打印头移动绘制一条扫描线，然后回车，纸往前走一个扫描线间距，接着绘制下一条扫描线。如果走纸太多或者太少，会做一些相应的细微调整。另一个方法就是把纸卷在一个鼓上，鼓快速地旋转，打印头则缓慢地沿鼓移动。图4-10演示了这种方法。在两种情况下，所有的颜色都是同时沉积，而不是像多趟激光打印机和静电绘图仪那样分别沉积。多数喷墨打印机对每个像素的控制也只是限于开和关（就是二值）。一些则具有变点大小的功能。

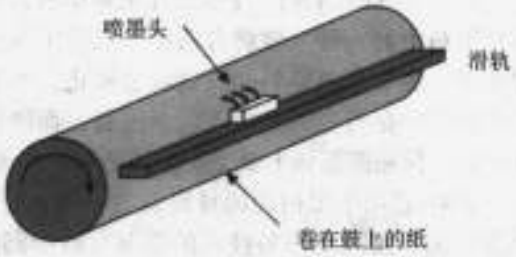


图4-10 旋转喷墨绘图仪

一些喷墨打印机既可以接受数字图像输入，也可以接受视频输入，这就使它们很有吸引力，因为它们可以创建光栅显示系统的硬拷贝图像。注意，结果图像的分辨率要受视频输入的分辨率的限制——通常从640×480到1280×1024，喷墨打印机比其他种类的打印机需要更多的维护。

热传导打印机是另外一种光栅硬拷贝设备。它是静电绘图仪的前身，均匀细密分布（通常是每英寸200颗）的加热笔尖把彩色蜡纸上的彩色粉剂传输到白纸上，加热笔尖带同时在蜡纸和白纸上绘制。笔尖有选择地被加热使得色剂能够传输。对彩色打印（这种技术最通常的用途），蜡纸是在一个交替着青色、紫色、黄色和黑色条带的卷轴上，每一条带的长度都和纸张的大小相等。因为笔尖的加热和冷却非常迅速，一张彩色图像的硬拷贝可以在一分钟内完成。一些热传导打印机能同时接受视频信号输入和数字位图输入，这使得创建视频图像的硬拷贝非常方便。 152

热升华印染传导打印机和热传导打印机的工作原理类似，但是加热和印染传导过程可允许有256种不同强度的青色、紫色、黄色，可以创建出空间分辨率为每英寸200点的高品质全彩色图像，它的打印过程比蜡传输要慢，但是其质量可以和照片接近。

拍摄阴极射线（电视）管（CRT）上显示的图像的照相机可以看做是一种硬拷贝设备。这是我们所讨论的可以在单一分辨率点上得到许多种颜色的最普通的硬拷贝技术。胶片可以记录很多种不同的颜色。

有两种彩色胶片记录器的基本技术。一种是：照相机直接拍摄彩色CRT上显示的彩色图像。因为彩色CRT的荫罩板的原因（参见4.2.2节），图像的分辨率是受限制的，而且彩色显示器使用的必须是光栅扫描技术。另一种是：透过彩色滤镜拍摄黑白CRT，图像中不同颜色的元素按次序显示（图4-11）。这种技术可以得到很高质量的光栅或者向量图像。通过两个或者多个滤镜两次曝光图像的一部分以形成混合的颜色，通常所用的CRT的亮度也不同。

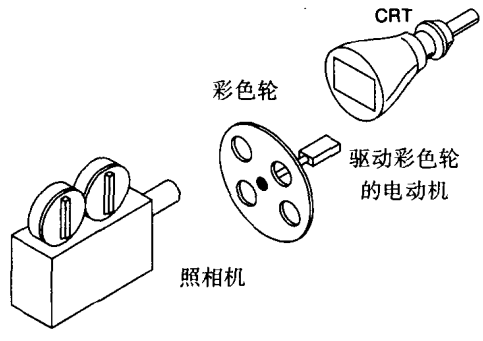


图4-11 使用彩色滤镜拍摄彩色照片的胶片记录器

胶片记录器的输入可以是光栅视频信号、位图或向量指令。视频信号可以直接驱动彩色CRT，也可以将信号中的红绿蓝分量分离出来按时间顺序透过滤镜显示。这两种情况 个摄制周期视频信号都必须保持恒定，如果使用的是较慢速度（低灵敏度）的胶片的话这个过程可能持续达1分钟。高速高分辨率的位图或者向量系统非常昂贵，因为驱动电子束和CRT本身必须精密地设计和校准。如果速度和分辨率降低的话，价格就会有极大的下降。

最近开发的Cycolor技术是在纸张中嵌入数百万的微胶囊，这些胶囊中装有青、紫、黄三种
153 颜色染料中的一种。胶囊有选择地在特定颜色的光源下曝光时会硬化。例如，当在绿色光源下曝光时，装有紫色染料的胶囊就会硬化。当纸通过起挤压作用的压轮时，没有硬化的胶囊（在这个例子中是青色和黄色）就会破裂，而硬化了的胶囊（紫色）就不会破裂。青色和黄色混合在一起，传输到纸张上就是高品质的绿色图像了。与其他技术不同，这种技术只需要一趟走纸。

表4-1总结了黑白硬拷贝设备的不同之处，表4-2总结了大部分彩色硬拷贝设备的不同之处。更多的关于硬拷贝设备技术的细节可以参阅[DURB88]。当然现在技术创新非常之迅速，某些设备的相对优点和缺点都会改变。某些设备其价格和性能变化范围也是很大。例如，胶片记录器和笔式绘图仪的价格从1000美元到100 000美元都有。

表4-1 几种单色硬拷贝技术的比较

	笔式绘图仪	点阵	静电	激光	喷墨	热传导	照片
每点亮度等级	2	2	2	2	2至更多	2	很多
寻址能力(点/英寸)	1000+	到250	到400	到1500	到200	到200	到800
点大小(千分之一英寸)	6~15	10~18	8	5	8~20	7~10	6~20
相对价格范围	L~M	VL~L	M	M~H	L~M	L~M	L~H
每图像相对价格	L	VL	M	M	L	M	H
图像质量	L~M	L~M	M	H	M	M	H
速度	L	M	H~H	M~H	M	M	L

注：VL=很低，L=低，M=中，H=高。

表4-2 几种彩色硬拷贝技术的比较

	笔式绘图仪	点阵	静电	激光	喷墨	热传导	照片
每点色彩等级	到16	8	8	8	8至更多	8至更多	很多
寻址能力(点/英寸)	1000+	到250	到400	到1500	到200	到200	到800
点大小(千分之一英寸)	15~6	18~10	8	5	20~8	10~7	20~6
相对价格范围	L~M	VL	M~H	M~H	L~M	M	M~H
每图像相对价格	L	VL	M	M	L	M	H
图像质量	L~M	L	M	H	M	M~H	M~H
速度	L	L~M	M	M	M	L~M	L

注：VL=很低，L=低，M=中，H=高。

注意，在所有这些彩色设备中，只有胶片记录器、Cycolor以及一些喷墨打印机可以记录很宽范围的颜色，其他的技术都是对三种或者四种它们能够直接记录的颜色使用二元的开关控制。还要注意到这种颜色控制是需要技巧的，不能保证一个设备上的八种颜色和显示器上的一样，也不能保证它和另外一个硬拷贝设备上的一样（参见第13章）。

现在已经有了很宽的激光打印机，它们在逐渐取代黑白对比度较差且很难维护的静电绘图仪。 154

4.2 显示技术

交互式计算机图形学需要可以迅速改变图像的显示设备。非永久性显示允许改变图像，使得动态移动图像的某部分成为可能。到目前为止，CRT是最普遍使用的显示设备而且还要普遍使用很多年。但是，固态技术的发展从长久来看将会充分削弱CRT的统治地位。

用于图形显示的单色CRT和家用黑白电视机中所用的没有什么区别，图4-12显示了具有代表性的CRT剖面视图，由电子枪发出的电子流（阴极射线）受CRT封装内靠近射线管表面的高正电压的加速，射向涂覆荧（磷）光层的屏幕，在射向屏幕的过程中，电子束通过聚焦系统的控制汇聚成极细的一束，然后受偏转线圈产生的磁场控制射向屏幕上的特定位置，当电子击中屏幕时，荧光物质就会发射出可见光。因为荧光物质的发光输出随时间指数递减，整个图像必须每秒钟刷新（重绘）许多次，观测者看到的才是稳定的不闪烁的图像。

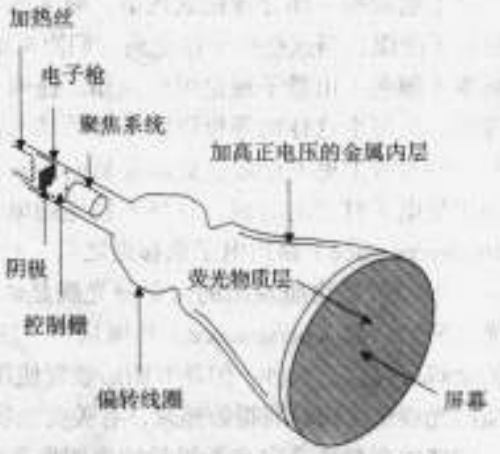


图4-12 CRT的剖面图（未按比例绘制）

光栅扫描显示系统的刷新速率通常至少要每秒60帧，这和图像的复杂度是无关的。向量显示系统的刷新速率则直接和图像复杂度有关（直线、点和字符的数目），图像复杂度越大，每个刷新周期所花的时间就越多，刷新速率就越低。

从加热的阴极射出的电子束被一个通常15 000伏到20 000伏的高电压加速，这个电压决定电子击中荧光物质之前的速度。控制栅电压决定电子束中实际有多少电子，这个负的控制栅电压越大，穿过栅的电子越少。这样就可以控制某个点的亮度，因为荧光物质的可见光随着电子束中电子数量的减少而减弱。 155

聚焦系统聚集电子束使得电子束击中荧光物质的时候会聚在一个小点上。仅仅让电子束中的电子平行移动是不够的，因为电子之间的相互排斥会使它们发散开，所以需要聚焦系统来汇聚它们以防止它们分散。除了电子束的这种分散的趋势以外，聚焦电子束和聚焦光线是类似的。光学透镜和电子透镜都有它们的焦距，在CRT中，设置的焦距要使得电子汇聚到荧光屏上。电子束中截面电子密度遵循高斯分布（正态分布），荧光屏上形成的光斑其亮度也遵循相同的分布，如图4-3所示。因此光斑没有明显的边界，光斑的大小尺寸则通常是指光强是光斑中心光强50%的环的直径。一般在高分辨率单色显示器上光斑的大小为0.005英寸。

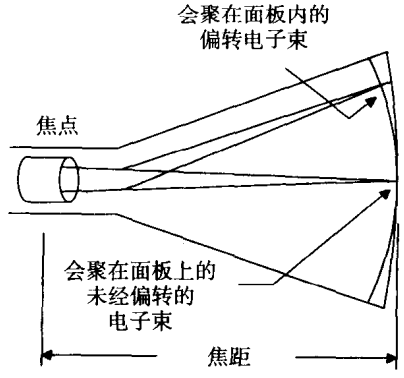


图4-13 电子束的聚焦。聚焦距离作为折射角的函数

图4-13演示了CRT的聚焦系统和CRT的困难之处。电子束在两个地方显示，在一种情况下，电子束在击打荧光屏的地方汇聚，而在第二种情况下汇聚点在荧光屏的后面，因此结果图像会有些模糊。这种情况为什么会发生呢？大多数CRT的表面都几乎是平的，因此有一个远比透镜到荧光屏之间距离大得多的曲面半径，所以，不是所有荧光屏上的点到透镜的距离都相等，如果电子束在荧光屏中心聚焦的话，它就不能在荧光屏上的其他点上聚焦。电子束被折射得离荧光屏中心越远，电子束散射就越厉害。在高精密的显示系统里，使用电子束位置的一个函数来动态地调整透镜，从而解决了这个问题。使荧光屏的表面成为一个突出的曲面不是一个好的解决方法。

当电子束击中覆盖着一层荧光物质的CRT屏幕时，单个电子移动所带的动能和加速电压成正
156 比，这个能量的一部分转化成热量，剩下的则传递给荧光物质原子中的电子，使它们跳跃到较高的量子能级，当这些电子恢复到它们原来的量子等级时，就会以光的形式释放出多余的能量，其频率（颜色）由量子理论可以预知。任何一种荧光物质都有几种不同的电子可以跳跃到的量子等级，从每个这样的等级返回未激活状态都对应着一种颜色。另外，电子在一些等级上要比在另一些等级上更不稳定，更加容易返回未激活状态。荧光物质的*荧光*（fluorescence）是指荧光物质受电子打击的时候，这些不稳定的电子丢失它们多余的能量从而释放出来的光线。*磷光*（phosphorescence）是指电子束移走之后，相对较稳定的激活电子返回到不激活状态时发出的光线。一般的荧光物质发出的大部分光线是磷光，因为激活和发生荧光只能持续几分之一微秒。荧光物质的*余辉*（persistence）是指从屏幕发光到磷光衰减到其原光强10%的时间。有些荧光物质的余辉可以多达数秒，但是对图形装置使用的大多数荧光物质而言，余辉为10μs到60μs。荧光物质的光线输出随时间指数递减，有关荧光物质特征的细节可参阅[SHER79]。

CRT的*刷新速率*定义为每秒钟里图像重绘的次数。光栅显示系统的刷新速率一般为每秒60次，当刷新速率减小的时候，因为人眼不能把从一个像素发出的单次光脉冲较长地整合在一起，就产生了*闪烁*（flicker）。当大于等于某个刷新速率的时候，闪烁停止，图像稳定，这个刷新速率就叫作*临界停闪频率*（critical fusion frequency，CFF）。停闪的过程对我们是熟悉的：当我们观看电视或者运动图像的时候就是这样，尽管实际上没有图像的时间比有图像的时间要长，但无闪烁的图像对观看者看来是恒定的或者稳定的。

CFF的一个决定因子是荧光物质的余辉。余辉越长CFF越低，停闪频率和余辉之间的关系是非线性的，余辉加倍并不能使得CFF减半。当余辉增加到几秒的时候，停闪频率变得非常小，

而另一极端是，绝对没有余辉的荧光物质照样可以使用，因为所有的眼睛只要求在短时间里看见一些以大于CFF的频率重复的光。

余辉不是影响CFF的惟一因素，CFF也随着图像强度和环境光强的增加而增加，还随发射出的光线的不同波长而变化，最后它还依赖于观察者。停闪是一种生理现象，据报告不同观察者之间CFF的差异可达20 Hz[ROGO83]。所引用的停闪频率通常是大量观察者的平均值。对99%的观察者而言，消除高亮度图像（特别普遍的是前景黑色背景白色的光栅显示器）的闪烁需要80~90 Hz的刷新速率。

*水平扫描率*定义为每秒钟驱动CRT的电路所能显示的扫描线的数目，它大概等于刷新速率和扫描线数目的乘积。对给定的扫描率，刷新速率增加意味着扫描线数目的减少。

单色CRT的*分辨率*的定义与硬拷贝设备类似，分辨率通常利用*收缩光栅*来测量：显示已知数量、距离相等、黑白交替的平行线，然后均匀地减小线间距离直到这些平行线合并成一块均匀的灰色区域，当线间距等于点亮度是点中心亮度约60%的直径时合并会发生。分辨率就等于两条最外直线之间的距离除以光栅中线的数目所得的结果。点的尺寸和分辨率之间有清楚的依赖关系：点尺寸越大，分辨率就越低。 157

在收缩光栅的过程中，不是通过修改光栅位图的内容来减小线间距的，而是通过改变垂直或者水平折射放大器（根据测量垂直或者水平分辨率）的增益（放大量）来得到的，这些放大器控制着位图图像覆盖了屏幕上多大的区域。因此，CRT的分辨率（完全地）不是位图分辨率的函数，而是可能比位图的分辨率高或者低。

分辨率不是一个常数，当电子束中的电子数目增加时，分辨率会降低，这是因为亮线要比暗线宽。这是*光晕*（bloom）的结果，被激活的荧光物质的范围要比被电子束轰击的范围大一点，也因为强电子束形成的点要比弱电子束形成的点大而发生。光栅显示器的垂直分辨率主要由点的大小决定，如果垂直分辨率是每英寸n线，点的尺寸大概是$1/n$英寸。水平分辨率（直线对是垂直的）由点的大小和电子束水平扫过荧光屏时打开和关闭的速度共同决定。这个比率与显示器的带宽有关，下一段我们将讨论这个问题。关于精确定义显示器的分辨率和关于我们感知图像的研究还在进行当中，研究中广泛应用的*调制传输函数*（modulation transfer function）把一个设备的输入信号和输出信号联系在一起[SNYD85]。

显示器的*带宽*跟电子枪打开和关闭的速度有关，要达到每条扫描线n个像素的水平分辨率，电子枪要能够在一条扫描线上打开$n/2$次并关闭$n/2$次以获得交替开关的线，以1000线 × 1000像素，显示刷新频率为60 Hz的光栅扫描为例，要在大约11 ns的时间里绘制一个像素[WHIT84]。一个开关的周期为22 ns，对应的频率就是45 MHz，这个频率就是要获得1000条扫描线（500个线对）的最小带宽，但这还不是实际的带宽，因为我们忽视了点大小的影响。必须要用一个更高的频率来补偿非零的点的大小的影响，这样电子束打开和关闭会更快，从而使点的边界比没有补偿的要更清晰。一个1000线 × 1000像素的显示器其实际带宽达到100 MHz是很普通的。分辨率、带宽和点大小之间的关系是复杂的，直到最近在量化这些关系方面才有了一些进展。

彩色电视机和彩色光栅显示器利用的是某种形式的*荫罩式*CRT，在这种CRT中，显像管观察面的内侧表面上覆盖着紧密分布的多组红、绿、蓝色荧光点，这些点组非常的小，以至于从不同的点发射出来的光线被观察者看成是三种颜色的混合。因此，根据单个荧光点被激活的强弱不同，可以产生出很宽范围内的彩色来。*荫罩*是一块薄金属片，上面钻了很多小孔，它放置在与观测表面很接近的地方，并且经过很精密的对齐，这样三条电子束（一束击打红色类荧光物质，一束绿色，一束蓝色）中的每条只能击打一种颜色类型的荧光点。因此，可以有选择地激活荧光点。 158

图4-14显示出了最普通的荫罩式CRT中的一种，即delta-delta CRT。荧光点以三角形荧光点组（triad）排列，三支电子枪也是以三角形排列，电子枪同时偏转，瞄准（会聚）的是观测面上的同一个点，荫罩上有一个小洞对应着每三个荧光点，这些洞与荧光点组和电子枪都经过精密细致的对齐，使得荧光点组中的每一个点只会暴露在一支电子枪发射出的电子下。高精度的delta-delta CRT的对齐特别困难。另一种方法是图4-15所示的按直线保证精度的（precision in-line delta）CRT，它容易会聚，经常用于高精度（1000条扫描线）的显示器。在这种方法中，排列成直线的电子枪同时激活排列成直线的荧光点。但是，这种直线排列会降低显像管边缘处图像的明锐程度。平板彩色CRT尽管还处于实验室研究阶段，但已经显现出了巨大的商业价值。在平板彩色CRT中，电子束平行于观测平面移动，然后再偏转90度击打荧光屏。

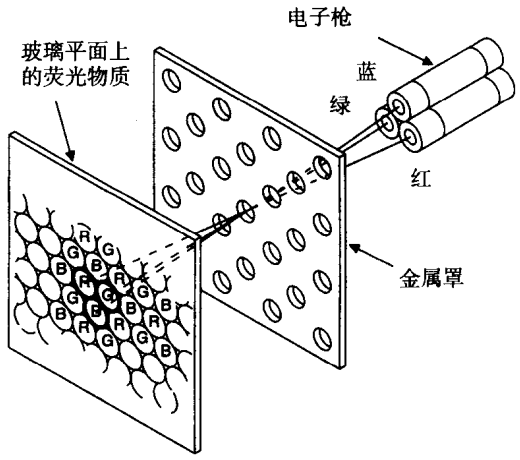


图4-14 delta-delta荫罩CRT。三支电子枪及荧光点按三角形(△)排列，荫罩使每支电子枪发射出的电子束只击中其对应的荧光

对荫罩和荧光点组的需要使得彩色CRT的分辨率有了一个单色CRT所没有的限制。在非常高分辨率的显像管中，荧光点组分布在离三点中心0.21mm的地方，在家用彩色电视机的显像管中分布在离中心0.60mm的地方（这个距离也叫作显像管的点距）。因为一束聚焦很好的电子束也不能保证能够正好击中荫罩的小洞的中心，所以电子束的直径（强度为最大强度的50%处的直径）必须是显像管点距的大约7/4倍。因此对一个点距为 0.25 mm（0.01英寸）的荫罩，电子束的直径大约为0.018英寸，而且分辨率不能高于1/0.018 = 55线每英寸。对一个点距为 0.25 mm、19英寸（沿对角线测量）（即大约15.5英寸宽、11.6英寸高）的显示器，能获得的分辨率仅为15.5 × 55 = 850乘以11.6 × 55 = 638。这个值可以拿来和通常的寻址能力1280 × 1024或者1024 × 800比较一下。如图4-2所示，比寻址能力略小的分辨率是有用的。

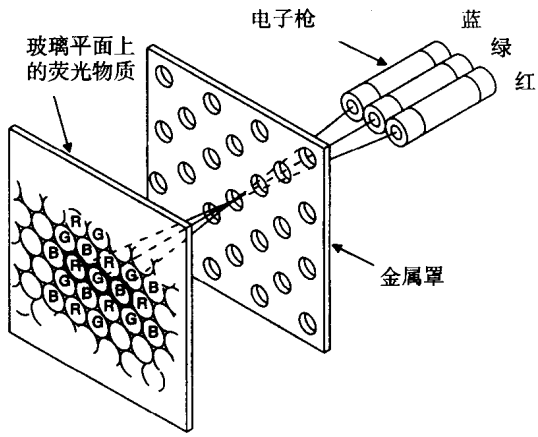


图4-15 精度in-line CRT：三支电子枪依直线排列

荫罩的点距对荫罩式CRT的分辨率是一个重要的限制，点距减少时，分辨率可以增加（假定带宽和点的大小合适），但是点距越小，显像管就越难制造，点距小的荫罩易碎，很难安装，而且更容易因为被电子束加热而弯曲。平面拉伸荫罩（flat-tension-mask）显像管具有平坦的荧光屏，其荫罩被拉伸得很紧以保证正确的位置，这种技术可以获得0.15mm的点距。

荫罩还限制了CRT的亮度。通常只有20%的电子束中的电子击中荧光物质，其余的都击打到了荫罩上。因此，产生光线的电子就比单色显示器的少。虽然可以增加电子束中电子的数目（电子束电流），但是电流越大，聚焦就越困难，而且荫罩上产生的热量也越多，进而加重荫罩

的弯曲。因为平面拉伸荫罩能更好地抵御受热造成的变形，它可以允许更大的光电流，因此可以获得更明亮的图像。

大多数高品质的荫罩式CRT对角线为15~20英寸，荧光屏略有弯曲，观测者会看到光学变形。几种类型的平面CRT逐渐变得实用，包括带有0.31mm点距的29英寸对角线的显像管。当然，价格也很高，但是随着需求的发展，价格会降下来的。

159
~
160

直视存储管（DVST）和标准的CRT相似，只不过它是通过紧贴在屏幕荧光层后的电荷分布来存储图像信息，而不需要再刷新屏幕。由于不需要刷新，没有传统CRT所需要的高扫描频率和高带宽，DVST也能显示复杂的图形。DVST系统的主要缺点是，要修改图像的一部分，必须重新绘制整个修改后的图像以建立DVST中新的电荷分布。重绘过程可能会慢得无法接受（对复杂的图形可能需要数秒）。

普遍应用的Tektronix 4010显示终端就是基于DVST技术的，它是第一种低价格、应用广泛的交互式图形终端，它在计算机图形学中的地位就像T型汽车在汽车业中的地位一样。它的应用是如此的普遍以至于它使用的指令集成了事实上的标准。直到今天，许多显示系统都包含与Tektronix兼容的特征，以便购买者可以继续使用他们为4010所开发的旧版软件的（通常很大）库文件。但是现在DVST已经被光栅系统所取代，基本上已经退出了图形学的舞台。

液晶显示器（LCD）由六层组成，如图4-16所示。最表面一层是垂直偏振器，接下来的一层是在与液晶相邻表面上电沉积的垂直网格线层，再下面一层是一片薄（0.0005英寸）的液晶层，然后是与液晶层相邻的表面上有水平网格线的层，然后是水平偏振器，最后是一层反射器。

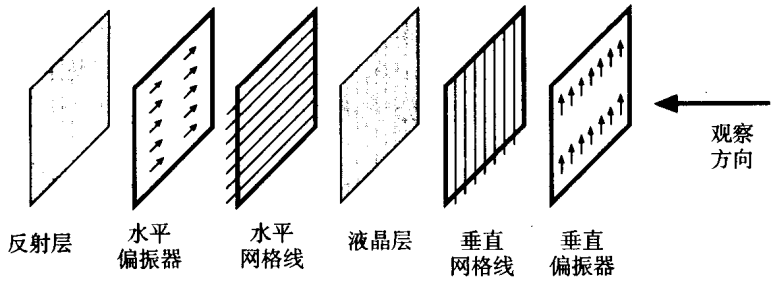


图4-16 液晶显示器的各层，所有层都拼贴在一起形成一薄板

液晶材料由长水晶分子构成，常态下分子排列成螺旋形，这样穿过的偏振光的偏振方向会旋转90°。进入前面一层的光线被垂直偏振，当光线穿过中间的液晶层时，光线的偏振方向被旋转90°成水平，这样光线就可以通过后面的水平偏振器，反射回来，再穿过两层偏振器和液晶。

当液晶处于电场中时，分子按照同一方向排成一线，就失去了偏振的作用，因此电场中的液晶不改变穿过光线的偏振方向，光线仍然保持垂直偏振，不能穿过后面的水平偏振器，光线被吸收，观测者在显示器上就会看到一个黑点。

161

一个在(x_1, y_1)上的黑点是通过*矩阵寻址*（matrix addressing）创建的，通过给水平网格线中的x_1加一个负电压$-V$，给垂直网格线中的导线y_1加一个正电压$+V$可以选择该点：单独的$-V$和$+V$还没有大到足够使液晶分子按照直线排列，但是$-V$和$+V$之间的电压差已经足够大，现在(x_1, y_1)处的液晶不再改变穿过光线的偏振方向，光线仍然保持着垂直偏振，不穿透后面的偏振器，光线被吸收，观测者在显示器上看到一个黑点。

如果要显示(x_1, y_1)和(x_2, y_2)上的点，我们不能简单地在x_1, x_2上加正电压，在y_1, y_2上加负电压：那样会使黑点出现在(x_1, y_1)，(x_1, y_2)，(x_2, y_1)和(x_2, y_2)。因为这些点都要受到电

压的影响。正确的是一个一个接连地选择这些点并且选择过程必须是重复的，以刷新每一个激活点。当然，如果$y_1 = y_2$，那么该行的两个点就可以同时选择。

显示器以光栅扫描的形式一次刷新一行，状态为“开”（例如，白底黑字的液晶显示器中的黑）的行上的点被选择的时间只是每次刷新时间的$1/N$，N是行数。幸运的是，液晶分子一旦按照直线排列后，即便撤去电压，它们也会保持这种状态达数百毫秒（等同于荧光物质的余辉），但即使这样，液晶也不是所有时间都是打开的。

有源阵列面板（active matrix panel）在每个（x, y）网格点放置一个晶体管，这些晶体管用来使液晶快速地转变状态，并用来控制液晶状态改变的程度，这两个属性使得液晶显示器可以用在具有连续色调的袖珍电视中。还能把液晶染上颜色以提供彩色，最重要的是，晶体管充当记录一个单元状态的存储器，并可以让这个单元一直保持这个状态直到被改变。也就是说，这个存储器让一个单元能够始终保持，因此要比需要周期刷新的单元亮度更高。已经制造出了对角线14英寸、分辨率800 × 1000的彩色液晶显示器。

液晶显示器的优点是成本低、重量轻、尺寸小、能耗低。过去液晶显示器最主要的缺点是它是无源的，只是靠反射入射光而不自己发光（尽管可以用背后照明光改进）：强光会使得图像看不清楚。近年来，有源面板的应用使得这个缺点已经不再被考虑。

无源液晶显示技术已经应用在彩色显示上并作为商品销售，如Tektronix液晶快门（liquid-crystal shutter, LCS）LCS放置在标准黑白CRT前面，由三层组成，离CRT最近的后面那层是垂直偏振器，用来偏振CRT发出的光线，这层上还覆盖有透明的导电薄层，接下来的一层是液晶层，第三（最前面）层是彩色偏振器，它把垂直偏振光传输成红光，把水平偏振光传输成绿光，这一层上也覆盖着透明的导电薄层。如果液晶分子处在常态下，它们会把偏振平面旋转90°，这样当光线到达第三层的彩色偏振器时已经是水平偏振的，看上去就是绿色，如果在前层及后层的导电覆盖薄层上加上合适的电压，液晶分子就会按照直线排列，不会影响垂直偏振，光线
162 看上去就是红色的。

液晶以60 Hz的频率在这两种状态之间来回切换，在同时或同步，被看成红色的和绿色的图像在单色显示器上交替，红色和绿色的混合可以通过在同一个点处分别在红色相位和绿色相位的时候赋予它们不同的亮度来创建。

LCS是荫罩式CRT的一种替代品，可是它的颜色分辨率有限，但是这种技术有可能扩展到以三种颜色工作。如果可以，荫罩将不再是获得高分辨率全彩色显示器的限制因素，而像单色显示器一样，点大小和带宽将成为主要的决定因素。另外，不使用荫罩还可以提高系统的强健性。因为LCD显示器又小又轻，还可以应用在头盔显示中，如8.1.6节中所讨论的。

*等离子平板*是若干充满氖气的小球单元的阵列。每一个单元可以设置到“开启”（强化的）状态或“关闭”状态，并且保持其状态直到明确地改变到另一状态。这个记忆属性意味着等离子平板不需要刷新，等离子平板通常每英寸有50到125个孔，对角线10到15英寸，商业上出售的一般是40英寸 × 40英寸，每英寸上有50个孔。用户可以定做更大更密的等离子平板。

氖气小球单元不是一个一个离散的单元，而是一块由三层玻璃组成的集成板的一部分，如图4-17所示。前面那层的内表面具有很薄的垂直电导体条带，中间层有大量的洞（小球单元），后面那层的内表面具有很薄的水平电导体条带。利用矩阵寻址来打开和关闭小球单元，要打开一个小球单元，系统调整相应位置导电线上的电压，使它们之间的差值足够大到拉走氖分子中的电子从而点亮氖气小球使之发亮。点亮后，用一个较低的电压来维持亮的状态。要关闭一个小球单元，系统立刻减小相应位置的电压使它小于维持电压即可。小球单元可以在15 μs内打

开或关闭。在一些设计里，用一个敞开的空穴来代替单个孔，因为氖气只是在局部区域发光，在这些例子里前后的玻璃层由分隔装置隔开。一些等离子平板也可以显示多级灰度。

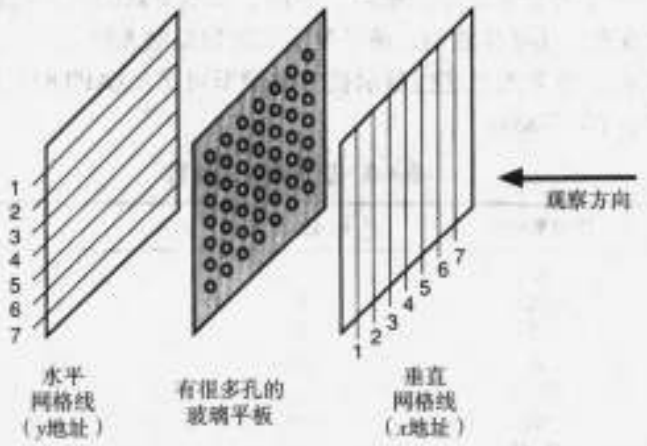


图4-17 等离子体显示器的各层，所有层拼贴在一起形成一薄板

等离子平板的优点是平坦、透明、结实而且不需要位图刷新缓存，它可以和背投系统一起使用，把照相幻灯片混合成计算机生成的动态图形的静态背景。但是等离子平板更多地用在军事上，因为军事上经常要求尺寸小而且结实。它的价格，尽管在不断下降，对于有限的分辨率来说还是相对太高了。实验室中已经有彩色等离子平板的演示，但还没有可用的商品。

电致发光显示器（electroluminescent display）有着与LCD和等离子平板类似的网格结构。在前后玻璃板中的是一薄层（通常为500 nm）电致发光物质，如涂有锰的锌的硫化物，电致发光物质在高电场（大约1 000 000 V/cm）下会发光，板上的一个点通过矩阵寻址的方案在水平和垂直选择线上加几百伏的电压来点亮。也有可用的彩色电致发光显示器。

这种显示器相当明亮而且可以快速地打开和关闭，每个像素位置的晶体管可用来存储图像。一般显示板的大小为6英寸×8英寸到12英寸×16英寸。每英寸上有70个可寻址的点，它的最大缺点是能耗比LCD的高，不过它们的明亮度使它们在一些便携式计算机中得以应用。

电泳显示器（electrophoretic display）有两块平行的紧密间隔的板，板上有矩阵寻址的选择线，中间封装着有色溶液，有色溶液中悬浮着带正电的带色微粒。加在前面选择线上的负电压和加在后面选择线上的正电压把带色微粒拉向前面的板，这样看到的就是微粒而不是有色溶液。加上相反的电压则把微粒拉向后面的板，看到的就是有色溶液。这种显示可以记忆：带色微粒一直会呆在它们所呆的地方，直到被显示地移动。

多数大屏幕显示器使用的是某种形式的投射CRT，在投射CRT中，从一个小（直径几个英寸）但是很明亮的单色CRT发射出的光线经过曲面镜放大并投射出来，彩色系统使用有红、绿、蓝三种滤镜的投影线。荫罩式CRT发出的光线不足以投射到一个大（对角线2 m）的屏幕上。

GE光阀投影系统用于投射CRT输出的光仍不够的非常大的屏幕，光阀正如其名字所暗示：控制通过阀门的光线多少的装置。光源可有远比CRT高得多的强度，最普遍的方法中，电子枪把图像绘制到一片玻璃上的薄油膜上，充电可使油膜的厚度发生变化：因为电子相互排斥的原因加上负电的区域会向外伸展，使得油膜变薄。从高强度光源发出的光线被引导到玻璃上，并由于油膜层的厚度不均而折射到不同的方向，包括Schlieren 条和透镜的光学装置把折射到特定方向上的光线投射到屏幕上，其他方向上的光线则不投射。利用三个投影仪或者使用更复杂的

带有一个投影仪的光学装置可以制成彩色的系统。更多细节参见[SHER79]。其他类似的光阀系统利用LCD来调整光线。

表4-3概括了四种主要显示技术的特点，然而，技术更新的步伐很快，在接下来的几年中一些关系可能发生改变。还要注意到，液晶显示的比较是对无源寻址的，使用有源阵列，可以获得灰度等级和彩色。更多关于这些显示技术的细节可参见[APT85；BALD85；CONR85；PERR85；SHER79；TANN85]。

表4-3 显示技术的比较

	阴极射线管	电致发光	液 晶	等离子体板
能耗	中	中~良	优	中
屏幕尺寸	优	良	中	优
深度	差	优	优	良
重量	差	优	优	优
鲁棒性	中~良	良~优	优	优
亮度	优	优	中~良	优
寻址能力	良~优	良	中~良	良
对比度	良~优	良	中	良
每点亮度等级	优	中	中	中
视角	优	良	差	良~优
色彩能力	优	良	良	中
相对费用	低	中~高	低	高

4.3 光栅扫描显示系统

光栅图形系统的基本概念在第1章已经提及，第2章对光栅显示可能的操作类型进行了更深的探讨，在本节中我们讨论光栅显示器的各种元素，重点放在各种光栅系统相互不同的两个基本方面。

第一，多数光栅显示器都有一个专门的硬件，用来帮助进行扫描转换，将输出图元转换成位图，并且执行移动、拷贝、修改像素或者像素块等光栅操作，我们把这个硬件称为图形显示处理器。显示系统之间最基本的不同在于显示处理器做多少工作，驱动光栅显示器的通用CPU上执行的图形子程序相对的又做多少工作。注意，有时候图形显示处理器也叫作图形控制器（强调它同其他外设的控制单元的相似性）或显示协处理器。第二个不同在于像素图与计算机通用内存的地址空间之间的关系，像素图是计算机通用内存的一部分，或者是独立的。

在4.3.1节我们介绍了一个简单的光栅显示器，它由一个CPU和一个视频控制器组成，像素图作为CPU内存的一部分，视频控制器驱动CRT。没有显示处理器，CPU既做了应用程序的工作也做了图形的工作。在4.3.2节，介绍了一个有单独像素图的图形处理器，4.3.3节讨论了广阔范围的图形处理器的功能，4.3.4节讨论了有图形处理器存在时，像素图可以集成到CPU地址空间中的方法。

165

4.3.1 简单的光栅显示系统

图4-18所示的是最简单和最普通的光栅显示系统的结构，内存和CPU之间的关系和其他非图形计算机系统的一样，但是，内存的一部分还充当像素图，视频控制器显示帧缓存中定义的图像，按照光栅扫描频率的规定通过一个独立的访问端口访问内存。在一些系统中，固定的一部分内存永久地分配给帧缓存，而一些系统有几个相同功能内存区域（在个人计算机中有时称为页），其他的系统则可以指定（通过寄存器）任意一部分内存作为帧缓存，在这种情况下，系统的组织结构可能如图4-19所示，或者整个系统内存可以都是双端口的。

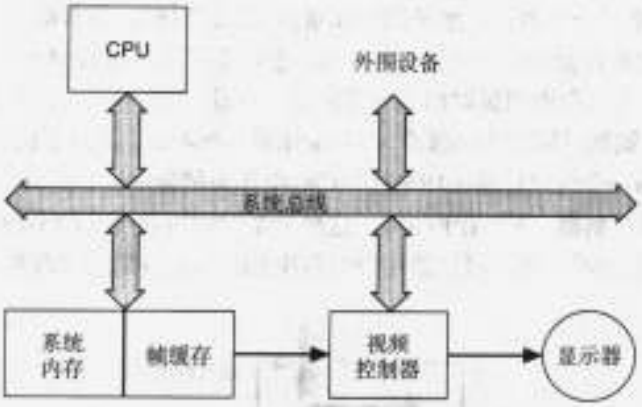


图4-18 普通光栅显示系统的结构，专用的一部分系统内存是双端口的，这样它可以直接被视频控制器访问，而不用中断系统总线的工作

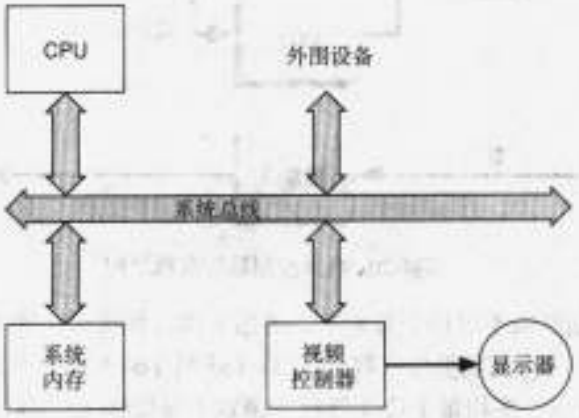


图4-19 简单的光栅显示系统结构。因为帧缓存可以存储在系统内存的任何地方，视频控制器通过系统总线访问内存

应用程序和图形子程序包共享系统内存，并由CPU执行。图形软件包包含扫描转换过程，当应用程序调用子程序时，比如说调用SRGP_lineCoord（x_1, y_1, x_2, y_2），图形软件包能设置帧缓存中适当的像素（关于扫描转换过程的细节参见第3章）。因为帧缓存在CPU的地址空间里，图形软件包可以很容易地访问它来设置像素，并实现第2章里描述过的PixBlt指令。

视频控制器在帧缓存里轮转，每次一条扫描线，通常是每秒钟60次，内存引用地址和光栅扫描同步生成，内存中的内容用来控制CRT电子束的亮度或者颜色。视频控制器的结构如图4-20所示。光栅扫描生成器产生偏转信号，这些信号用来产生光栅扫描，它还控制X地址寄存器和Y地址寄存器，这两个寄存器依次定义了下一个要访问的内存位置。

假设帧缓存的x编址是从0到x_{max}，y编址是从0到y_{max}，那么在一个刷新周期的开始，X地址寄存器设置为0，Y地址寄存器设置为y_{max}（最顶部的扫描线），当第一条扫描线生成的时候，X地址每次增加1直到x_{max}，每个像素的值被取出，并用来控制CRT电子束的强度，第一条扫描线生成完毕后，X地址寄存器复位为0，Y地址寄存器减1，然后依次处理该条扫描线上的各个像素，整个过程对后面的扫描线重复执行，直到最后一条扫描线（$y=0$）生成。

在这种简单的情形下，对每一个要显示的像素的帧缓存都要进行一次内存访问，对一个640像素×480行的中等分辨率显示器，估计显示一个1位像素所用时间的简单方法是：1/(480×640

×60) = 54 ns。这里忽略了一个事实：水平回扫和垂直回扫的时候，像素都不显示（见习题4.10），但是一般的RAM存储器芯片循环周期大约为200 ns，它们不可能支持每54 ns访问一次的要求！因此视频控制器必须在一个内存周期里取出多个像素值，在这个例子中，控制器必须在一个内存周期里取出16个位，从而刷新时间变为16像素×54 ns/像素 = 864 ns。这16个位存放在视频控制器的寄存器里，然后每54 ns一个位逐位移出用来控制CRT电子束的强度，在这864 ns里，大约有4个内存周期：一个用于视频控制器，3个用于CPU。这样分享时间可能要让CPU访问内存的时候等待，就有可能降低CPU速度的25%。当然可以使用CPU芯片上的cache存储器来改善这个问题。

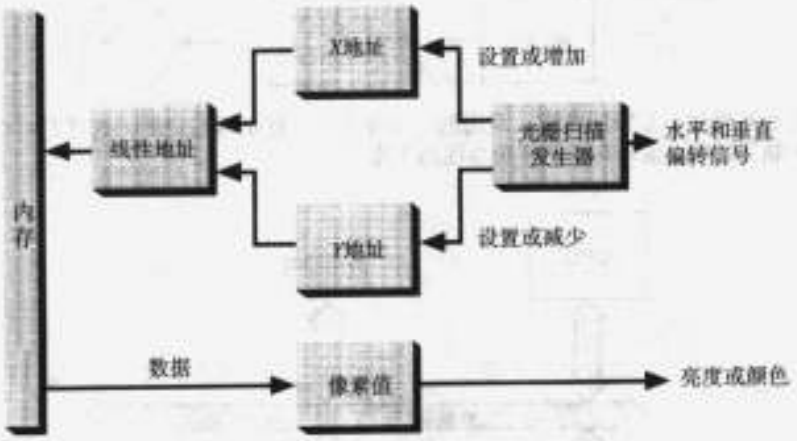


图4-20 视频控制器的逻辑结构

可能一个内存周期里取不出16个像素来，考虑下面这种情况：像素图使用5个64 KB的内存芯片实现，每个芯片在一个周期里可以取出1个位（这叫作64 KB×1片芯片结构），在200 ns的周期时间里取出一共5个位，平均每个位（即每个像素）需要40 ns，这个速度比54ns/像素的速度快不了多少，几乎没有时间让CPU访问内存（除非是在7 ms的扫描线间回扫时间和1250 ms的帧间垂直回扫时间里），但若使用5个32 KB×2片的芯片，200 ns内可以取出10个像素，就给CPU留有一半多一点的时间。对1200×1600的显示器，像素时间是1/(1600×1200×60) = 8.7 ns，200 ns的内存周期时间里，每个周期必须取出200/8.7 = 23个像素。1600×1200的显示器需要1.92 MB内存，可以由8个256 KB的芯片实现，可是256 KB×1片的芯片每个周期只能取出8个像素，而32 KB×2片的芯片每周期可以取出64个像素，就给CPU提供了三分之二的空闲内存周期时间。

CPU对内存的访问和视频控制器对内存的访问明显是一个问题：表4-4说明了问题的数量级。解决的办法是适应光栅显示需要的RAM体系结构，我们将在第18章讨论这些体系结构。

表4-4 描绘图像时处理器可以访问包含位图的内存的时间的百分比

可视区域像素×线	芯片大小	芯片数	每次访问的像素	视频控制器两次访问的时间间隔(ns)	处理器访问时间的百分比
512×512	256K×1	1	1	64	0
512×512	128K×2	1	2	127	0
512×512	64K×4	1	4	254	20
512×512	32K×8	1	8	507	60
512×512	16K×16	1	16	1017	80
1024×1024	256K×1	4	4	64	0

（续）

可视区域 像素×线	芯片大小	芯片数	每次访问 的像素	视频控制器两次 访问的时间间隔(ns)	处理器访问时间 的百分比
1024×1024	128K×2	4	8	127	0
1024×1024	64K×4	4	16	254	20
1024×1024	32K×8	4	32	407	60
1024×1024	16K×16	4	64	1017	80
1024×1024	1M×1	1	1	16	0
1024×1024	64K×16	1	16	254	21
1024×1024	32K×32	1	32	509	61

注：假定内存周期200ns，显示频率为60Hz，假定512×512显示器的像素时间为64ns，1024×1024的为16ns，这些时间是随意假设的，因为它们没有包括水平和垂直描绘时间，实际像素时间相应的大约是45ns和11.5ns

迄今为止我们讨论的只是单色的，每个像素一个位的位图。这个假设对一些应用是可以的，但对其他应用却非常的不满意。对每个像素的亮度的附加控制可以通过为每个像素存储多个位来获得：2位可以获得4个亮度等级，等等。这些位不仅可以用于控制亮度，还可以用于控制颜色。需要多少个位才能使存储的图像看上去是具有连续灰度的？通常5或者6位就足够了，但是8位或者更多位可能是需要的。因此，对彩色显示器，一个有些简单的说法提出需要三倍的位数：添加的原色红、绿、蓝中的每种需要8个位（见第13章）。

尽管固态RAM的价格在下降，每像素24位的系统仍然相对昂贵，并且许多彩色应用在一幅图像中不需要2^{24}种不同的颜色（一般只有2^{18}~2^{20}种）。另一方面，在一幅给定的图像或者一个应用中经常需要使用很少的颜色，也需要从图像到图像或者从应用到应用之间改变颜色的能力，还有，在许多图像分析和图像增强的应用中，想要改变图像的视觉效果，但是不改变定义图像的基本数据。比如说，把所有值低于某个阈值的像素显示成黑色，把亮度范围扩大，给单色图像创建出伪彩色显示。

由于这些各种各样的原因，视频控制器经常包括一个视频查找表（video look-up table）（也称为查找表(look-up table, LUT)），查找表的条目数和像素值一样多，像素值不再用来直接控制电子束强度，而是作为索引值来访问查找表，查找表中条目的值再用来控制CRT的亮度或者颜色。一个值为67的像素就表示访问查找表中第67项的内容，并且用这个内容控制电子束强度。在一个显示周期中对每个像素进行这样的查找操作，所以查找表必须能够快速访问，CPU必须能在程序命令的时候装入查找表。

在图4-21中，查找表插入在帧缓存和CPU的中间，帧缓存每个像素有8位，所以查找表有256项。

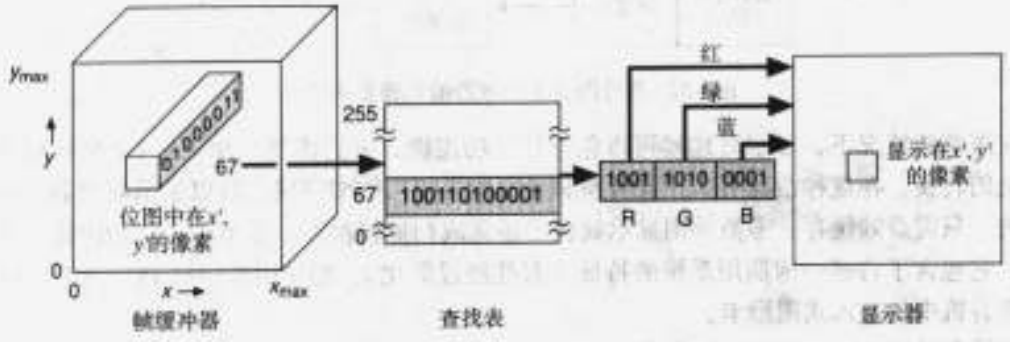


图4-21 视频查找表的结构。值为67（二进制值01000011）的像素显示到屏幕上，红色电子枪的强度为最大值的9/15，绿色的为10/15，蓝色的为1/15，所示的是12位的查找表，通常的会达到24位

图4-18和图4-19显示的简单光栅显示系统的结构使用在很多便宜的个人计算机中。这样的系统造价低，但是有很多缺点。首先，软件扫描转换很慢，例如一条扫描线上的每个像素的(x, y)地址都要计算，然后转换成由一个字节和字节中的位组成的内存地址。尽管每一步都很简单，但是都要重复许多次，基于软件的扫描转换使得应用程序与用户交互的速度全部变慢，可能会使用户不满。

这种结构的第二个缺点是：当寻址能力或者显示刷新速率增加时，视频控制器的内存访问次数也随之增加，因此降低了CPU可用的内存周期的数目，CPU也因此慢下来。特别是图4-19中的结构。在图4-18中，系统内存有一部分是双端口的，当CPU访问帧缓存以做扫描转换或者光栅操作时，速度下降就会发生。在考虑CPU访问帧缓存的方便性和系统的结构简单性时，也还要考虑这两个缺点。

4.3.2 具有外围显示处理器的光栅显示系统

具有外围显示处理器的光栅显示系统是一种避免了简单光栅显示器的缺点的普遍结构（见图4-22），它引入了一个独立的执行诸如扫描转换和光栅操作等图形功能的图形处理器和一个独立的用于图像刷新的帧缓存。现在我们有两个处理器：通用CPU和专用显示处理器。我们还有三块内存区域：系统内存，显示处理器内存和帧缓存。系统内存存放数据以及在CPU上执行的程序：应用程序，图形软件包和操作系统。相似地，图形处理器内存上也存放着数据和进行扫描转换及光栅操作的程序。帧缓存存放扫描转换和光栅操作生成的可显示的图像数据。

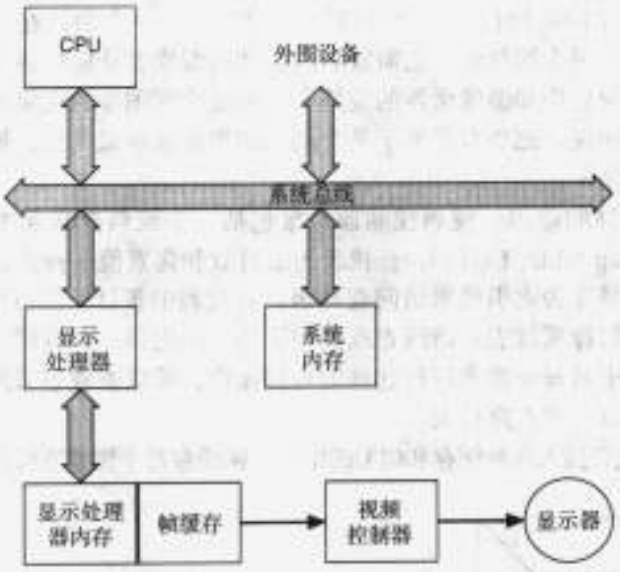


图4-22 有外围显示处理器的光栅显示结构

在简单的情况下，显示处理器可以包含特定的逻辑，来完成从二维(x, y)坐标到线性内存地址的转换。在这种情况下，扫描转换和光栅操作仍由CPU完成，所以显示处理器内存是不需要的，只需要帧缓存。多数外围显示处理器还完成扫描转换，在本节中，我们讨论一个原型系统，它包含了许多一般商用系统的特征（有些经过简化），如使用在IBM PC、XT、AT、PS以及兼容机中的插入式图形卡。

帧缓存是1024 × 1024 × 8位/像素，查找表有256项，每项12位，红、绿、蓝每种颜色使用4位，坐标原点在左下方，仅显示像素图的前768行（y从0到767），显示器有6个状态寄存器，可

以被不同的指令设置，并影响其他指令的执行。这些寄存器是：CP（由X位置寄存器和Y位置寄存器组成），FILL，INDEX，WMODE，MASK和PATTERN。下面将解释它们的操作。

简单光栅显示的指令如下：

- Move（x, y） 将定义当前位置（CP）的X、Y寄存器设置成 x 和 y，因为像素图是1024×1024的，x和y必须在0到1023之间。
- MoveR（dx, dy） dx和dy的值被加到X、Y寄存器，因此定义了新的CP，dx和dy必须在－1024到1023之间，以2的补码表示。所做的加法可能导致溢出，因此X寄存器和Y寄存器环绕式处理。
- Line（x, y） 从CP到（x, y）画一条直线，（x, y）成为新的CP。
- LineR（dx, dy） 从CP到CP＋（dx, dy）画一条直线，CP＋（dx, dy）成为新的CP。
- Point（x, y） 设置（x, y）处的像素，（x, y）成为新的CP。
- PointR（dx, dy） 设置CP＋（dx, dy）处的像素，CP＋（dx, dy）成为新的CP。
- Rect（x, y） 在CP和（x, y）之间画一个矩形，CP不受影响。
- RectR（dx, dy） 在CP和CP＋（dx, dy）之间画一个矩形，参数dx可以看成是矩形的宽，dy看成是矩形的高，CP不受影响。
- Text（n, *address*） 从CP开始显示内存位置为*address*的n个字符。字符定义在7×9的像素网格中，垂直和水平分别有额外的2个分隔像素用于分隔字符和行。CP更新为第$n+1$个字符将被显示的区域的左下角。
- Circle（*radius*） 以CP为圆心画一个圆，CP不受影响。
- Arc（*radius*，*startAngle*，*endAngle*） 画圆心在CP的一段圆弧，角度单位是十分之一度，从x轴沿逆时针方向增加。CP不受影响。
- CircleSector（*radius*，*startAngle*，*endAngle*） 画一个扇形的封闭区域，直线从CP连接到圆弧的两个端点，CP不受影响。
- Polygon（n，*address*） 地址为*address*的内存存储顶点列表（$x_1, y_1, x_2, y_2, x_3, y_3, \cdots, x_n, y_n$），以（$x_1, y_1$）为起点画多边形，经过所有这些顶点直到（$x_n, y_n$），然后回到（$x_1, y_1$），CP不受影响。
- AreaFill（*flag*） *flag*用来设置光栅显示中的FILL标志，当此标志设置成ON的时候（用一个非零的*flag*值），用命令Rect、RectR、Circle、CircleSector、Polygon创建的区域都会用Pattern命令定义的图案填充。
- PixelValue（*index*） 像素值*index*装入INDEX寄存器，当任何一个之前列表中的输出图元进行扫描转换时将该值填入像素图。
- Pattern（*row*1，*row*2，…，*row*16） 16个2字节的参数定义了填充图案，用来填充由Rect、RectR、Circle、CircleSector、Polygon创建的区域。图案是一个16×16的位矩阵，当创建一个填充区域且FILL标志为ON的时候，如果PATTERN寄存器中的一个位是1，则INDEX寄存器中的像素值填入到像素图中，否则像素图不受影响，当PATTERN寄存器中的所有位都为1时，进行的就是实心填充。
- WBlockR（dx, dy, *address*） 将开始地址为*address*的主内存中存放的8位像素值写入到像素图中从CP到CP＋（dx，dy）的矩形区域，从区域的左上角开始向顶向下逐行写入。
- RBlockR（dx, dy, *address*） 将像素映射中从CP到CP＋（dx, dy）的矩形区域读入开始地址为*address*的主内存中，从区域的左上角开始向顶向下逐行读入。

166 ~ 171

- RasterOP（*dx*, *dy*, *xdest*, *ydest*） 将帧缓存中从CP到CP +（*dx*, *dy*）的矩形区域和左下角为（*xdest*, *ydest*）同样大小的目标区域结合，目标区域被覆写，结合受WMODE寄存器的控制。
- WMode（*mode*） *mode*的值被装入WMODE寄存器中，该寄存器控制帧缓存中的像素和要写入帧缓存的任何像素值结合的方式，参数*mode*有四个值：**replace**（替换），**xor**（异或），**and**（与），**or**（或）。这些模式进行的操作如第2章所述。注意，为了简洁，前面对命令的描述编写得似乎**replace**（替换）是仅有的一种写模式值。在xor模式下，写入到帧缓存中的新像素值和当前的值逐位地进行异或操作结合到一起。
- Mask（*mask*） 8位的*mask*值装入到MASK寄存器中，该寄存器控制帧缓存的哪一位在写帧缓存的时候被修改：1表示允许修改对应的位，0则禁止修改。
- LuT（*index*, *red*, *green*, *blue*） 用给定的颜色装入查找表的第*index*项中。每个颜色参数的低4位装入到查找表中。

表4-5总结了这些命令。注意，MASK寄存器和WMODE寄存器影响所有写入帧缓存的命令。

表4-5 光栅显示命令概括

命令助记符	参数及长度	CP所受的影响	影响命令的寄存器
Move	*x*(2), *y*(2)	CP := (*x*, *y*)	–
MoveR	*dx*(2), *dy*(2)	CP := CP + (*dx*, *dy*)	CP
Line	*x*(2), *y*(2)	CP := (*x*, *y*)	CP, INDEX, WMODE, MASK
LineR	*dx*(2), *dy*(2)	CP := CP + (*dx*, *dy*)	CP, INDEX, WMODE, MASK
Point	*x*(2), *y*(2)	CP := (*x*, *y*)	CP, INDEX, WMODE, MASK
PointR	*dx*(2), *dy*(2)	CP := CP + (*dx*, *dy*)	CP, INDEX, WMODE, MASK
Rect	*x*(2), *y*(2)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
RectR	*dx*(2), *dy*(2)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
Text	*n*(1), *address*(4)	CP := next char pos'n	CP, INDEX, WMODE, MASK
Circle	*radius*(2)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
Arc	*radius*(2), *startAngle*(2), *endAngle*(2)	–	CP, INDEX, WMODE, MASK
CircleSector	*radius*(2), *startAngle*(2), *endAngle*(2)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
Polygon	*n*(1), *address*(4)	–	CP, INDEX, WMODE, MASK, FILL, PATTERN
AreaFill	*flag*(1)	–	–
PixelValue	*index*(1)	–	–
Pattern	*address*(4)	–	–
WBlockR	*dx*(2), *dy*(2), *address*(4)	–	CP, INDEX, WMODE, MASK
RBlockR	*dx*(2), *dy*(2), *address*(4)	–	CP, INDEX, WMODE, MASK
RasterOp	*dx*(2), *dy*(2), *xdest*(2), *dest*(2)	–	CP, INDEX, WMODE, MASK
Mask	*mask*(1)	–	–
WMode	*mode*(1)	–	–
LuT	*index*(1), *red*(1), *green*(1), *blue*(1)	–	–

注：每个参数后括号内的数字是参数的长度（以字节为单位），本表还指出了命令对CP的影响以及哪些寄存器影响命令运作的方式。

命令和立即数通过位于CPU地址空间中专用部分的先进先出（FIFO）缓冲区（如队列）传输到显示处理器，图形软件包把命令送入到队列中，显示处理器取得指令并执行。在特定的内存位置还存放着指向这个缓冲区的起始地址和结束地址的指针，供CPU和显示处理器访问。每次移走一个字节时显示处理器修改指向起始地址的指针，每次加入一个字节时CPU修改指向结束地址的指针，要做适当的测试以保证对空的缓冲区不进行读操作，对满的缓冲区不进行写入操作，使用直接内存访问来给指令提供寻址数据。

对于命令传递，队列比使用单个指令寄存器或显示处理器可以访问的位置要更吸引人。第一，变长的指令适合队列的概念；第二，CPU可以超前于显示处理器，把许多显示命令排在队列里。当CPU发布完显示命令后，在显示处理器处理命令清空队列时它可以去处理其他工作，

编程实例

对显示的编程有点类似于第2章所述的SRGP软件包的使用。所以我们这里只给出了一些例子。“Z”表示指定的是十六进制值，“A”表示使用后面括号中的地址列表。

下面一段程序创建了全黑色背景上的一条白色直线：

```
LuT          5, 0, 0, 0                  查找表第5项为黑色
LuT          6, Z'F', Z'F', Z'F'         查找表第6项为白色
WMode        replace
AreaFill     true                        打开FILL标志
Pattern      32Z'FF'                     32个全1的字节，实心图案
Mask         Z'FF'                       使能够写入到所有平面
PixelValue   5                           使用像素值5进行扫描转换
Move         0, 0
Rect         1023, 767                   帧缓存中可视的部分现在是黑色
PixelValue   6                           使用像素值6进行扫描转换
Move         100, 100
LineR        500, 400                    画直线
```

173 ~ 174

下一段程序创建黑色背景上与蓝色三角形交迭的红色圆：

```
LuT          5, 0, 0, 0                  查找表第5项为黑色
LuT          7, Z'F', 0, 0               查找表第7项为红色
LuT          8, 0, 0, Z'F'               查找表第8项为蓝色
WMode        replace
AreaFill     Z'FF'                       打开
Pattern      32Z'FF'                     32个全1的字节，实心图案
Mask         Z'FF'                       使能够写入到所有平面
PixelValue   5                           准备好画黑色的矩形
Move         0, 0
Rect         1023, 767                   现在帧缓存中可视部分为黑色
PixelValue   8                           接下来画蓝色三角形，把三角形当成三个顶点的多边形
Polygon      3, A(200, 200, 800, 200, 500, 700)
PixelValue   7                           把圆画在三角形的上面
Move         511, 383                    把CP移动到显示器的中心
Circle       100                         以CP为圆心画半径100的圆
```

4.3.3 显示处理器的附加功能

我们的简单显示处理器只完成一些能被实现的与图形相关的操作。设计者所面临的诱惑是：给显示处理器增加功能，以更多地减轻CPU的负担，比如使用局部内存来存放显示指令列表，完成裁剪和窗口-视口转换，也许还提供拾取相关逻辑和图形元素被拾取后的自动反馈。最终，显示处理器变成了另一个完成通常图形交互工作的通用CPU，设计者又要尝试着增加特定功能的硬件以减轻显示处理器的负担。

Myer和Sutherland在1968年确定了这个“轮回”（wheel of reincarnation）[MYER68]。作者的观点是在通用功能和专用功能之间要有一个折衷，通常，专用硬件完成工作比通用处理器快。另一方面，专用硬件更昂贵且不能用于其他用途。这种折衷在图形系统设计中是持久的题目。

如果裁剪（第3章）增加到显示处理器的功能中，输出图元可以用坐标指定给处理器而不使用设备坐标。这种说明可以在浮点坐标中给定，虽然一些显示处理器只能处理整数（随着低廉的浮点芯片的使用，这种情况正在改变中）。如果只使用整数，应用程序使用的坐标也必须是整数的，或者图形软件包必须把浮点坐标映射到整数坐标。要使映射成为可能，应用程序必须给图形包一个矩形，这个矩形保证包含了指定给图形包的所有输出图元的坐标。然后这个矩 175

形映射到最大的整数范围，这样在这个矩形内的一切都是在整数坐标的范围里的。

如果子程序包是三维的，显示处理器就能够完成第5章和第6章中所述的复杂得多的三维几何变换和裁剪。同样，如果图形包里包含了三维曲面图元，比如多边形区域，显示处理器还能进行第15章和第16章讨论的可见面判定（visible surface-determination）和绘制步骤（rendering step）。第18章讨论了一些能使这些步骤完成得更快的把通用和专用VLSI芯片组织在一起的基本方法。许多商用的显示器都提供了这些特征。

另外一个经常给显示处理器增加的功能是*本地段存储*（local segment storage），也叫*显示列表存储*（display list storage）。显示指令被分组到命名的段中，具有未被裁剪的整数坐标，并存储在显示处理器内存中，允许显示处理器的操作更自主于CPU。

显示处理器对这些存储起来的段能做些什么呢？它可以对它们进行变换和重绘，像缩放和滚动。能够提供这些段到新位置的本地拖动功能。通过让显示处理器对光标位置和所有图形图元（更有效率的方法在第7章讨论）比较来实现本地拾取。当删除一个段的时候，需要用段存储来重新生成以填充产生的洞。段可以被创建、删除、编辑，还可以使段可见或者不可见。

段还可以被拷贝或者引用，这两种操作都减少了必须从CPU传送给显示处理器的信息，也使显示处理器自己的内存使用更经济。例如，创建一个VLSI芯片衬垫外形的显示指令在绘制的时候要多次用到，这些指令只需要给显示处理器传送一次，存储成一个段，然后每次出现同样的衬垫时就传送引用这个段的显示指令。利用这个功能可以建立复杂的层次数据结构，许多具有本地段内存的商用的系统可以拷贝和引用其他段。当显示段时，必须保存显示处理器的当前状态，然后进行另一个段的引用，就像保存CPU的当前状态，然后进行子程序调用一样。引用可以嵌套，导致了*结构显示文件*或*层次显示列表*的出现，如在PHIGS中 [ANSI88]，第7章将进行更深的讨论。在GKS图形包[ENDE87；HOPG86]中使用的是线性无嵌套的显示列表，一个已存在的段可以拷贝到将要创建的段中。

段数据结构不是必须放在显示处理器的内存中（图4-22），它们可以直接由图形包在系统内存中创建，由显示处理器访问。当然在这种方式下需要显示处理器必须直接连接在系统总线上，RS232接口和以太网速度的连接都是不可行的。

如果所有要显示的信息都用段数据结构表示的话，显示处理器还可以实现窗口管理器的操作，如移动、打开、关闭、改变大小、滚动、入栈、出栈等。当窗口平移时，段也进行有效的视点旋转。在某些地方“轮回”又再度出现，但是我们要注意到便宜得令人吃惊的专用VLSI芯片，等到本书过时，窗口管理芯片可能只花几个美元就能买一片。确实，技术更新换代是如此之快，显示处理器中的图形功能将继续惊人地增加，而其价格则继续下降。

尽管和4.3.1节中的简单光栅显示系统相比，具有图形显示处理器和独立帧缓存的光栅显示
176 系统结构有着许多优点，它也还是有着一些缺点。如果显示处理器是作为DMA端口上或者RS232接口上的外设，则每次传送指令给它的时候就会有相当多的操作系统管理开销（显示处理器的指令寄存器映射到CPU的地址空间就不会发生这种情况，因为图形包很容易就可以直接设置寄存器）。

如图4-22所示，也可对内存做有标记的划分。在显示列表内存中建立显示列表是很慢的，因为需要发布增加或者删除元素的显示处理器命令。显示列表可能不得不在主处理器内存中保存一份拷贝，因为它经常读不回来。因此，显示列表由图形子程序包直接建立在主存中的环境更灵活、更快速，对程序更方便。

光栅操作命令特别复杂，从概念上说，它应该有四种可能的源-目的对：系统内存到系统

内存，系统内存到帧缓存，帧缓存到系统内存，帧缓存到帧缓存（在此，图4-22中的帧缓存和显示处理器的内存被认为是相同的，因为它们都在同一个系统地址空间里）。但是在显示处理器系统里，对不同的源-目的对使用不同的方式处理，可能系统内存到系统内存的这种情况不存在。缺乏对称性使得程序员的任务变得复杂，降低了灵活性。比如说，如果位图超出屏幕的部分填充着的是菜单、字体等等，就很难利用主存来作为溢出区域。更进一步，因为像素图的使用是如此的广泛，不支持对存储在主存中的像素图进行光栅操作是不可行的。

另外一个问题是：扫描转换算法的输出必须输出到帧缓存，这个要求排除了双缓存：扫描转换在系统内存中创建一幅新的图像，然后把它拷贝到像素图替换掉存储在那里的原有图像。另外，某些窗口管理器策略和动画技术要求部分或者全部变暗的窗口在屏幕以外的调色板中保持，这又要求扫描转换到系统内存去（第10章和第19章）。

本节前面定义的显示处理器像许多真正的显示处理器一样，通过系统总线上的I/O传输在系统内存和帧缓存之间移动光栅图像。但是，实时操作里这种移动可能太慢，像动画、拖动、弹出窗口和菜单，操作系统初始化传输的时间和总线的传输率是瓶颈所在。通过增加显示处理器的内存以装载更多的屏幕外的像素图可以部分克服这个问题，但那样一来这部分内存就不能用于其他用途了——总之，几乎永远都不会有足够多的内存！

4.3.4 具有集成显示处理器的光栅显示系统

把帧缓存作为系统内存的一部分，我们可以克服上节所讨论的外围显示处理器的许多缺点，这就是图4-23所示的单地址空间（single-address-space, SAS）显示系统体系结构。这里显示处理器、CPU和视频控制器都在系统总线上，因此都可以访问系统内存。帧缓存的起始地址，一些情形下还有大小，都存放在寄存器中，双缓存就变得很简单，只需重新填入寄存器；扫描转换的结果可 177
以送到帧缓存用于直接显示，或者送到系统内存别的地方用于以后的显示。类似地，显示处理器进行的光栅操作的源和目的可以在系统内存的任何地方（现在我们只对内存感兴趣）。这种安排还有一点吸引人之处，是因为CPU可以直接操作帧缓存中的像素，只要简单地读写合适的位就可以。

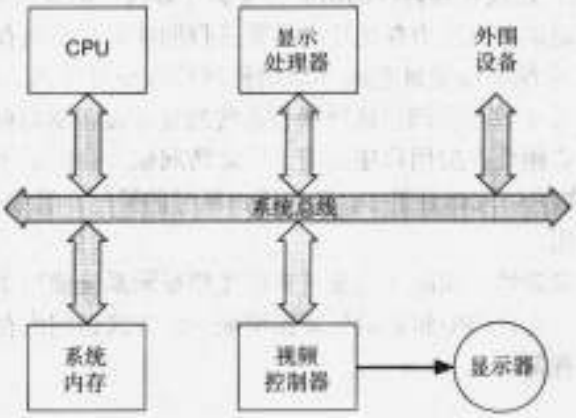


图4-23 单地址空间（SAS）光栅显示系统的体系结构，它有一个集成的显示处理器，该显示处理器可以有自己的用于存放算法的内存和工作存储器

但是，SAS体系结构有许多缺点，对系统内存访问的竞争是最严重的。通过把系统内存的特定部分专用于帧缓存和通过提供从视频控制器到帧缓存的第二个访问端口，至少可以部分解决这个问题，如图4-24所示。另一个解决方法是使用带有指令或数据高速缓冲存储器的CPU，减少CPU对频繁快速访问系统内存的依赖。当然这些方法和其他方法可以巧妙的集成到一起。

第18章将讨论更多的有关细节。由于限制，硬件实现的PixBlt只能在帧缓存上工作，应用程序员看到的一个PixBlt指令可能是分几种不同的情况来对待的，如果硬件不支持源和目的，还要进行软件模拟，一些处理器实际上已经足够快到完成软件模拟，特别是有指令高速缓冲存储器（instruction-cache memory），软件模拟的最内层循环可以保持在高速缓冲存储器里。

和前面提到的一样，帧缓存的非传统内存芯片结构也能避免内存竞争问题。一种方法是在一次访问时间里打开扫描线上所有的像素，因此减少了扫描转换到内存所需要的内存周期，特别是对于填充区域。德州仪器（Texas Instruments）开发的视频RAM（VRAM）结构可以在一
178 个周期里读出一条扫描线上的所有像素。第18章也会给出更多的细节。

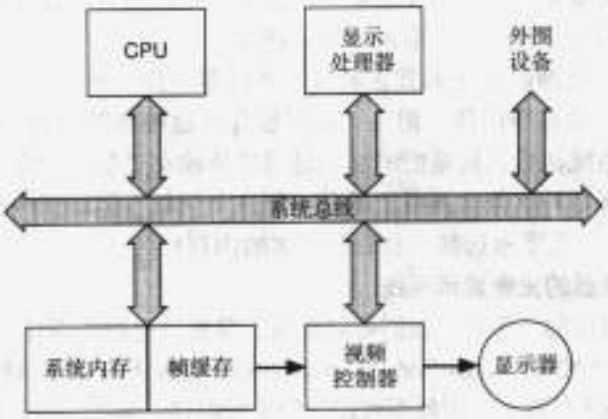


图4-24 公用单地址空间光栅显示系统的体系结构，具有一个集成的显示处理器（和图4-23比较），该显示处理器可以有私有的用于存放算法的内存和工作存储器，系统内存的专用部分是双端口的，以便视频控制器可以直接访问，而不用中断系统总线的工作

如果CPU具有虚拟地址空间时，出现了另一个设计上的复杂因素，像普遍应用的Motorola 680x0和Intel 80x86系列，以及各种各样的精简指令集计算机（RISC）处理器。在这种情况下，显示处理器生成的内存地址和其他内存地址一样要进行同样的动态内存地址转换。另外，许多CPU结构区分核心操作系统虚拟地址空间和应用程序虚拟地址空间，经常也需要帧缓存（在SRGP术语中是画布0）处于核心空间，这样操作系统的显示设备驱动程序可以直接访问。但是应用程序所分配的画布必须处在应用程序空间。因此访问帧缓存的显示指令必须区分核心地址空间和用户地址空间。如果访问的是核心，就应该由耗时的操作系统服务调用来调用显示指令，而不是由简单子程序调用。

尽管有这些潜在的复杂性，实际上越来越多的光栅显示系统使用单地址空间结构，一般是图4-24所示的那种类型。允许CPU和显示处理器用统一的方式访问内存的任何部分这一灵活性非常引人注目，并且编程简单。

4.4 视频控制器

视频控制器的最主要任务是持续地刷新显示。有两种基本的刷新方式：交错的和不交错的。前者使用在广播电视和设计用来驱动正规电视的光栅显示中。其刷新周期被分成两个部分，每
179 部分持续1/60秒，一次完整的刷新持续1/30秒，所有奇数行扫描线在第一部分时间里显示，所有偶数行扫描线在第二部分时间里显示。隔行（交错）扫描的目的是每次以60 Hz的频率在屏幕的所有区域放置一些新的信息，因为30 Hz的刷新频率容易导致闪烁。交错显示的净效果是

产生的图像其有效刷新频率更接近60 Hz而不是30 Hz。在相邻扫描线实际上显示相似信息时这一技术较为有用；在交替的扫描线上有水平线的图像将有严重的闪烁。大部分视频控制器以60 Hz或者更高的刷新频率刷新并且使用非交错扫描。

视频控制器的输出分为以下三种模式：RGB、单色和NTSC。对RGB（red，green，blue），用单独的电缆来传输红、绿、蓝信号，控制荫罩式显示器的三支电子枪，另外一根电缆传输标志开始垂直回扫和水平回扫的同步信号。RGB信号的电压、波形和同步时间都有标准，对480扫描线的单色信号，RS-170是其标准；彩色信号的标准是RS-170A；高分辨率单色信号的标准是RS-343。同步时间经常和绿色信号包含在同一根电缆里，这种情况下信号叫作复合视频（composite video）信号。单色信号使用同样的标准，但是只有亮度和同步信息，或者仅仅只是一根传输亮度和同步信号的复合电缆。

NTSC（国家电视系统委员会）视频是北美电视商品中使用的标准。颜色、亮度和同步信息都合并在一个带宽为5 MHz的信号中，以525条扫描线进行广播，分为两个262.5条扫描线的部分，只有480条扫描线是可见的，剩下的扫描线在每个部分结束进行垂直回扫的时候出现。单色电视使用亮度和同步信息，彩色电视还使用彩色信息控制三支电子枪。带宽的限制可以允许在分配给电视的频率范围内使用许多不同的频道进行广播。不幸的是，这个带宽限制了图像的质量，使得它的有效分辨率只有350 × 350。不过，NTSC是录影带录制装置的标准。问题总会改善，现在对用于录像带和卫星广播的1000线高清晰度电视（HDTV）的兴趣越来越大。欧洲和前苏联电视广播和录影带标准是SECAM和PAL，两个625扫描线、50 Hz标准。

一些视频控制器加入了可编程光标，光标的形状存储在16 × 16或者32 × 32大小、位于帧缓存顶部的像素图中，这样就避免了每次刷新周期里都需要把光标形状PixBlt到帧缓存中去。类似地，一些视频控制器在帧缓存的顶部增加了几个小的固定尺寸的像素图（叫作精灵（sprite）），这个特征经常用于视频游戏。

4.4.1 查找表动画

光栅图像的动画效果可以从几个途径获得。要显示一个旋转的物体，我们可以逐一地从略有不同的位置把物体的图像扫描转换到像素图中，扫描转换至少必须每秒钟10次（最好是15次到20次）才能获得平滑的效果，因此一幅新的图像必须在不多于100 ms的时间内创建出来，但是如果对物体进行扫描转换占用了这100 ms中的大部分，比如说75 ms，那么完整的物体只能在剩下的25 ms中显示，然后必须删除它，重新绘制，效果很差。使用双缓冲可以避免这个问题。帧缓存分为两个图像，每个图像拥有整个帧缓存中每个像素一半的位，我们把像素图的两半分别称为*image*0和*image*1，下面描述动画的生成：

180

```
装入查找表，用背景色显示所有像素；
物体扫描转换到image0；
装入查找表，仅显示image0
do {
  物体扫描转换到image1
  装入查找表，仅显示image1
  旋转物体的数据结构描述
  物体扫描转换到image0
  装入查找表，仅显示image0
  旋转物体的数据结构描述
} while（不结束条件）
```

当然，如果旋转和扫描转换物体的时间多于100 ms，那么动画看上去就在跳动，但从一幅图像到另外一幅图像的过渡是立即的，因为装入查找表通常只花不到1 ms的时间。

查找表动画的另外一种方式是显示短的重复的图像系列[SHOU79]。假设我们要显示一个弹跳的球。图4-25说明了帧缓存的装入方式，标志像素值的数字放置在帧缓存的每个区域。图4-26演示了如何在每一步装入查找表的内容，在颜色为0的背景上显示其中一个小球。循环查找表中的内容，我们可以获得动画效果。不仅是弹跳的小球，电影帐篷中移动的光线的效果、管子中流出的水流、旋转的车轮等等都可以模拟出来。21.1.4节将更深入地讨论这个主题。

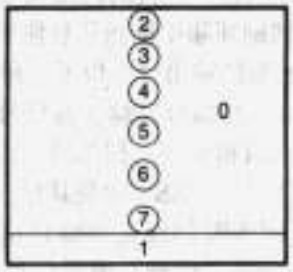


图4-25 弹跳的球的动画所对应帧缓存的内容

通道号	动画每一步装入查找表的颜色										
	1	2	3	4	5	6	7	8	9	10	11
0	white	white	white	white	white	white	white	white	white	white	white
1	black	black	black	black	black	black	black	black	black	black	black
2	red										red
3		red								red	
4			red						red		
5				red				red			
6					red		red				
7						red					

图4-26 白色背景，在黑色表面上拍红球的查找表，未标明的项均为白色

对更复杂的循环动画（比如旋转一个复杂的线框架物体）可能在帧缓存中存放的就是独立的图像，而不是交迭的图像。在此情况下，一些显示的图像会有“洞”，少数这样的洞是没有影响的，特别是不追求完美时。但是出现更多的“洞”时，动画就失去了它的效果，这时候双缓存就显得更吸引人。

4.4.2 位图变换和窗口技术

在一些视频控制器里，像素图和观测表面之间的联系被切断，也就是说，帧缓存中的位置和观测表面上的位置不再有固定的对应关系，而由图像变换来定义对应关系。图像变换把帧缓存变换到观测表面，变换通常包括平移、放缩、旋转和裁剪操作。

图4-27演示了一些光栅显示系统中发现的变换的类型，帧缓存中由裁剪区域定义的一部分放大到充满整个观测表面。裁剪窗口与观测表面大小的比率必须是整数（图中为3）。没有使用帧缓存中裁剪窗口以外的像素，也没有修改任何像素的值，视频控制器以刷新速率进行变换。

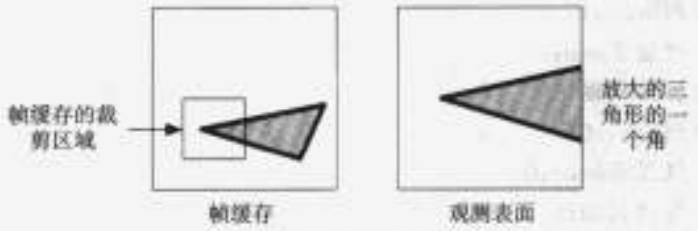


图4-27 在观测表面放大的帧缓存的一部分

图像变换每秒钟可以进行许多次，造成图像的翻卷或者放大的实时动态效果。还能够快速显示任意图像的系列，把它们装入到帧缓存中的不同区域，定时进行图像变换，首先显示第一个区域，然后显示下一个，等等。

放大一幅图像需要的比例变换在显示图像时可简单地通过重复窗口中的像素值来实现。比例因子为2时，每个像素值使用4次，在两条相邻的扫描线上每条两次。图4-28显示了以2为比例因子放大一个字母及和它相邻的一条直线的效果。放大并不能展现更多的细节，除非存储的图像分辨率比它显示的分辨率高。图形放大后有了更明显的锯齿形的外观。因此这种动画效果牺牲了空间分辨率，但是保持了全部的颜色范围。前面一节所述的双缓存技术保持了图像的空间分辨率，但是却减少了任一个图像中可使用的颜色数。

181 ~ 182

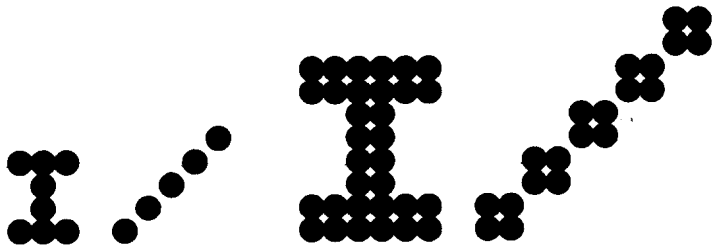


图4-28 因子为2，按比例放大一幅像素图的效果

在一些更普通的图像变换应用中，按比例变换的图像只覆盖由视口所定义的观测表面的一部分，如图4-29所示。现在我们必须给系统定义应该在视口以外的区域显示什么。一种可能是显示一种恒定的颜色或亮度，图示的另一种可能是显示帧缓存自己。后一种选择的硬件实现很简单，将存放视口边界坐标的寄存器与定义光栅扫描当前位置的X、Y寄存器比较，如果电子束处于视口中则从帧缓存中窗口区域取得像素，并且按照需要复制，否则像素从帧缓存中与电子束相同的坐标（x,y）位置取出。

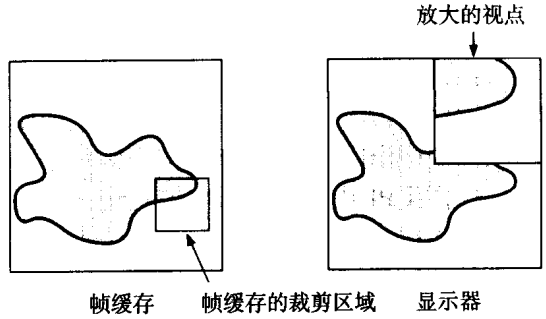


图4-29 帧缓存的一部分，由帧缓存的裁剪区域所指定，放大两倍之后叠放在未放大的帧缓存上

183

VLSI芯片已经实现了许多这样的图像显示功能，这些各种各样的变换是通用窗口管理操作需要的特定的例子。已经有了在视频控制器中使用的窗口管理芯片[SHIR86]。每个窗口在系统内存中有一块独立的像素图和一个描述窗口大小和原点的内存中的数据结构。显示扫描线的时候芯片知道哪个窗口是可见的，因此知道从哪个像素图中取得像素值。这些问题和其他高级的硬件问题将在第18章中讨论。

4.4.3 视频混合

视频控制器另一个有用的功能是视频混合。一幅定义在帧缓存中的图像可以和一个来自电视摄像头、录像机或者其他来源的视频信号混合一起形成一个复合图像。这种合成的例子在电视新闻、体育运动节目和天气预报中经常可以看到，图4-30描绘了通常的系统结构。

有两种类型的合成，一种是把图形图像置到视频图像中去，新闻报告员肩上显示的图表或者图形就是这个类型的典型例子，这种合成可以由硬件实现，它把帧缓存中指定的像素值作为

一个标志，指示应该显示视频信号，还是显示帧缓存中的信号，通常这个指定的像素值和帧缓存图像的背景颜色相对应，但是使用其他像素值能够获得感兴趣的效果。

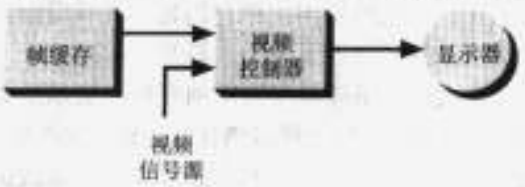


图4-30 把帧缓存的图像和视频信号源图像合成到一起的视频控制器

第二种合成是把视频图像放置在帧缓存图像的上面，就像天气预报员站在全屏幕的天气图前面那样，天气预报员实际上是站在一个背景前，背景的颜色（通常为蓝色）用来控制合成：当视频信号是蓝色时，显示帧缓存图像，否则显示视频图像，只要天气预报员不穿蓝色衬衫或者系蓝色领带，这种技术会很好地工作。

4.5 随机扫描显示处理器

图4-31演示了一个典型的随机（向量）显示系统，它一般和4.3.2节中基于显示处理器的光栅显示系统类似。当然刷新显示的时候没有像素图，显示处理器也没有扫描转换算法使用的本地内存，因为这个功能通常使用可编程逻辑阵列或者微代码来实现。

184

随机扫描图形显示处理器经常被称为显示处理单元（display processing unit，DPU），或者称为图形控制器。DPU有一个指令集和一个指令寄存器，并和任何计算机一样经过经典的取指、解码和执行周期。因为没有像素图，显示处理器必须每秒钟执行30~60次程序才能避免闪烁。DPU执行的程序存放在主存中，可以由通用CPU和DPU共享。

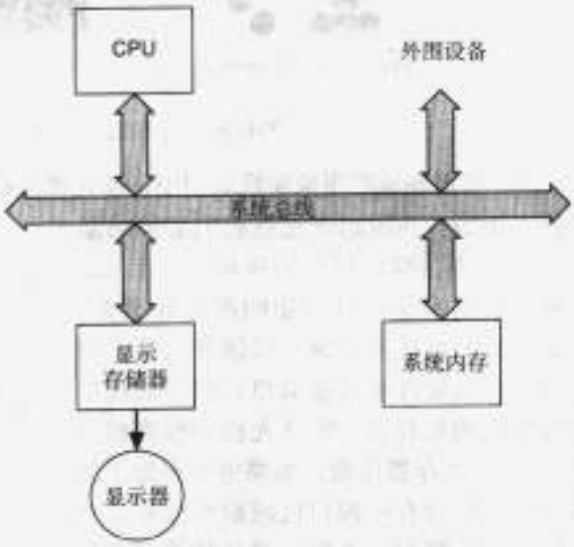


图4-31 随机显示系统的结构

应用程序和图形子程序包也驻留在主存中，在通用CPU中执行，图形包生成一个DPU指令的显示程序并告诉DPU从哪里开始执行程序，然后DPU异步地执行这个显示程序，直到图形包让它停止。程序结束处的JUMP指令把控制转回到程序的开始处，这样不用CPU干预，显示就可以继续刷新。

图4-32展示了一个简单随机扫描DPU的指令集和助记符，处理器有X、Y寄存器和一个指令计数器，指令定义为16位字长，LD指令中的R/A（relative/absolute）修饰符指明下面的地址是按照11位的相对地址对待还是按照10位的绝对地址对待。前一种情况下，11位的值加到X或者Y寄存器上，后一种情况下，10位的值替换寄存器中的内容。（相对移动需要使用11位是因为可以让2的补码表示的值在 - 1024到 + 1023的范围内。）

JUMP指令使用同样的R/A修饰符，只是修改的是指令计数器，从而影响控制流程的改变，SM、SP和SL指令提供了轮廓线、散布图等等的紧凑表示。

185

图4-33是一段简单的DPU程序，以汇编语言的风格写成，使用了许多指令。注意正方形和菱形是如何绘制的：第一个move指令是绝对的，而其他的是相对的，这样可以帮助在屏幕上拖动对象。如果不使用相对指令，拖动就需要修改用于显示对象的所有指令中的坐标值。最后

一条指令跳回到程序的开始。因为由L修饰符指定设置了帧锁定位，跳转的执行要延缓直到30 Hz时钟的下一次触发，这样可以允许DPU以30 Hz的频率刷新，但是却避免了短小程序更频繁的刷新，那样会烧坏荧光物质。

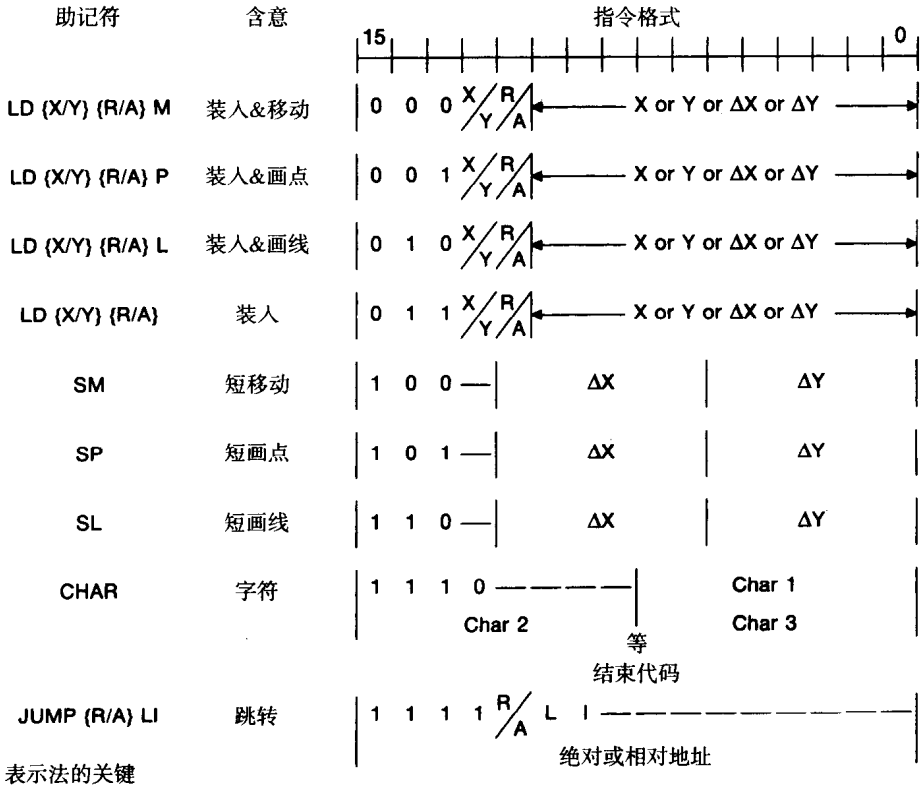
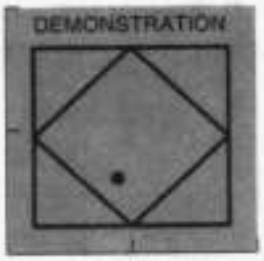


图4-32 随机扫描显示系统的指令集

注意，设置了这个指令后，只需要一个load命令（助记符为LD）就可以绘制或者垂直、水平移动图像。因为其他坐标都是固定的。但是间接的移动需要两个load命令，这两个命令可以先x再y，也可以先y然后x，两个中的第二个通常指定移动、画点或者画线操作。字符命令（助记符为CHAR）后面跟随一个字符串，最后以一个非显示字符终止代码结束。 186

DPU指令和通用计算机中使用的指令集有两个主要的区别。除了JUMP指令以外，其他所有的指令都是特定用途的，寄存器可以装入或者加上某个数，但是不能保存结果，寄存器值控制CRT电子束的位置。第二个区别是，除了JUMP指令以外，所有数据都是立即数，即它们是指令的一部分。LDXA 100意味着“把100这个值装入X寄存器”，而不是像计算机指令中那样“把地址为100的内存中的内容装入到X寄存器”。在第18章中描述的一些更高级的DPU中已经去掉了这个限制。

一般的随机处理器和光栅处理器指令集只有很小的差别，随机处理器没有区域填充、位操作和查找表的命令，但是因为有指令计数器，随机显示处理器有控制转移的命令，随机扫描显示器可以在比光栅扫描显示器更高的分辨率下工作，能够绘制出平滑的没有锯齿边的直线，最快的随机显示器可以在一个刷新周期里绘制大约100 000个短向量，允许极度复杂形状的实时动画。




```
SQUARE:   LDXA     100          准备好画正方形
          LDYAM    100          移到(100, 100)
          LDXRL    800          画线到(900, 100)
          LDYRL    700          画线到(900, 800)
          LDXRL   -800          画线到(100, 800)
          LDYRL   -700          画线到(100, 100)，正方形的起点
POINT:    LDXA     300
          LDYAP    450          在(300, 450)画点
DIAMOND:  LDXA     100
          LDYAM    450          移到(100, 450)
          LDXR     400
          LDYRL   -350          画线到(500, 100)
          LDXR     400
          LDYRL    350          画线到(900, 450)
          LDYR     350
          LDXRL   -400          画线到(500, 800)
          LDXR    -400
          LDYRL   -350          画线到(100, 450)，菱形的起点
TEXT:     LDXA     200
          LDYAM    900          移到(200, 900)画文字
          CHAR     'DEMONSTRATION !'   r是终止代码
          JUMPRL   SQUARE       重新生成图像，锁住帧
```

187

图4-33 随机扫描显示处理器的程序

4.6 用于操作者交互的输入设备

在本节中我们描述最普通的输入设备的工作。我们简单并高层次地讨论可用的设备是如何工作的。在第8章我们将讨论各种输入设备的优缺点，还描述一些更先进的设备。

我们的介绍是围绕着逻辑设备（logical device）的概念进行组织的，逻辑设备在第2章中已有介绍，并在第7章和第8章中进行更深入的讨论。有五种基本逻辑设备：定位设备，用来指明位置或者方向；拾取设备，用来选择显示的实体；定值设备，用来输入一个实数；键盘，用来输入字符串；选择设备，用来选择可能的行为或者选项集合中的一个或者多个。根据设备提供给应用程序的信息种类，逻辑设备的概念定义了这些输入设备的等价分类。

4.6.1 定位设备

1. 输入板

输入板（或数据输入板）是一个平板，其尺寸从6英寸×6英寸到48英寸×72英寸或者更大。它可以探测用户手中可移动触笔或者手持光标定位器的位置。图4-34显示了一个同时具有触笔和光标定位器（我们今后将主要只提及触笔，尽管讨论对两者相关）的小输入板。大部分输入板使用一种电感应的机制来确定触笔的位置，在一种设计中，网格宽度为1/4英寸或者1/2英寸的矩形网格线嵌在输入板的表面，沿金属线生成电磁脉冲系列，激励触笔中的线圈，感应出电信号，每个脉冲感应出来的电信号的强度可以用来确定触笔的位置。这个信号强度还粗略地用于确定触笔或者光标距离输入板有多远（“远”、“近”（比如说离数据板大约1/2英寸）或者

"触及")。如果答案是"近"或者"触及",显示器上显示出光标,给用户提供一个反馈。当触笔尖端在输入板上按压,或者手持光标定位器上的任一个按钮被按下时,会传送给计算机一个信号。



图4-34 带有触笔及光标定位器的数据输入板。触笔顶端有一个压敏开关,当触笔下压时,开关闭合。光标定位器有几个按键,用于输入命令,还有一个十字准线光标,用于精确地将输入板上的图形数字化后输入计算机。(Summa graphics公司提供。)

每秒钟内可以30~60次获取输入板的(x, y)位置、按钮状态以及靠近程度状态(如果靠近程度状态为"远"时,没有(x, y)位置可用)。在发生下列任何一种事件的时候一些输入板产生中断。

- 过了t个单位时间(输入板充当了触发光标位置更新的时钟)。
- 手持光标定位器或者触笔的移动超过了某特定距离d。这种距离间隔采样在数字化制图中很有用,它可以避免记录过多数目的点。
- 靠近程度状态发生改变(一些输入板在状态从"近"改变到"远"的时候不做报告)。
- 触笔尖端开关或者手持光标定位器的按钮被按下或者被释放。从输入板或者其他类似设备感知按钮按下和按钮弹起这两种事件是很重要的。比如,一个按钮按下的事件使一个现实的对象开始放大,而按钮弹起的事件则停止尺寸变化。
- 光标进入了输入板上某个特定的矩形区域。

与输入板或者其他定位设备相关的参数有分辨率(每英寸可区别的点的数目)、线性度、可重复性和尺寸或范围。这些参数对于把地图或者工程图数字化特别重要,当设备只用于定位屏幕光标时就没有那么多考虑了,因为用户可以通过显示器上的反馈来指导他的手的移动,并且一般显示器的分辨率要比便宜的输入板的分辨率低得多。

其他的输入板技术使用声音耦合或者电阻耦合。*声音输入板*(sonic tablet)利用声波将触笔耦合到麦克风,麦克风放置在数字化区域的外围。触笔尖端发出由电激励产生的声音,电激励产生的声音到达麦克风的时间差和触笔到每个麦克风的距离成正比(见图4-35),声音输入板在将厚书中的图画边界数字化的时候特别优越,在这种情况下使用普通的数字化仪,触笔和输入板距离太远,不能记录下精确的位置。声音输入板也不像其他输入板那样需要特定的工作区域。声音耦合在三维定位设备中也很有用,第8章将讨论此问题。



图4-35 2D声音图形输入板。从触笔发出的声波被输入板后面的两个麦克风接收

有一类电阻输入板使用电池供电的触笔，触笔发出高频无线电信号，输入板是一块覆盖着导体薄层的玻璃板，无线电信号在导体薄层中引起电压，输入板边缘的信号强度和输入板边缘到触笔的距离成反比，可以利用它来计算触笔的位置。另一种类似的输入板使用拉伸在CRT显示面上的电阻聚合体网格[SUNF86]。触笔在网格上引起电压，由网格上电压的下降可以确定触笔的位置。

大多数输入板触笔必须用一根导线与输入板控制器相连接。图4-36所示的电阻输入板是电池供电的，因为没有连线的阻碍，这种触笔很吸引人，另一方面，离开的时候就可以很方便地把它放在你的口袋里面了。



图4-36 电阻输入板。带有无连线的、由电池供电的触笔和光标定位器，而且有三个按钮。（由Kurta公司提供。）

有几种输入板是透明的，在数字化X射线片和照片底片时可以用背光照亮，也可以直接安装在CRT的前面，这时用电阻输入板特别适合，因为它可以按照CRT的形状弯曲。

188, 190

2. 鼠标

鼠标是小型的手持设备，它在平面上的相对移动可以测量出来。鼠标之间的不同之处在于鼠标按钮的数量和测量相对运动的方法。各类鼠标之间的其他重要区别在8.1.7节中讨论。机械鼠标底部滚轮的移动被转换成用于确定移动的方向和数量的数字值。光学鼠标在专门的鼠标衬板上移动，衬板上交替着亮线和暗线的网格，鼠标底部的发光二极管把光线直射到衬板上，光线反射回来被鼠标底部的探测器所感应。当鼠标移动时，每次穿过暗线，反射光线都会被中断，产生的脉冲数量和穿过的暗线数量相等，用来向计算机报告鼠标的移动。

因为鼠标是相对设备，它可以拿起来，移动，然后再放下来，而不改变所报告的位置（这样的一系列操作常被称为“抚摸”鼠标）。鼠标的相对特征意味着计算机必须保存“当前鼠标位置”，鼠标移动的时候会改变它。

3. 跟踪球

图4-37显示了跟踪球的一种，它经常被描述成上下颠倒的机械鼠标。在套中自由旋转的跟踪球的运动由电位计或者轴编码器感应，用户通常用手掌挨着跟踪球旋转它。各种开关安装在跟踪球上手指够得到的地方，开关的用法和鼠标或者输入板手持光标定位器的按钮类似。



图4-37 跟踪球带有几个相近的开关。（由Measurement Systems公司提供。）

4. 游戏杆

游戏杆（图4-38）可以左右移动或者上下移动，也是使用电位计来感应游戏杆的移动，经常用弹簧来使游戏杆恢复到原来的中心位置。一些游戏杆，包括图示的这种，有第三个自由度：杆可以顺时针或逆时针扭动。图4-39所示的等轴游戏杆（isometric joystick）是刚性的：轴上的张力测量表测量施加在轴上的力所引起的微小偏转。

191

使用游戏杆直接控制屏幕光标的绝对位置是非常困难的，因为（通常）较短的轴上微小的移动会放大成光标5倍或者10倍的移动。这会使屏幕光标的移动急促跳动，很难快速精确地定

位。因此游戏杆经常用来控制光标移动的速率而不是绝对光标位置。这意味着屏幕光标的当前位置以游戏杆确定的速率改变。



图4-38 带第三个自由度的游戏杆，这种游戏杆可以做顺时针及逆时针扭动。（由Measurement Systems, Inc.提供。）



图4-39 等轴游戏杆。（由Measurement Systems, Inc.提供。）

游戏开关（joyswitch）是游戏杆的一个变体，在家庭或者街道的计算机游戏中可以看到。它的杆可以向8个方向移动：上、下、左、右以及四个对角方向。有微小的开关来检测游戏杆向8个方向的哪一个方向移动。 192

5. 触摸板

鼠标、跟踪球和游戏杆都要占据工作台空间，而触摸板允许用户使用手指直接指向屏幕并在屏幕上移动光标。不同的触摸板使用了几种不同的技术。低分辨率的触摸板（每个方向上有10~50个可分解的位置）使用一系列的红外线发光二极管和光传感器（光敏二极管或者光敏晶体管）在显示区域形成看不见的光线网格，触摸屏幕会打断一条到两条光线束，因此通过被打断光线束就可以确定手指的位置。如果两条平行的光线束被打断，则认为手指处在这两条光线的中间，如果只有一条光线被打断，则认为手指处在这条光线上。

电容耦合的触摸板可以在每个方向上提供100个可分解的位置，当用户接触涂有导体膜的玻璃板时，电路从导体膜阻抗的改变来探测接触的位置[INTE85]。

一种高分辨率的触摸板（大约每个方向上有500个可分解的位置）使用声纳风格的探测。交替沿水平方向和垂直方向传播的高频声波脉冲引入到玻璃板的边缘，手指接触玻璃会使得部分声波反射到它的源头，可以通过发射声波和接收到发射回来的声波之间的时间间隔来计算手指到边缘的距离。另外一种高分辨率的触摸板使用两层透明物质，它们之间的间隔很小，一层覆盖着导体材料而另外一层覆盖着电阻材料。手指的轻压使两层接触到一起，可以测量到阻抗亚层上电压的下降，然后计算出接触位置的坐标。应用这种方法的低分辨率触摸板使用嵌在透明材料中的导体条或者细导线。触摸板技术已经用来给键盘制造小定位板。

触摸板最重要的参数是分辨率、激活所必需的压力（对光线触摸板这个不是问题）和透明度（这个对光线触摸板也不是问题）。对一些技术来说，一个重要的问题是视差：如果触摸板离显示器1/2英寸远，那么用户接触的位置是用户的眼睛和显示器上目标点对齐的位置，而不是接触板上直接垂直于目标点的位置。

用户习惯于得到可感知的反馈，但是触摸板并没有提供反馈。因此其他形式的立即反馈就很重要，如听得到的声音或者特定目标或位置的高光显示。

6. 光笔

光笔在交互式计算机图形学历史上很早就开发出来的，光笔实际上是错误命名，它探测光

脉冲，而不是像它的名字暗示的那样发射光线。光笔探测到光栅显示器上的光脉冲时引发的事件可以用来保存视频控制器的 X、Y寄存器的值并中断计算机。通过读出保存的这两个值，图形包可以确定光笔探测到的这个像素的坐标。光笔不能报告一个全黑的点的坐标，所以要使用特定的技术来使光笔能够指明任意位置：在一个帧时间里在通常图像的位置处显示一片深蓝色的区域。

193 和向量显示器一起使用的光笔充当的是拾取设备而不是定位设备。当光笔探测到光线时，DPU停止运行，且CPU被中断，然后CPU读出DPU指令地址寄存器的内容，从而确定当中断发生的时候正在绘制的是哪个输出图元。和光栅显示器一起工作时，需要一种称为*光笔跟踪*的特定技术来标明一个简单向量中的位置[FOLE82]。

光笔已经是老化的技术，用途狭窄，如果不进行合适的调整，有时候光笔会探测到错误的目标，如荧光灯或者其他附近的图形图元（比如邻近的字符）而不能探测到有意探测的目标。当需要使用好几个小时的时候，对没有经验的用户而言光笔是很累人的，因为每次使用它都要拿起，指向目标，然后放下。

4.6.2 键盘设备

*数字字母键盘*是原始的文本输入设备，许多种不同的技术被用来探测键的按下，包括机械接触开关，容积变化以及磁场耦合等。键盘设备最重要的功能特征是它创造一个和按下的键惟一对应的码（ASCII、EBCDIC等），有时候在数字字母键盘上需要允许和弦（一次按下几个键），让有经验的用户可以迅速地使用许多不同的命令。通常在标准*编码键盘*上这是不可能的，因为每次击键它返回的是一个ASCII码，如果两个键同时按下时它什么也不返回（除非附加的键是Shift，Ctrl或者其他特殊的键）。相反，*未编码键盘*返回同时按下的所有键的标志，因而允许和弦。

4.6.3 定值设备

大多数提供标量值的定值设备是基于电位计的，就像立体声音响中的音量和声调控制一样。定值设备通常是旋转电位计（刻度盘），电位计一般8个或10个一组安装在一起，如图4-40所示。简单的旋转电位计可以旋转330°，这可能提供不了足够的范围和分辨率，连续旋转的电位计可以自由地朝两个方向旋转，因此在范围上没有限制，线性电位计显然是有限制的，它很少在图形系统中使用。



图4-40 有8个旋转电位计的面板。位于每个旋钮上方的发光二极管的读数可用来标识各旋钮，或给出当前的设置值。（由Evans和Sutherland Computer公司提供。）

4.6.4 选择设备

*功能键*是最普通的选择设备，有时候它们制造成独立的单元，但更通常的情况是和键盘集成在一起。其他的选择设备是*按钮*，在许多输入板的手持光标上或者鼠标上都有。选择设备通常用来为图形程序输入命令或者菜单选项。专用的系统可以使用贴有永久标签的功能键，标签也可以是可替换的或者说“软”的。功能键可以在按钮旁边或者按键上包含一个小LCD或者LED显示，如图4-41所示。另外一种选择是把按钮安装在显示器的边角上，以便按钮标签在显示器上显示出来，正好挨着实际的按钮。



图4-41 键盘中带发光二极管（LED）显示的功能键（由Evans和Sutherland Computer公司提供。）

4.7 图像扫描仪

尽管数据输入板可以手工地数字化现有的线框图，但是这是一个缓慢而乏味的过程，不适用于复杂的图，而且它还不能数字化半色调的图像，图像扫描仪提供了有效的解决方法，一个电视摄像头和一个数字帧捕捉器的结合是不昂贵的方法，它可以获得中等分辨率（1000 × 1000，多种亮度级别）的黑白或彩色光栅图像。慢扫描电荷耦合器件（CCD）电视摄像头可以在30s内创建一幅2000 × 2000的图像。更便宜的方法是使用扫描头，扫描头由感光单元的网格组成，它安装在打印机的打印头上，它以每英寸80个单元的分辨率扫描图像。但是这些扫描分辨率是高品质的出版工作所不能接受的。出版中使用的是照片扫描仪（photo scanner），照片安装在一个旋转的鼓上，一束精确校准的光线被引导到照片上，反射光被光电管所测量。对于照相底片，透射光由鼓内的光电管测量，鼓是透明的。当鼓旋转时，光源缓慢地从一端移动到另一端，从而对整个照片进行了光栅扫描（图4-42）。扫描彩色照片时，扫描过程要进行多趟，每次在光电管前面使用不同的滤镜把各种颜色分离出来。最高分辨率的扫描仪使用激光光源，其分辨率高于2000单元/英寸。

194 ↓ 195



图4-42 照片扫描仪。光源沿鼓轴偏转，测量出偏转的光量

另一类扫描仪使用一细长条的CCD（叫作CCD阵列）。让图画从CCD阵列下面经过就可以将它数字化，根据所需要的分辨率来加快或减慢图画的移动。因此花1到2分钟走一趟，就可以数字化一幅大图。CCD阵列的分辨率为200~1000单元/英寸，低于照片扫描仪的分辨率，图4-43显示了这样的扫描仪。

线框图使用上面描述的任何一种方法都可以很容易地扫描，困难的地方是从扫描结果的一大堆像素中提取出有意义的信息来，向量化（vectorize）就是从光栅图像中提取直线、字符以及其他几何图元的过程。这个任务需要合适的算法而不是扫描硬件，向量化本质上是一个图像处理的问题，它分为以下几步：首先使用取阈值和边缘增强来整理光栅图像，消除图像中的污迹和污点，填补裂缝。然后使用特征提取算法把相邻的都为“打开”的像素合并在一起形成几何图元，如直线。在复杂度的第二级，模式识别算法用来把简单的图元合并成圆弧、字母、符号等等，可能需

要用户交互来消除由于中断的线条、暗污迹和许多邻近的直线交叉而引起的歧义性。



图4-43 液晶显示扫描仪。扫描头（上部）里的镜头和LCD线性阵列左右移动，扫描一幅照片或物体。（由Truvel公司提供照片。）

更困难的问题是将几何图元的集合组织成有意义的数据结构，“当给CAD或者地形学（地图绘制）应用程序作输入时，无组织的线条集合没有太大用处”。图画中表示高级的几何结构需要识别，因此，定义一块国土边界的轮廓线应该组织成多边形图元，代表圆弧的圆心的“+”要和圆弧本身组织在一起。这些问题已经有了部分解决。尽管算法在不断地改善，当情况困难时，商业系统还要依靠用户干预。

习题

4.1 一个使用18英寸宽纸的静电绘图仪，每个方向上的分辨率是200单元/英寸，走纸速度是3英寸/秒，要使纸全速移动，问每秒钟要提供多少位？

4.2 如果长余辉荧光物质降低停闪频率，为什么不例行公事地使用它们？

4.3 编写一个在光栅显示器上显示测试图案的程序，必须提供三种不同的图案：(1) 宽度为1的水平线，分别间隔0、1、2、3个像素；(2) 宽度为1的垂直线，分别间隔0、1、2、3个像素；(3) 一个像素的点的网格，网格间隔为5个像素。每个图案都要能用白色、红色、绿色、蓝色以及交替颜色带来显示。显示图案时你所观察到的东西与光栅分辨率的讨论如何联系到一起？

4.4 准备一个关于大屏幕显示器技术的报告。

4.5 假设每8个像素压缩到一个字节中，字节能以100 000字节/秒的速度传输并解压缩，装入一个512 × 512 × 1的位图需要多少时间？1024 × 1280 × 1的位图呢？

4.6 设计一个硬件单元的逻辑，完成二维光栅地址到字节地址加上字节中位地址的转换工作。给该单元的输入如下：(1) (x, y)，光栅地址；(2) *base*，第0位包含光栅地址（0, 0）的内存字节的地址；(3) x_{max}，最大光栅x地址（最小为0）。单元的输出如下：(1) *byte*，其中某位中包含光栅地址（x, y）的字节的地址；(2) *bit*，在*byte*中包含（x, y）的位号。如果$x_{max}+1$是2的幂时可以做什么简化？

4.7 编写一个光栅拷贝操作的程序，从一幅位图拷贝到散布在几个内存页的虚拟内存区。你的显示器上的硬件光栅拷贝只能工作在物理地址空间，所以你必须以多次调用移动建立散布

写入条件，每一个包含在目的中的逻辑页调用一次。 196 ~ 197

4.8 为4.3.2节中的简单光栅显示指令集设计有效的指令编码，所谓“有效”是指“用在每一条指令的位数最少”。

4.9 使用4.3.2节中简单光栅显示的指令集编写程序段来完成以下工作：

a. 双缓冲一个移动的对象，你不需要实际编写移动和绘制对象的代码，只要标明它们在整个程序代码序列中的放置位置。

b. 绘制一个饼图，输入为数据值的列表，饼图的每一片使用不同的颜色，选择你认为在一起看上去不错的颜色。

c. 显示从左到右移动的水平排成一行的圆形的动画，每个圆的半径为5个单元，圆形圆心距离15个单元，使用4.4.1节中讨论的动画技术，在动画中的每一步，第四个圆形都应该是可见的。

d. 写一段程序，使用RasterOP来在屏幕上拖动一个25 × 25的图标，使用位图中屏幕以外的区域来存储图标和当前被图标遮挡的位图部分。

4.10 在n条扫描线，每条扫描线m个像素，每秒钟显示r个周期的光栅扫描中，有一定数量的时间里没有显示图像：每条扫描线出现一次的水平回扫时间t_h和每帧出现一次的垂直回扫时间t_v。

a. 推导出没有图像显示的时间所占百分比的表达式，记住交错和非交错情况下的表达式不一样，因为交错（隔行）扫描每帧两个部分中的每个部分都需要垂直回扫时间。

b. 用下列数据评估表达式，数据引自[WHIT84]。

可见区 像素数 × 行数	刷新频率 Hz	是否隔行	垂直回扫时间 (ms)	水平回扫时间 (ms)
640 × 485	30	yes	1271	11
1280 × 1024	30	yes	1250	7
640 × 485	60	no	1250	7
1280 × 1024	60	no	600	4

4.11 开发一个视频控制器的设计策略，使它能在480 × x/2的显示器上显示960 × x的图像，其中x是水平分辨率。（这是从高分辨率显示系统驱动视频录像机所需要的设备。）

4.12 光栅显示每像素4位，查找表每项12位（红、绿、蓝每种颜色4位）。考虑到像素图为了装入两幅图像而分成的四个平面：图像1在高两位平面，图像0在低两位平面。对每幅图像中每两位的值，颜色分配如下：00 = 红，01 = 绿，10 = 蓝，11 = 白。

a. 演示如何装入查找表从而只显示图像0。

b. 演示如何装入查找表从而只显示图像1。

注意，查找表需要双缓冲的两相（见4.4.1节）。

4.13 给定一个16项、每项12位的查找表和两幅2位的图像，演示如何装入查找表以实现重叠溶解，表达式为$image1*t + image0*(1 - t)$，$t$的值分别为0.0、0.25和0.5。查找表的颜色分配和习题4.12题一样（见第17章）。

4.14 4.4.1节中讨论了两种动画显示方法，讨论各自的优点和缺点。 198

4.15 使用16位的字重新设计图4-32中的简单DPU指令集，假定所有操作码长度相等。

4.16 考虑在所有可以检测的只是单个像素时在光栅显示器上拾取一条直线的手段（如使用光笔）。 199

第5章 几何变换

本章将介绍图形学中基本的二维和三维几何变换。在这里讨论的平移、缩放和旋转变换对于许多图形学的应用都非常基本，在后面的各章中还要经常出现。

变换被应用程序直接使用，或者用在图形子程序包中。一个城市规划应用程序使用平移变换将表示建筑或树木的符号放在合适的位置上，用旋转变换调整符号的朝向，用缩放变换改变符号的大小。总之，许多图形应用程序在绘图时使用几何变换改变物体（也称为符号或模板）的位置、朝向和尺寸。在第6章，三维旋转、平移和缩放变换将被用做生成三维物体的二维投影过程的一部分。在第7章中，我们将看到当今的图形软件包如何使用变换作为其实现的一部分并且使应用程序能够使用它们。

5.1 二维变换

我们通过给点的坐标增加平移量来实现将(x, y)平面上的点平移到新的位置。对于每一个点$P(x, y)$，在与x轴平行的方向上平移dx且在与y轴平行的方向上平移dy得到新点$P'(x', y')$，可以表示为

$$x' = x + d_x, \qquad y' = y + d_y \tag{5-1}$$

如果我们定义列向量

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, T = \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{5-2}$$

201

那么公式(5-1)可以被表示成更加简洁的形式

$$P' = P + T \tag{5-3}$$

对物体上的每个点应用公式(5-1)，我们可以平移该物体。因为直线段由无限个点组成，所以这一过程将持续无限长的时间。幸运的是，我们只需要平移直线段的端点，然后在平移后的端点之间绘制新的直线段就可以平移直线段上的全部点；这对于缩放（拉伸）和旋转变换也是正确的。图5-1显示平移房子的轮廓线的效果，平移量为$(3, -4)$。

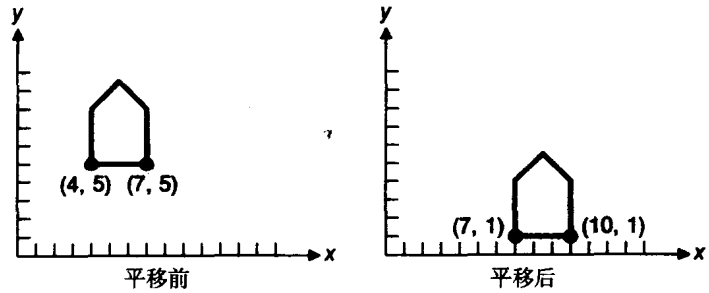


图5-1 房子的平移

通过以下乘法运算，点可以实现在x轴方向缩放（拉伸）s_x且在y轴方向缩放s_y到新的位置：

$$x' = s_x \cdot x, \qquad y' = s_y \cdot y \tag{5-4}$$

用矩阵的形式表示为

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \text{ 或 } P' = S \cdot P \tag{5-5}$$

其中S是公式(5-5)中的矩阵。

在图5-2中，房子在x轴方向被缩小为原来的1/2，在y轴方向缩小为原来的1/4。注意，缩放变换是相对于原点的：房子变小，更靠近原点。如果缩放因子比1大，房子会变大而且远离原点。相对于某一点而非原点的缩放变换将在5.3节中加以讨论。房子的比例已被改变：因为使用不同的缩放因子，$s_x \neq s_y$。使用均匀缩放，即$s_x = s_y$，房子的比例不受影响。

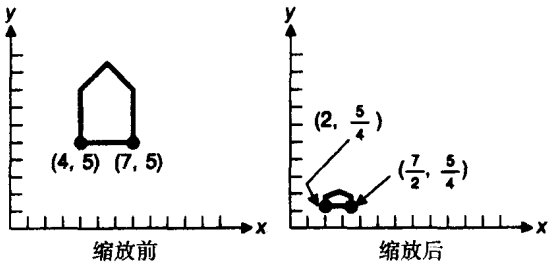


图5-2 房子的缩放，缩放是非均匀的，并且房子改变了位置

202

点可以绕原点旋转θ角。旋转在数学上被定义为

$$x' = x \cdot \cos\theta - y \cdot \sin\theta, \quad y' = x \cdot \sin\theta + y \cdot \cos\theta \tag{5-6}$$

用矩阵的形式表示为

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \text{ 或 } P' = R \cdot P \tag{5-7}$$

其中R为公式(5-7)中的旋转矩阵。图5-3显示旋转45°的房子。与缩放变换相似，旋转是绕原点进行的：绕任意点的旋转在5.3节中介绍。

正的旋转角度是从x轴向y轴逆时针方向测量。对于负的旋转角度（顺时针方向），恒等式$\cos(-\theta) = \cos\theta$和$\sin(-\theta) = -\sin\theta$用于修改公式(5-6)和公式(5-7)。

公式(5-6)很容易从图5-4中提取出，在图5-4中一个角度为θ的旋转将$P(x, y)$变换为$P'(x', y')$。因为旋转绕原点进行，所以从原点到P和P'点的距离相等，在图中标记为r。通过简单的三角学，我们发现

$$x = r \cdot \cos\phi, \; y = r \cdot \sin\phi \tag{5-8}$$

且

$$\begin{aligned} x' &= r \cdot \cos(\theta + \phi) = r \cdot \cos\phi \cdot \cos\theta - r \cdot \sin\phi \cdot \sin\theta \\ y' &= r \cdot \sin(\theta + \phi) = r \cdot \cos\phi \cdot \sin\theta + r \cdot \sin\phi \cdot \cos\theta \end{aligned} \tag{5-9}$$

将公式(5-8)代入公式(5-9)得到公式(5-6)。

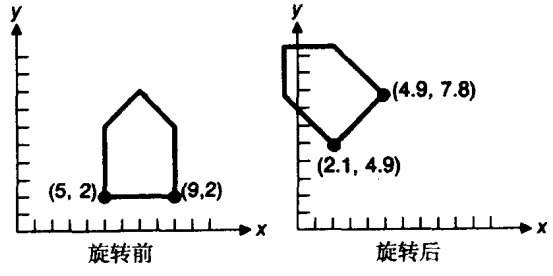


图5-3 房子的旋转，而且房子改变了位置

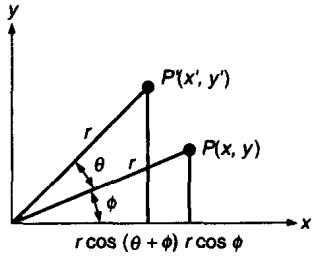


图5-4 旋转方程的推导

5.2 齐次坐标和二维变换的矩阵表示

平移、缩放和旋转的矩阵表示分别为

$$P' = T + P \tag{5-3}$$
$$P' = S \cdot P \tag{5-5}$$
$$P' = R \cdot P \tag{5-7}$$

可是，平移的处理方法（加法）不同于缩放和旋转（乘法）。我们更愿意用一种一致的方式处理三种变换，这样它们可以很容易地结合起来。

如果点被表示为齐次坐标（homogeneous coordinates），则这三种变换都可以用乘法处理。齐次坐标首先从几何中发展起来[MAXW46；MAXW51]，随后被应用于图形学中[ROBE65；BLIN77b；BLIN78a]。大量的图形子程序包和显示处理器使用齐次坐标和齐次变换。

在齐次坐标中，我们为每个点增加第三维坐标。每个点被表示为一个三元组(x, y, W)，而不是一对数(x, y)。与此同时，我们说两个齐次坐标(x, y, W)和(x', y', W')表示同一个点，当且仅当其中的一个是另一个的倍数。因此，(2, 3, 6)和(4, 6, 12)是用不同的三元组表示的同一个点。也就是说，每个点有许多不同的齐次坐标表示。同时，至少有一个齐次坐标值为非零：即(0,0,0)是不允许的。如果坐标W非零，我们可以用它去除齐次坐标：(x, y, W)与$(x/W, y/W, 1)$表示同一个点。当W非零时，我们通常做这个除法，数x/W和y/W被称为齐次点的笛卡儿坐标。$W=0$的点被称为无穷远点，在我们的讨论中不经常出现。

坐标三元组通常表示三维空间中的点，但是在这里我们使用它们表示二维空间中的点。两者之间的联系是：如果我们取所有表示同一个点的三元组（也就是说，所有形式为（tx, ty, tW）的三元组，其中$t \neq 0$），我们得到三维空间中的一条直线。因此，每一个齐次点表示三维空间中的一条直线。如果我们齐次化（homogenize）点（除以W），我们得到一个形式如$(x, y, 1)$的点。所以，齐次化的点形成一个被等式$W=1$定义的(x, y, W)空间中的平面。图5-5显示了这种关系。无穷远点不在该平面上。

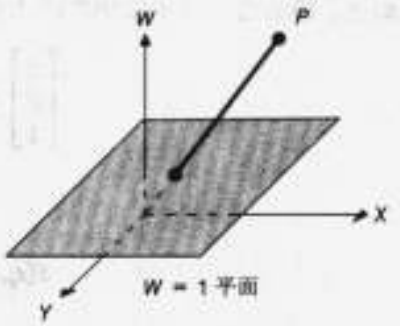


图5-5 XYW齐次坐标空间，带有$W=1$平面和投影到$W=1$平面上的点$P(X, Y, W)$

203
↓
204

因为现在点表示为三个元素的列向量，所以用于乘以一个点向量来产生另一个点向量的变换矩阵必须是3×3的矩阵。在齐次坐标的3×3矩阵形式中平移公式(5-1)为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-10}$$

我们提醒读者注意，一些图形学的教科书，包括[FOLE82]，使用左乘行向量矩阵的习惯，而不是右乘列向量。从一种习惯到另一种习惯变换矩阵必须被转置，正如行向量和列向量互为转置：

$$(P \cdot M)^T = M^T \cdot P^T$$

公式(5-10)可以被表示为另外一种形式

$$P' = T(d_x, d_y) \cdot P \tag{5-11}$$

其中

$$T(d_x, d_y) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-12}$$

如果一个点被平移$T(d_{x1}, d_{y1})$到达P'点，然后又被平移$T(d_{x2}, d_{y2})$到P''会发生什么呢？凭直觉我们期望结果是一个净平移$T(d_{x1}+d_{x2}, d_{y1}+d_{y2})$。为了确认这种直觉，我们从已知入手：

$$P' = T(d_{x1}, d_{y1}) \cdot P, \tag{5-13}$$

$$P'' = T(d_{x2}, d_{y2}) \cdot P' \tag{5-14}$$

现在，将公式(5-13)代入公式(5-14)，我们得到

$$P'' = T(d_{x2}, d_{y2}) \cdot (T(d_{x1}, d_{y1}) \cdot P) = (T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})) \cdot P \tag{5-15}$$

矩阵乘积$T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})$为

$$\begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_{x1}+d_{x2} \\ 0 & 1 & d_{y1}+d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \tag{5-16}$$

净平移确实是$T(d_{x1}+d_{x2}, d_{y1}+d_{y2})$。矩阵乘积被不同地称为$T(d_{x1}, d_{y1})$和$T(d_{x2}, d_{y2})$的复合（compounding）、连接（catenation）、串联（concatenation）或合成（composition）。这里，我们将规范地使用术语合成。

类似地，缩放公式(5-4)以矩阵形式表示为

205

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-17}$$

定义

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-18}$$

我们有

$$P' = S(s_x, s_y) \cdot P \tag{5-19}$$

如同净平移是加法，我们希望连续缩放是乘法。已知

$$P' = S(s_{x1}, s_{y1}) \cdot P, \tag{5-20}$$

$$P'' = S(s_{x2}, s_{y2}) \cdot P' \tag{5-21}$$

那么，将公式(5-20)代入公式(5-21)，我们得到

$$P'' = S(s_{x2}, s_{y2}) \cdot (S(s_{x1}, s_{y1}) \cdot P) = (S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})) \cdot P \tag{5-22}$$

矩阵乘积$S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})$为

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-23}$$

因此，连续的缩放变换也确实是乘法。

最后，旋转公式(5-6)可以表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-24}$$

令

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-25}$$

我们有

$$P' = R(\theta) \cdot P \tag{5-26}$$

两个连续旋转是加法的证明留为习题5.2。

在公式(5-25)的左上角的2×2子矩阵中，将矩阵的两行的每一行都看成向量。这两个向量有如下三个特性：

1) 每个都是单位向量。

2) 两个向量正交（它们的点积为0）。

3) 第一个向量和第二个向量分别旋转$R(\theta)$，将得到x轴和y轴的正方向（因为条件1和条件2的存在，这一性质等价于子矩阵的行列式为1）。 206

前两个特性对2×2子矩阵的列向量也成立。两个方向则是向量沿着x轴和y轴正方向旋转到的方向。这些特性表明当我们知道想要经过旋转得到的效果时，有两种有用的提取旋转矩阵的方法。一个具有这些特性的矩阵被称为特殊正交矩阵。

一个变换矩阵的形式如下：

$$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-27}$$

其中左上角2×2子矩阵是正交的，该变换矩阵保持角度和长度不变。也就是说，一个单位正方形仍然保持为单位正方形，既不会变成具有单位边长的菱形，也不会变成非单位边长的长方形。这样的变换也被称为刚体变换（rigid-body transformation），因为变换后的物体在任何情况下都不会扭曲。任意的旋转和平移矩阵序列都将产生这种形式的矩阵。

任意的旋转、平移和缩放矩阵序列的乘积如何呢？它们被称为仿射变换（affine transformation），并且具有保持直线平行性的特性，但是不保持长度和角度不变。图5-6显示对一个单位正方形施加 -45°角的旋转和非均匀缩放后的结果。很明显角度或长度在序列中都发生变化，但是平行线仍然平行。进一步的旋转、缩放和平移操作都不会导致平行线不平行。$R(\theta)$、$S(s_x, s_y)$和$T(d_x, d_y)$也是仿射变换。

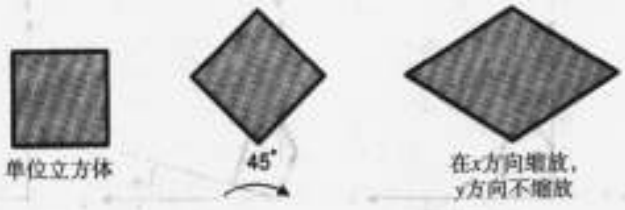


图5-6 单位立方体被旋转 -45°，并且被非均匀缩放。结果是单位立方体的仿射空间，在这个空间中保持了线的平行性，但角度和长度都没有保持

另一种类型的基本变换错切变换（shear transformation）也是仿射变换。二维错切变换分为两类：沿x轴错切和沿y轴错切。图5-7显示沿每个轴错切正方形的效果。该操作用矩阵表示为

$$SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-28}$$

错切矩阵中项a是比例常数。注意，乘积$SH_x[x\ y\ 1]^T$等于$[x+ay\ y\ 1]^T$，清楚地显示在x方向上的比例变化是y的函数。

207

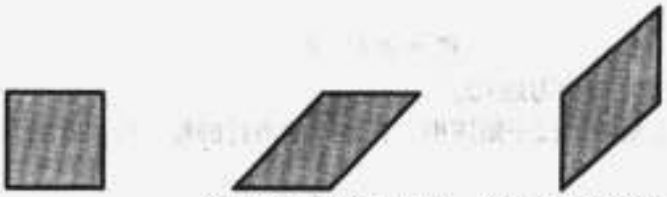


图5-7 应用于单位立方体的图元错切操作。在每种情况中，斜线的长度现在都大于1

类似地，矩阵

$$SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5-29}$$

沿y轴错切。

5.3 二维变换的合成

合成的观点在前面的部分已经介绍过。这里，我们使用合成将基本的R、S和T矩阵结合起来，以产生所需的一般的结果。合成变换的基本目的是通过对一个点施加单个合成后的变换来获得效率，而不是一个接一个地应用一系列变换。

考虑物体绕某个任意点P_1旋转。因为我们仅知道如何绕原点旋转，我们将原来（困难的）的问题转换为三个分开（容易的）的问题。因此，为了绕P_1旋转，我们需要三个基本变换的序列：

1) 平移使得P_1位于原点。

2) 旋转。

3) 平移使得P_1回到原来的位置。

这个序列如图5-8所示，在图5-8中房子绕$P_1(x_1, y_1)$旋转。首先平移$(-x_1, -y_1)$，反之后面反向平移(x_1, y_1)。最终结果与仅施加旋转操作不同。

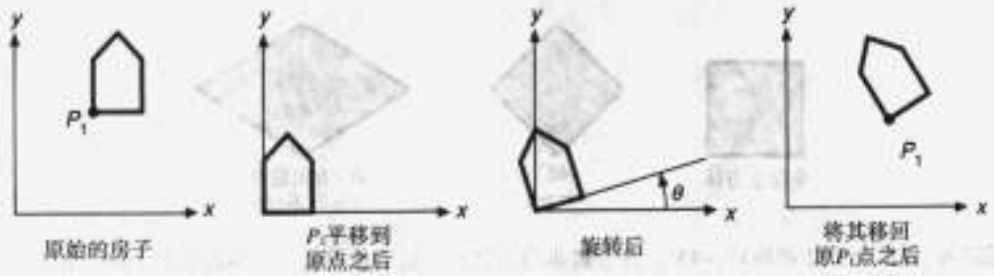


图5-8 房子绕着点P_1以角度θ旋转

净变换（net transformation）为

$$T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_1(1-\cos\theta) + y_1\sin\theta \\ \sin\theta & \cos\theta & y_1(1-\cos\theta) - x_1\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{5-30}$$

类似的方法用于相对任意点P_1缩放物体。首先，平移使得P_1到达原点，然后进行缩放，最后平移回到P_1。在这种情况下，净变换为

$$T(x_1, y_1) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & x_1(1-s_x) \\ 0 & s_y & y_1(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \tag{5-31}$$

假设我们希望缩放、旋转和设置房子的位置如图5-9所示，并以P_1为旋转和缩放的中心。变换序列为将P_1平移到原点，进行缩放和旋转，然后从原点平移到放置房子的新位置P_2。记录这个变换的数据结构可能包括缩放因子、旋转角度和平移量，以及变换的次序，或者简单地记录合成变换矩阵：

$$T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) \tag{5-32}$$

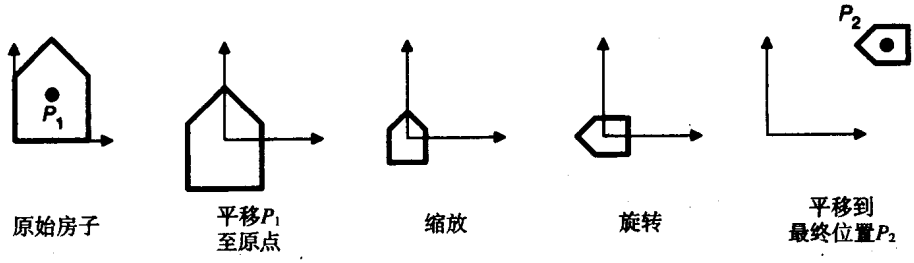


图5-9 房子绕P_1旋转，并将在P_1放置的房子放在P_2位置

如果M_1和M_2表示基本的平移、缩放或旋转，什么时候$M_1 \cdot M_2 = M_2 \cdot M_1$？也就是说，什么时候$M_1$和$M_2$可交换？通常，矩阵乘法当然是不可交换的。但是，很容易证明，在下列特殊情况下，可交换性成立：

208
~
209

M_1	M_2
平移	平移
缩放	缩放
旋转	旋转
缩放（当$s_x = s_y$时）	旋转

在这些情况下，我们不需要关心矩阵操作的次序。

5.4 窗口到视口的变换

一些图形软件包允许程序员在浮点世界坐标系下指定输出图元的坐标，使用对应用程序有意义的单位：埃，微米，米，英里，光年等等。使用术语“世界”是因为应用程序正在表示一个正在被交互创建或向用户显示的世界。

假定输出图元在世界坐标系下指定，则必须告诉图形子程序包如何将世界坐标系映射到屏幕坐标系（我们使用特殊的术语屏幕坐标以便将这部分与SRGP联系起来，但是因为可能使用硬拷贝输出设备，在这种情况下术语设备坐标更加准确）。我们可以通过由程序员向图形软件包提供变换矩阵来实现这种映射。另一种方式是让程序员在世界坐标中指定一个矩形区域，称为世界坐标系窗口，另外在屏幕坐标上指定一个对应的矩形区域，称为视口，世界坐标系窗口将被映射到视口中。将窗口映射到视口的变换被应用到世界坐标系中的所有输出图元，从而将它们映射到屏幕坐标系中。图5-10显示了这一概念。如图5-10所示，如果窗口和视口高度宽度比不同，就会出现一个非均匀缩放变换。如果应用程序改变窗口或视口，那么新绘制到屏幕上的输出图元会受到变化的影响，而现存的输出图元则不受这种变化的影响。

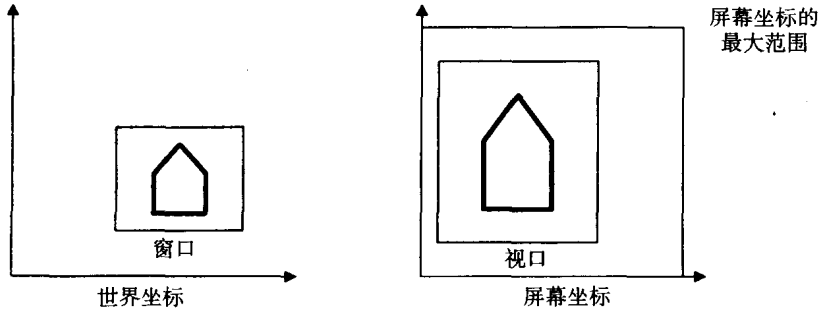


图5-10 世界坐标中的窗口和屏幕坐标中的视口确定了一个映射，该映射作用于世界坐标中的所有图元

210 修饰语世界坐标与窗口一起使用，以强调我们不是在讨论一个窗口管理器窗口，后者是一个不同的更新的概念，却不幸具有相同的名字。当不会产生歧义时，我们就丢弃这个修饰语。

如果SRGP提供世界坐标下的输出图元，视口将出现在当前的画布上，默认为画布0，即屏幕。应用程序能够在任何时刻改变窗口或视口，在这种情况下随后指定的输出图元将被施加新的变换。如果这种变化包括一个不同的视口，那么新的输出图元将被定位在画布上不同于原来输出图元的位置上，如图5-11所示。

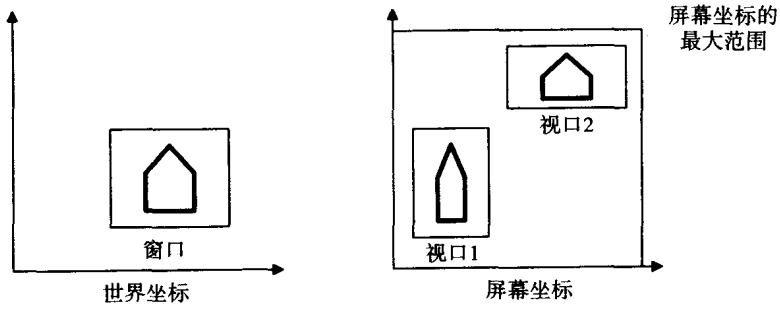


图5-11 以两个视口画输出图元的效果。描述房子的输出图元先被画在视口1，该视口变化成视口2，然后应用程序再次调用软件包来画输出图元

一个窗口管理器可能将SRGP的画布0映射为全屏窗口的一部分，在这种情况下并不是整个画布或者视口都必须可见。在第10章中，我们进一步讨论世界坐标系窗口、视口和窗口管理器窗口三者之间的关系。

给定窗口和视口，什么是将窗口从世界坐标系映射到屏幕坐标系下的视口的变换矩阵？这个矩阵可以由一个三步的变换合成求出，如图5-12所示。首先，用左上和右下角点表示的窗口被平移到世界坐标系的原点。第二步，窗口的尺寸被缩放成与视口尺寸相等。最后，用一个平移变换设置视口的位置。完整的矩阵M_{wv}为：

$$
\begin{aligned}
M_{wv} &= T(u_{min}, v_{min}) \cdot S\left(\frac{u_{max}-u_{min}}{x_{max}-x_{min}}, \frac{v_{max}-v_{min}}{y_{max}-y_{min}}\right) \cdot T(-x_{min}, -y_{min}) \\
&= \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & 0 \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & -x_{min} \cdot \frac{u_{max}-u_{min}}{x_{max}-x_{min}} + u_{min} \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & -y_{min} \cdot \frac{v_{max}-v_{min}}{y_{max}-y_{min}} + v_{min} \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{5-33}
$$

乘法$P = M_{wv}[x\ y\ 1]^T$给出期望的结果：

$$
P = \begin{bmatrix} (x - x_{min}) \cdot \frac{u_{max}-u_{min}}{x_{max}-x_{min}} + u_{min} & (y - y_{min}) \cdot \frac{v_{max}-v_{min}}{y_{max}-y_{min}} + v_{min} & 1 \end{bmatrix} \tag{5-34}
$$

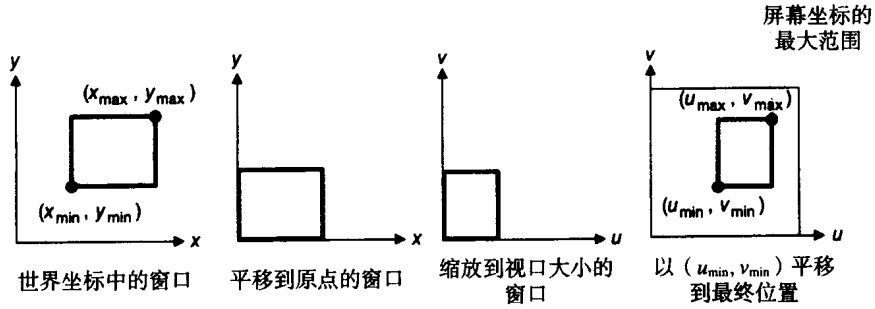


图5-12 变换世界坐标窗口到视口的步骤

许多图形软件包将窗口-视口变换与用窗口对输出图元的裁剪结合起来。裁剪的概念在第3章已经介绍过；图5-13显示了在窗口和视口环境（context）下的剪裁。

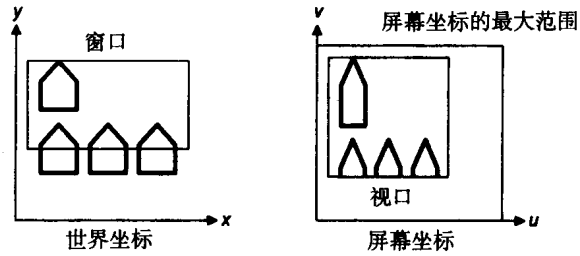


图5-13 世界坐标中的输出图元被窗口裁剪。保留的部分被显示在视口中

5.5 效率

R、S和T操作最普通的合成产生

形式如下的矩阵

211 ~ 212

$$M = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5-35}$$

左上角2×2子矩阵是一个合成的旋转和缩放矩阵，而t_x和t_y是合成的平移。用3×3矩阵乘以向量来计算$M \cdot P$需要九个乘法和六个加法。然而，公式(5-35)最后一行的固定结构将实际运算简化为

$$\begin{aligned} x' &= x \cdot r_{11} + y \cdot r_{12} + t_x \\ y' &= x \cdot r_{21} + y \cdot r_{22} + t_y \end{aligned} \tag{5-36}$$

将这一过程简化为四个乘法和四个加法——意义重大的加速，特别是当这个运算被应用到每张图上数以百计甚至千计的点时。所以，尽管3×3矩阵对合成二维变换很方便也很有用，但是通过利用最终矩阵的特殊结构我们能够在程序中最有效地利用它。一些硬件矩阵乘法器具有并行的加法器和乘法器，从而减少或消除了这个问题。

另一个注重效率的领域是生成物体的连续视图，例如一个在连续视图之间旋转一个很小角度的分子或飞机。如果每个视图可以被足够快地生成和显示（每个30~100μs），那么物体将呈现出动态地旋转。为了达到这一速度，我们必须尽可能快地变换物体上每个单独的点和直线。旋转公式（公式(5-6)）需要做四个乘法和两个加法。我们还可以减少操作的数量，通过认识到因为θ很小（只有几度），$\cos\theta$非常接近1。用这种近似的方法，公式(5-6)变为

$$x' = x - y\sin\theta, \qquad y' = x\sin\theta + y \tag{5-37}$$

仅需要两次乘法和两次加法。两次乘法的节省对于缺少硬件乘法器的计算机意义重大。

但是，公式(5-37)仅仅是正确值x'和y'的逼近：引入了小误差。每一次对新的x和y值应用该公式，误差都会变大。如果我们无限地重复使用该公式，误差将淹没正确的值，而且旋转的图像看起来就像随机画的直线的集合。

一个更好的逼近是在第二个公式中使用x'代替x：

$$\begin{aligned} x' &= x - y\sin\theta \\ y' &= x'\sin\theta + y = (x - y\sin\theta)\sin\theta + y = x\sin\theta + y(1 - \sin^2\theta) \end{aligned} \tag{5-38}$$

该公式比公式(5-37)逼近得更好，因为对应的2×2矩阵的行列式为1，这意味着用公式(5-38)变换的区域保持不变。应该注意的是当重复使用正确的旋转公式时，累积误差仍然会出现（见习题5.19）。

5.6 三维变换的矩阵表示

正如使用齐次坐标时二维变换可以表示为3×3的矩阵一样，如果我们也使用齐次坐标表示三
213 维空间上的一个点，三维变换能够表示成4×4的矩阵。因此，我们用(x, y, z, W)表示一个点而不是用(x, y, z)，其中如果一个四元组是另一个四元组的非零倍数，则两个四元组表示同一个点；四元组$(0,0,0,0)$是不允许的。与二维相同，点(x, y, z, W)的标准表示为$(x/W, y/W, z/W, 1)$，其中$W \neq 0$。将点变换为这种形式被称为*齐次化*，如前所述。同样，W坐标为零的点被称为*无穷远点*。齐次化也有一个几何解释。每个三维空间上的点被表示成通过四维空间原点的一条直线，这些点的齐次化表示形成一个四维空间的三维子空间，该子空间可以由单个方程$W = 1$定义。

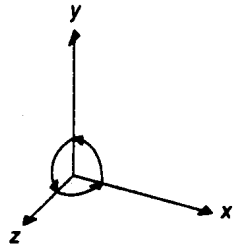
在本书中三维坐标系采用右手系，如图5-14所示。按照惯例，一个右手系的正向旋转是，当从一个坐标轴的正方向向原点看时，一个90°的逆时针旋转将一个轴的正方向变换为另一个

轴的正向。下面的表从这个惯例得出：

如果旋转轴是	则正的旋转方向是
x	y到z
y	z到x
z	x到y

图5-14描绘了这些正方向。读者需要注意，不是所有的图形学教科书都遵守这个惯例。

这里我们使用右手系统，因为它是标准的数学惯例，即使在图形学中考虑到添加在显示器表面的左手系统更加方便（见图5-15），因为一个左手系统给出一种自然的解释，即大的z值更加远离视点。注意到在左手系中，从坐标轴的正方向向原点看时正向旋转是顺时针。这种正向旋转的定义使得不论是右手还是左手系都可以使用本节给出的相同的旋转矩阵。从右手系统到左手系和相反的转换在5.8节中讨论。



（页的外面）

图5-14 右手坐标系

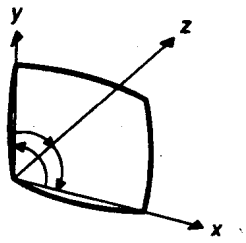


图5-15 左手坐标系，带有添加的显示屏幕

三维平移是二维平移的简单扩展：

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-39}$$

也就是说，$T(d_x, d_y, d_z) \cdot [x\ y\ z\ 1]^T = [x + d_x\ \ y + d_y\ \ z + d_z\ 1]^T$。

类似地，缩放扩展为：

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-40}$$

经核对，我们得到$S(s_x, s_y, s_z) \cdot [x\ y\ z\ 1]^T = [s_x \cdot x\ s_y \cdot y\ s_z \cdot z\ 1]^T$。

二维旋转公式(5-26)仅仅是绕z轴进行的三维旋转，表示为

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-41}$$

这很容易验证：对沿x轴的单位向量$[1\ 0\ 0\ 1]^T$进行90° 的旋转，得到沿y轴的单位向量$[0\ 1\ 0\ 1]^T$。计算乘积

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5-42}$$

得到预测的结果$[0\ 1\ 0\ 1]^T$。

绕x轴的旋转矩阵为

$$R_x(\theta)=\begin{bmatrix}1&0&0&0\\0&\cos\theta&-\sin\theta&0\\0&\sin\theta&\cos\theta&0\\0&0&0&1\end{bmatrix} \tag{5-43}$$

绕y轴的旋转矩阵为

$$R_y(\theta)=\begin{bmatrix}\cos\theta&0&\sin\theta&0\\0&1&0&0\\-\sin\theta&0&\cos\theta&0\\0&0&0&1\end{bmatrix} \tag{5-44}$$

214
~
215

$R_z(\theta)$、$R_x(\theta)$和$R_y(\theta)$的左上角3×3子矩阵的列（和行）是互相垂直的单位向量，并且子矩阵的行列式为1，这意味着这三个矩阵是特殊正交阵，如在5.2节中所述。同样，由任意的旋转序列所形成的左上3×3子矩阵也是特殊正交阵。记住，正交变换保持距离和角度不变。

所有这些矩阵都有逆（逆矩阵）。T的逆通过对d_x，d_y和d_z取负得到；对于S，其逆通过将s_x、s_y和s_z替换为它们的倒数得到；三个旋转矩阵的逆通过对旋转角度取负值得到。

任何正交矩阵B的逆恰好是它的转置：$B^{-1}=B^T$。事实上，转置不需要交换存储矩阵的数组的元素——只需要在存取数组时交换行列的下标即可。注意到这种求逆的方法与求R_x、R_y和R_z的逆对θ取负值的结果一致。

任意次的旋转、缩放和平移矩阵都可以乘到一起。其结果总是具有如下形式：

$$M=\begin{bmatrix}r_{11}&r_{12}&r_{13}&t_x\\r_{21}&r_{22}&r_{23}&t_y\\r_{31}&r_{32}&r_{33}&t_z\\0&0&0&1\end{bmatrix} \tag{5-45}$$

正如二维中的情况那样，3×3左上子矩阵R给出合计的旋转和缩放，同时T给出随后总的平移。一些计算效率通过显式地执行如下变换获得

$$\begin{bmatrix}x'\\y'\\z'\end{bmatrix}=R\cdot\begin{bmatrix}x\\y\\z\end{bmatrix}+T \tag{5-46}$$

其中R和T是公式(5-45)中的子矩阵。

与5.2节中的二维错切矩阵对应的是三维错切矩阵。(x,y)的错切为

$$SH_{xy}(sh_x,sh_y)=\begin{bmatrix}1&0&sh_x&0\\0&1&sh_y&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \tag{5-47}$$

对点$[x\ y\ z\ 1]^T$施加SH_{xy}，我们有$[x+sh_x\cdot z\ \ y+sh_y\cdot z\ \ z\ \ 1]^T$。沿$x$轴和$y$轴的错切具有相似的形式。

至此，我们一直集中于单个点的变换。由两个点定义的直线段通过变换其端点来实现整条直线段的变换。如果平面由三个点定义，则可以用相同的方法处理，但是通常平面用平面方程来定义，方程的系数必须用不同的方法变换。令平面用平面方程的系数列向量$N=[A\ B\ C\ D]^T$表示，那么一个平面被定义为满足方程$N\cdot P=0$的所有点，其中“$\cdot$”是向量点积且$P=[x\ y\ z\ 1]^T$。

216

这个点积得到熟悉的平面方程$Ax+By+Cz+D=0$，该方程也可以表示为平面方程系数的行向量与列向量P的乘积：$N^T\cdot P=0$。现在，假设我们通过某个矩阵M变换平面上所有的点P。为了使变换后的点保持$N^T\cdot P=0$，我们应该用某个（待定）矩阵Q对N进行变换，得到方程$(Q\cdot$

$N)^T \cdot M \cdot P = 0$。运用恒等式$(Q \cdot N)^T = N^T \cdot Q^T$，这种表示可以被依次重新写成$N^T \cdot Q^T \cdot M \cdot P = 0$。如果$Q^T \cdot M$是单位矩阵的倍数，则该等式成立。如果倍数是1，则有$Q^T = M^{-1}$，或$Q = (M^{-1})^T$。这意味着用M变换以后的平面系数的列向量N'为

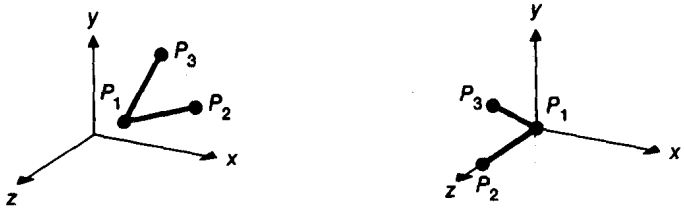
$$N' = (M^{-1})^T \cdot N \tag{5-48}$$

矩阵$(M^{-1})^T$一般不一定存在，因为M的行列式可能为0。如果M包括一个投影（我们可能想要研究平面上的透视投影的效果），这种情况就会发生。用M的余因子代替$(M^{-1})^T$是可能的，余因子用于根据克莱姆法则计算M的逆。更多的细节参见附录。

如果仅仅变换平面的法线（例如，在第16章中讨论的明暗处理的计算），并且M仅包括平移、旋转和均匀缩放矩阵的合成，那么数学更加简单。公式(5-48)中的N'可以被简化为$[A'\ B'\ C'\ 0]^T$。（如果有一个为0的W分量，该齐次点表示无穷远点，可以被看成一个方向。）

5.7 三维变换的合成

在本节中，我们用一个例子讨论如何合成三维变换矩阵，这个例子在6.4节还要用到。目标是将有向线段P_1P_2和P_1P_3从图5-16a的开始位置移到图5-16b的结束位置。所以，点P_1将被平移到原点，P_1P_2位于z轴的正方向上，P_1P_3位于y轴正向的(y, z)半平面上。直线的长度不受变换的影响。



a) 最初位置　　b) 最终位置

图5-16 将P_1, P_2和P_3从最初位置变换为它们的最终位置

这里给出两种完成所希望的变换的方法。第一种方法是将基本变换T、R_x、R_y和R_z合成起来。这种方法虽然有些烦琐，但是很容易用图例说明，而且理解这种方法会帮助我们建立对变换的理解。第二种方法利用前一节所描述的正交矩阵的性质，可以更简明地解释但是要抽象一些。 217

要用基本变换实现，我们还要将一个复杂的问题分解为简单一些的子问题。在这种情况下，希望的变换可以用四步完成：

1) 将P_1平移到原点。

2) 绕y轴旋转使得直线P_1P_2位于（y, z）平面上。

3) 绕x轴旋转使得直线P_1P_2落在z轴上。

4) 绕z轴旋转使得直线P_1P_3位于（y, z）平面上。

第一步：将P_1平移到原点

平移变换为

$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-49}$$

对P_1、P_2和P_3施加T变换得到

$$P'_1 = T(-x_1, -y_1, -z_1) \cdot P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5-50}$$

$$P'_2 = T(-x_1, -y_1, -z_1) \cdot P_2 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix} \tag{5-51}$$

$$P'_3 = T(-x_1, -y_1, -z_1) \cdot P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix} \tag{5-52}$$

第二步：绕y轴旋转

图5-17显示经过第一步后的P_1P_2，以及P_1P_2在（x,z）平面上的投影。旋转角度是$-(90-\theta)=\theta-90$。那么

$$\begin{aligned} \cos(\theta - 90) &= \sin\theta = \frac{z'_2}{D_1} = \frac{z_2 - z_1}{D_1} \\ \sin(\theta - 90) &= -\cos\theta = -\frac{x'_2}{D_1} = -\frac{x_2 - x_1}{D_1} \end{aligned} \tag{5-53}$$

其中

$$D_1 = \sqrt{(z'_2)^2 + (x'_2)^2} = \sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2} \tag{5-54}$$

当这些值被代入公式(5-44)中，我们得到

$$P''_2 = R_y(\theta - 90) \cdot P'_2 = [0 \quad y_2 - y_1 \quad D_1 \quad 1]^T \tag{5-55}$$

218 正如期望的那样，P_2"的x分量等于0，z分量等于长度D_1。

第三步：绕x轴旋转

图5-18显示经过第二步后的P_1P_2。旋转角度为ϕ，有

$$\cos\phi = \frac{z''_2}{D_2}, \ \sin\phi = \frac{y''_2}{D_2} \tag{5-56}$$

其中$D_2 = |P_1"P_2"|$，为直线$P_1"P_2"$的长度。但是$P_1"P_2"$的长度与P_1P_2相等，因为旋转和平移变换保持长度不变，所以

$$D_2 = |P''_1P''_2| = |P_1P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{5-57}$$

第三步旋转的结果为

$$\begin{aligned} P'''_2 &= R_x(\phi) \cdot P''_2 = R_x(\phi) \cdot R_y(\theta - 90) \cdot P'_2 \\ &= R_x(\phi) \cdot R_y(\theta - 90) \cdot T \cdot P_2 = [0 \quad 0 \quad |P_1P_2| \quad 1]^T \end{aligned} \tag{5-58}$$

也就是说，P_1P_2与z轴的正向一致。

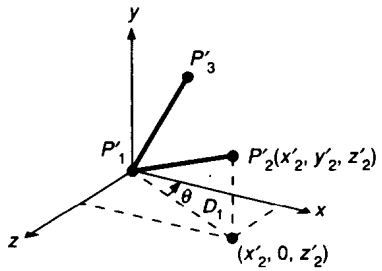


图5-17 绕y轴旋转：$P'_1P'_2$的投影（其长度为D_1）被旋转到x轴。角度θ表明绕y轴旋转的正方向：实际角度是$-(90-\theta)$

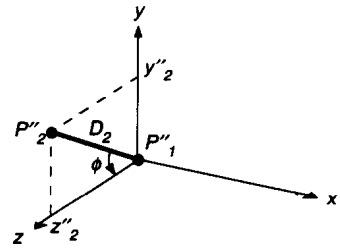


图5-18 绕x轴旋转：$P_1''P_2''$被以角度ϕ旋转到z轴，D_2是线段的长度。线段$P_1''P_3''$没有显示，因为它没有用于确定旋转的角度。两条线都按$R_x(\phi)$旋转

第四步：绕z轴旋转

图5-19显示经过第三步后的P_1P_2和P_1P_3，这时P_2'''在z轴上，P_3'''的位置为

$$P_3''' = [x_3''' \quad y_3''' \quad z_3''' \quad 1]^T = R_x(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) \cdot P_3 \tag{5-59}$$

219

旋转一个正的角度α，有

$$\cos\alpha = y_3'''/D_3, \quad \sin\alpha = x_3'''/D_3, \quad D_3 = \sqrt{x_3'''^2 + y_3'''^2} \tag{5-60}$$

第四步完成变换，结果如图5-16b所示。

合成矩阵

$$R_z(\alpha) \cdot R_x(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) = R \cdot T \tag{5-61}$$

为所要求的变换，其中$R = R_z(\alpha) \cdot R_x(\phi) \cdot R_y(\theta - 90)$。我们给读者留下一个问题，对$P_1$、$P_2$和$P_3$应用这个变换，以验证是否：$P_1$被平移到原点，$P_2$被平移到$z$轴正向，$P_3$被平移到$y$轴正向的$(y,z)$半平面。

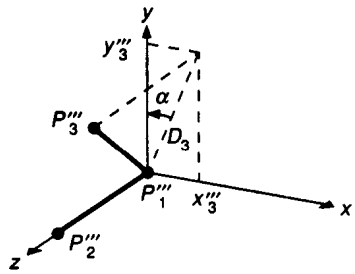


图5-19 绕z轴旋转：$P_1'P_3'$（其长度是D_3）的投影被旋转正α角到y轴，将线带到(y,z)平面。D_3是投影的长度

第二种获得矩阵R的方法是利用5.2节中讨论的正交矩阵的特性。回忆一下R的单位行向量旋转成基本坐标轴。为了描述方便用x, y和z代替公式(5-45)中的第二个下标。

$$R = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} \\ r_{1y} & r_{2y} & r_{3y} \\ r_{1z} & r_{2z} & r_{3z} \end{bmatrix} \tag{5-62}$$

因为R_z是沿着P_1P_2的单位向量，P_1P_2将被旋转到z轴正向，所以

$$R_z = [r_{1z} \quad r_{2z} \quad r_{3z}]^T = \frac{P_1P_2}{|P_1P_2|} \tag{5-63}$$

另外，单位向量R_x与P_1、P_2和P_3构成的平面垂直，并且将被旋转到x轴正向，所以R_x一定是平面上两个向量规格化的叉积：

$$R_x = [r_{1x} \quad r_{2x} \quad r_{3x}]^T = \frac{P_1P_3 \times P_1P_2}{|P_1P_3 \times P_1P_2|} \tag{5-64}$$

最后，

220

$$R_y = [r_{1y} \quad r_{2y} \quad r_{3y}]^{\mathrm{T}} = R_z \times R_x \tag{5-65}$$

将被旋转到y轴的正向。合成矩阵给出如下：

$$\begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot T(-x_1, -y_1, -z_1) = R \cdot T \tag{5-66}$$

其中R和T与公式(5-61)中的相同。图5-20显示单个的向量R_x、R_y和R_z。

让我们考虑另外一个例子。图5-21显示一架定义在x_p、y_p和z_p坐标系中并以原点为中心的飞机。我们想要对飞机做一个变换，使得飞机朝向由向量DOF（飞行方向）指定的方向，以P为中心，并且不倾斜，如图5-22所示。达到这个结果的变换包括一个使飞机朝向合适方向的旋转，接着是从原点到P点的平移。为了找到旋转矩阵，我们仅需要确定图5-22中x_p，y_p和z_p轴的朝向，确保这些方向是归一化的，然后作为旋转矩阵的列向量使用它们。

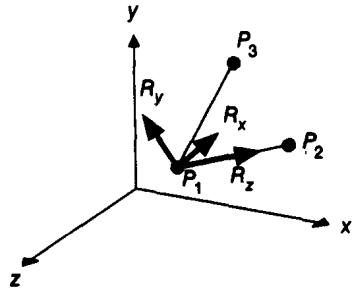


图5-20 单位向量R_x, R_y和R_z，它们被变换到主轴

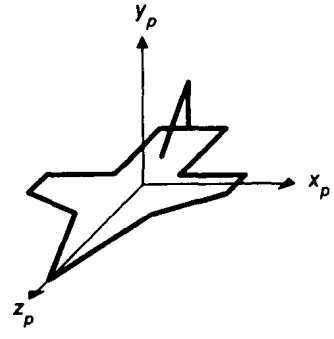


图5-21 (x_p, y_p, z_p)坐标系中的飞机

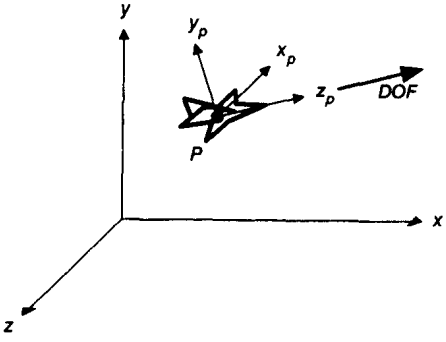


图5-22 图5-21的飞机定位在点P，并且头在DOF方向

z_p轴必须被变换到DOF的方向上，x_p轴则必须被变换为一个垂直DOF的水平向量——也就是说，在$y \times DOF$的方向上，即y和DOF的叉积。y_p方向由$z_p \times x_p = DOF \times (y \times DOF)$给出，即$z_p$和$x_p$的叉积；因此，旋转矩阵的三个列向量分别为归一化的向量$|y \times DOF|$、$|DOF \times (y \times DOF)|$和$|DOF|$：

$$R = \begin{bmatrix} |y \times DOF| & |DOF \times (y \times DOF)| & |DOF| & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-67}$$

DOF与y轴同向是一种退化的情况，因为此时的水平向量有无限种可能。这种退化也被反映到代数上，因为$y \times DOF$和$DOF \times (y \times DOF)$等于0。在这种特殊情况下，$R$不再是旋转矩阵。

5.8 坐标系的变换

我们一直在讨论当两个点集在同一个坐标系中时将属于一个物体的点集变换为另一个点集。

用这种方法，坐标系保持不变而物体相对于坐标原点被变换。考虑变换的另一种可选择并且等价的方式是坐标系的改变。这种观点在多个物体被结合时十分有用，其中每个物体都被定义在自己的局部坐标系中，而我们希望在一个全局坐标系中表示这些物体的坐标。这种情况在第7章中将要出现。

我们定义$M_{i\leftarrow j}$为一个变换，该变换将点在坐标系j中的表示转换为在坐标系i中的表示。

我们再定义$P^{(i)}$为一个点在坐标系i中的表示，$P^{(j)}$为点在坐标系j中的表示，$P^{(k)}$为点在坐标系k中的表示，那么，

$$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)} \quad 且 \quad P^{(j)} = M_{j\leftarrow k} \cdot P^{(k)} \tag{5-68}$$

221 ~ 222

替换$P^{(j)}$，我们得到

$$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)} = M_{i\leftarrow j} \cdot M_{j\leftarrow k} \cdot P^{(k)} = M_{i\leftarrow k} \cdot P^{(k)} \tag{5-69}$$

所以

$$M_{i\leftarrow k} = M_{i\leftarrow j} \cdot M_{j\leftarrow k} \tag{5-70}$$

图5-23显示四个不同的坐标系。通过观察我们看到从坐标系2到坐标系1的变换为$M_{1\leftarrow 2} = T(4, 2)$（通过观察找到这种变换不总是很简单——见附录）。相似地，$M_{2\leftarrow 3} = T(2, 3) \cdot S(0.5, 0.5)$，$M_{3\leftarrow 4} = T(6.7, 1.8) \cdot R(-45°)$。那么$M_{1\leftarrow 3} = M_{1\leftarrow 2} \cdot M_{2\leftarrow 3} = T(4, 2) \cdot T(2, 3) \cdot S(0.5, 0.5)$。这个图也显示了一个点在坐标系1到坐标系4中的坐标分别为$P^{(1)} = (10, 8)$，$P^{(2)} = (6, 6)$，$P^{(3)} = (8, 6)$和$P^{(4)} = (4, 2)$。很容易验证$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)}$，当$1 \leqslant i, j \leqslant 4$时。

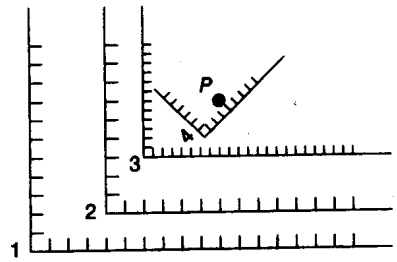


图5-23 点P和坐标系1, 2, 3, 4

我们也注意到$M_{i\leftarrow j} = M_{j\leftarrow i}{}^{-1}$。所以，$M_{2\leftarrow 1} = M_{1\leftarrow 2}{}^{-1} = T(-4, -2)$。因为$M_{1\leftarrow 3} = M_{1\leftarrow 2} \cdot M_{2\leftarrow 3}$，所以$M_{1\leftarrow 3}{}^{-1} = M_{2\leftarrow 3}{}^{-1} \cdot M_{1\leftarrow 2}{}^{-1} = M_{3\leftarrow 2} \cdot M_{2\leftarrow 1}$。

在5.6节中，我们讨论过左手坐标系和右手坐标系。将点在左手坐标系和右手坐标系之间转换的矩阵是它本身的逆，表示为

$$M_{R\leftarrow L} = M_{L\leftarrow R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-71}$$

在前面几节中使用的方法——在世界坐标系中定义所有的物体，然后将它们变换到希望的位置——暗含着有些不现实的概念，即所有的物体初始都一个接一个地定义在同一个世界坐标系下。一种更加自然的想法是认为每个物体定义在自己的坐标系中，然后通过在新的世界坐标系中重定义它的坐标来进行缩放、旋转和平移变换。在第二种观点中，我们会自然地想到撕开的纸片，每张纸片上有一个物体，而纸片在世界坐标系的平面上被缩小或拉伸、旋转或平移。我们当然也可以想像平面相对于每张纸片被缩小或拉伸、倾斜或者滑动。数学上说，所有这些观点都是等价的。

223

考虑简单的情况，将图5-8中定义房子的点集平移到原点。这个变换是$T(-x_1, -y_1)$。在图5-24中标记两个坐标系，我们看到将坐标系1映射成坐标系2的变换（即$M_{2\leftarrow 1}$）为$T(x_1, y_1)$，恰好是$T(-x_1, -y_1)^{-1}$。确实，通常的规则是在单个坐标系中对点集进行的变换恰好是对应的改变点被表示的坐标系的变换的逆。这种关系可以从图5-25中看出，也可以直接从图5-9中得到。在单个坐标系下表示的点的变换为

$$T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) \tag{5-32}$$

在图5-25中，坐标系变换正是

$$\begin{aligned} M_{5\leftarrow 1} &= M_{5\leftarrow 4}\, M_{4\leftarrow 3}\, M_{3\leftarrow 2}\, M_{2\leftarrow 1} \\ &= (T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1))^{-1} \\ &= T(x_1, y_1) \cdot S(s_x^{-1}, s_y^{-1}) \cdot R(-\theta) \cdot T(-x_2, -y_2) \end{aligned} \tag{5-72}$$

所以

$$P^{(5)} = M_{5\leftarrow 1} P^{(1)} = T(x_1, y_1) \cdot S(s_x^{-1}, s_y^{-1}) \cdot R(-\theta) \cdot T(-x_2, -y_2) \cdot P^{(1)} \tag{5-73}$$

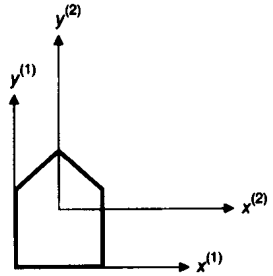


图5-24 房子和两个坐标系。房子上点的坐标被在两个坐标系中表示

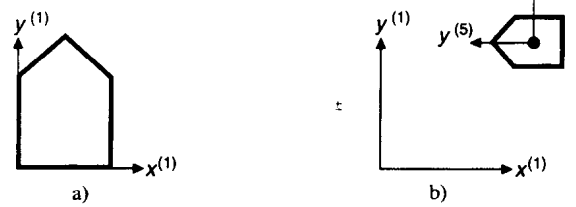


图5-25 原始房子a)在其坐标系中和在其坐标系中被变换的房子b)相对于原始坐标系

224

与改变坐标系相关的一个重要问题是我们如何改变变换。假设$Q^{(j)}$是坐标系j中的变换。举例说明，它可能是从前面几节中提取出的合成变换。假设我们想要找到一个在坐标系i中的变换$Q^{(i)}$，该变换将被应用到坐标系i中的点$P^{(i)}$上，并产生与$Q^{(j)}$被应用到坐标系j中相对应的点$P^{(j)}$相同的结果。这个等式被表示为$Q^{(i)} \cdot P^{(i)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot P^{(j)}$。将$P^{(i)} = M_{i\leftarrow j} \cdot P^{(j)}$代入，等式变为$Q^{(i)} \cdot M_{i\leftarrow j} \cdot P^{(j)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot P^{(j)}$。经过简化，我们有$Q^{(i)} = M_{i\leftarrow j} \cdot Q^{(j)} \cdot M_{i\leftarrow j}{}^{-1}$。

当用户在后面的局部坐标系中指定了子物体的额外信息时，改变坐标系的观点将十分有用。例如，如果在图5-26中三轮车的前轮绕z_{wh}坐标旋转，所有的轮子必须被恰当地旋转，而且我们需要知道三轮车作为一个整体如何在世界坐标系中移动。这个问题非常复杂，因为几个连续的坐标系变换同时发生。首先，三轮车和前轮坐标系在世界坐标系中具有初始的位置。当三轮车向前移动时，前轮绕车轮坐标系的z轴旋转，与此同时车轮坐标系和三轮车坐标系相对于世界坐标系移动。车轮坐标系和三轮车坐标系通过在x和z方向上随时间变化的平移加上绕z轴的旋转，与世界坐标系相关联。当手把被旋转时，车轮坐标系与三轮车坐标系则通过一个绕z轴随时间变化的旋转相互关联。（三轮车坐标系固定在车架上，而不是在手把上。）

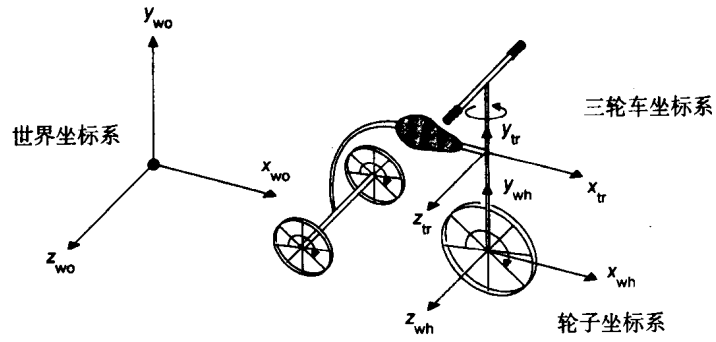


图5-26 带有三个坐标系的三轮车

为了使问题更加简单，我们假设车轮和三轮车坐标系的轴平行于世界坐标系的轴，而且车轮在与世界坐标系x轴平行的一条直线上运动。当车轮旋转α角度时，车轮上的一点P旋转路程αr，其中r是车轮的半径。因为车轮在地面之上，三轮车向前移动αr单位。所以，车轮上的边缘点P相对于初始车轮坐标系移动和旋转，产生一个连续经过平移距离αr和旋转角度α的最终效果。因此，在原始车轮坐标系中新的坐标P'为

$$P'^{(wh)} = T(\alpha r, 0, 0) \cdot R_z(\alpha) \cdot P^{(wh)} \tag{5-74}$$

225

并且它在新的（经过平移）车轮坐标系中的坐标仅由旋转给出

$$P'^{(wh')} = R_z(\alpha) \cdot P^{(wh)} \tag{5-75}$$

为了在世界坐标系中找到点$P^{(wo)}$和$P'^{(wo)}$，我们将车轮坐标系变换到世界坐标系：

$$P^{(wo)} = M_{wo\leftarrow wh} \cdot P^{(wh)} = M_{wo\leftarrow tr} \cdot M_{tr\leftarrow wh} \cdot P^{(wh)} \tag{5-76}$$

$M_{wo\leftarrow tr}$和$M_{tr\leftarrow wh}$是平移矩阵，由三轮车和车轮的初始位置给出。

$P'^{(wo)}$可以用公式(5-74)和公式(5-76)求得：

$$P'^{(wo)} = M_{wo\leftarrow wh} \cdot P'^{(wh)} = M_{wo\leftarrow wh} \cdot T(\alpha r, 0, 0) \cdot R_z(\alpha) \cdot P^{(wh)} \tag{5-77}$$

另外，我们认识到通过车轮坐标系的平移$M_{wo\leftarrow wh}$被变换为$M_{wo\leftarrow wh'}$，以一种不同的方式得到与公式(5-77)相同的结果：

$$P'^{(wo)} = M_{wo\leftarrow wh'} \cdot P'^{(wh')} = (M_{wo\leftarrow wh} \cdot M_{wh\leftarrow wh'}) \cdot (R_z(\alpha) \cdot P^{wh}) \tag{5-78}$$

因此，一般而言，通过从三轮车部件的运动方程应用合适的变换，我们可以从以前的值求出新的$M_{wo\leftarrow wh}$和$M_{tr'\leftarrow wh'}$。我们再对局部坐标系中更新后的点应用更新后的变换，得到世界坐标系中等价的点。我们将旋转三轮车的前轮来改变方向和计算后轮的旋转角度的问题留给读者，要用到车轮的半径和三轮车的轨迹。

习题

5.1 证明：通过变换一条直线的端点，然后在变换后的端点之间构造新的直线段，我们可以实现对直线段的变换。

5.2 证明：连续的二维旋转是加法：$R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$。

5.3 证明：如果$s_x = s_y$或$\theta = n\pi$，其中n为整数，二维旋转和缩放可交换，否则不可交换。

5.4 找到一种表示，描述式(5-37)中的累积误差与和增量旋转的操作次数之间的关系。对式(5-38)做相同的工作。

5.5 在你喜欢的计算机上写一个程序完成二维增量旋转。对于每个端点需要多长时间？与绝对二维旋转每个端点需要的时间进行比较。

5.6 一个包含N个端点的图形将被绕单个坐标轴动态地旋转。在你的计算机上乘法花费时间t_m；加法花费时间t_a。写出用式(5-37)、式(5-38)和式(5-7)旋转N个点所需时间的表达式。忽略控制步骤。现在以N作为变量，使用你的计算机的实际指令时间,计算上述表达式。

5.7 对点P_1、P_2和P_3应用5.7节中的变换，以验证这些点与预想的一样变换。

5.8 重做5.7节中的工作，假设$|P_1P_2| = 1$，$|P_1P_3| = 1$，并且P_1P_2和P_1P_3的方向余弦已知（直线的方向余弦是直线和x、y和z轴之间的夹角的余弦）。对于一条从原点到(x, y, z)的直线，方向余弦等于$(x/d, y/d, z/d)$，其中d是直线的长度。

226

5.9 齐次坐标具有吸引力的另一个原因是笛卡儿坐标系中的三维无穷远点可以在齐次坐标中被清楚地表示出来。这是如何完成的?

5.10 证明：式(5-61)与式(5-66)等价。

5.11 已知一个单位立方体，其中一个角点在(0, 0, 0)，另一个对角点在(1, 1, 1)，导出绕主对角线（从(0, 0, 0)到(1, 1, 1)）以逆时针方向旋转θ角所必需的变换。逆时针方向是沿着对角线向原点看。

5.12 假设窗口的底边被从x轴旋转θ角，就像在Core System[GSPC79]。什么是窗口到视口的映射？通过对窗口的每个角点应用变换来验证你的答案，确认这些角点被变换到合适的视口角点。

5.13 考虑一条从右手坐标系的原点到点$P(x, y, z)$的直线。用三种不同的方式找到将直线旋转到z轴正方向所需的变换矩阵，并用代数操作显示每一种情况中P确实到达z轴。对于每一种方法，计算旋转角度的正弦和余弦。

a. 绕x轴旋转到(x, y)平面，然后绕y轴旋转到z轴。

b. 绕y轴旋转到(y, z)平面，然后绕x轴旋转到z轴。

c. 绕z轴旋转到(x, z)平面，然后绕y轴旋转到z轴。

5.14 将一个物体在方向余弦为（α, β, γ）的方向上以缩放因子S进行缩放。导出变换矩阵。

5.15 找到绕任意方向旋转θ角的4×4变换矩阵，该方向由方向向量$U = (u_x, u_y, u_z)$给出。通过合成变换矩阵完成这个任务，首先用一个旋转变换$R_z(\theta)$将U旋转到z轴（记为M），然后用M^{-1}合成这个结果。结果应该是

$$\begin{bmatrix} u_x^2 + \cos\theta(1 - u_x^2) & u_x u_y(1 - \cos\theta) - u_z\sin\theta & u_x u_z(1 - \cos\theta) + u_y\sin\theta & 0 \\ u_x u_y(1 - \cos\theta) + u_z\sin\theta & u_y^2 + \cos\theta(1 - u_y^2) & u_y u_z(1 - \cos\theta) - u_x\sin\theta & 0 \\ u_x u_z(1 - \cos\theta) - u_y\sin\theta & u_y u_z(1 - \cos\theta) + u_x\sin\theta & u_z^2 + \cos\theta(1 - u_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5-79}$$

验证：如果U是一个主轴，矩阵变为R_x、R_y或R_z。基于向量操作的推导参见[FAUX79]。注意，同时对U和θ取负值结果保持不变。解释其原因。

5.16 证明在5.2节结尾处描述的$R(\theta)$的性质。

5.17 将5.4节中讨论的增量旋转扩展到三维，形成绕任意轴旋转的合成操作。

5.18 假设物体被旋转不产生令人心烦地慢的最低速率是30秒360°。也假设，为了平滑，旋转步长至多4°。利用习题5.6的结果确定多少点可以使用绝对旋转进行旋转，多少点可以使用增量旋转进行旋转。

5.19 假设你正在通过多次增量旋转来创建一个旋转物体的界面，使用“Spin X”、“Spin Y”和“Spin Z”按钮。每一次按其中一个按钮，则用绕指定轴轻微旋转的矩阵与当前旋转矩阵的乘积替换当前旋转矩阵。尽管数学上这个想法是正确的，但是在实际上累积的浮点舍入误差将产生点被不正确变换的结果。证明：通过对矩阵的列向量应用Gram-Schmidt过程（参见A.3.6节），你可以将新矩阵转换回正交矩阵。再解释一下，如果变换矩阵已经正交，为什么应用这个过程不改变这个矩阵。

227

第6章　三维空间的观察

在三维空间中的观察过程根本就比在二维空间中的观察过程复杂得多。对于二维情况，我们仅需要在二维空间指定一个窗口并在二维观察表面给定一个视口。从概念上讲，可以使用窗口对世界的物体进行裁剪，然后变换到视口进行显示。三维观察过程的额外的复杂性一部分是由被添加的维引起的，还有一部分是由显示设备仅是二维的这一事实引起的。

三维物体和二维显示之间的不匹配问题是通过引入投影来解决的，而投影就是把三维物体变换到二维的投影平面上去。这一章的大部分内容将讨论什么是投影，它们的数学表示是什么，以及如何在当前的图形子程序包PHIGS[ANSI88]中使用它们。它们的使用也在第7章深入讨论。

在三维观察过程中，我们需要在世界坐标系中设定一个视见体(view volume),在投影平面上给出一个投影，并在观察表面上给定一个视口。从概念上讲，三维世界坐标系中的物体被三维视见体裁剪，之后进行投影。视见体落在投影平面上的投影的内容（称为窗口）再被变换(映射)到视口进行显示。图6-1表示了三维观察过程的概念模型，这是提供给许多三维图形子程序包用户的模型。和二维观察的情况一样，可以采用各种各样的策略实现一个观察过程。这些策略不需要与概念模型一样，只要结果是模型定义的就可以了。一个典型的线框图的实现策略在6.5节描述。对于实现可见面确定和明暗处理的图形系统，应用了有一点不同的流水线，这些将在第16章讨论。



图6-1　三维观察过程的概念模型

6.1　投影

一般说来，投影是把 n维坐标系中的点变换成小于n维的坐标系中的点。实际上，计算机图形学很长一段时间被用来通过投影n维物体到二维进行观察来研究n维物体[NOLL67]。这里，我们将限于讨论从三维到二维投影。一个三维物体的投影是用从投影中心发射出来的许多直的投影射线（称之为投影线（projector））来定义的，这些投影线通过物体的每一点和投影平面相交，形成该物体的投影。图6-2显示同一条直线的两个不同的投影。幸运的是，一条直线段的投影本身仍是一条直线段，因此，实际上只需对直线段的端点做投影变换。

我们在这里处理的这类投影被称为平面几何投影，因为投影是到平面上而不是到曲面上，并且投影线是直线而不是曲线。许多制图学方面的投影可能或者是非平面的或者是非几何的。类似地，Omnimax电影格式要求一个非几何的投影[MAX82]。



图6-2 a) 直线AB和它的透视投影$A'B'$，b) 直线AB和它的平行投影$A'B'$。投影线AA'和BB'是平行的

平面几何投影（此后简称为投影）可以分为两种基本类型：透视投影和平行投影。它们的
230 区别在于投影中心与投影平面之间的关系不同。如果投影中心到投影平面之间的距离是有限的，那么投影是透视投影；如果它们之间的距离是无限的，则是平行投影。图6-2说明了这两种情况。平行投影之所以这样命名，是因为投影中心到投影平面之间的距离是无限的，所以投影线是互相平行的。在定义透视投影的时候，我们显式地给定它的投影中心；对于平行投影，我们给出它的投影方向。投影中心是一个点，具有形式$(x, y, z, 1)$的齐次坐标。由于投影方向是一个向量（也就是点间坐标之差），可以通过两个点相减获得：$d=(x, y, z, 1)-(x', y', z', 1)=(a, b, c, 0)$。所以方向和无穷远点以一种自然的方式对应。中心是一个无限点的透视投影成为平行投影。

透视投影的视觉效果类似于照相系统和人的视觉系统，称为透视缩小效应（perspective foreshortening）：一个物体的透视投影的大小与物体到投影中心的距离成反比。因此，尽管物体的透视投影倾向于看起来真实，但这对于记录物体的精确形状和尺寸并不特别有用，因为不能从物体的投影获得距离，而且仅保留那些与投影平面相平行的物体表面的角度，平行线的投影一般并不是平行的。

平行投影产生的视图真实性较差，这是因为虽然沿着每一个坐标轴都有不同的固定透视缩小系数，但仍然不具备透视缩小效应。平行投影可以被用于精确的测量，并且平行线确实保持平行。与透视投影相同，也保留了那些与投影平面平行的物体各表面的角度。

不同类型的透视投影和平行投影在Carlbom和Paciorek的综述论文中被详细讨论和说明[CARL78]。在下面的两个小节里我们总结被更广泛使用的投影的基本定义和特征；然后转向6.2节以理解对PHIGS各种投影怎样被实际定义。

6.1.1 透视投影

任何一束不平行于投影平面的平行线的透视投影将会聚成一个点，称为灭点（vanishing point）。在三维空间中，平行线仅在无限远处相聚，于是灭点可被看成无限远处一个点的投影。当然也存在着一个灭点的无穷远点，它是每一条直线指向的方向的无穷远。

如果直线的集合平行于三个主轴之一，这时的灭点称为轴灭点（axis vanishing point）。至多存在三个这样的轴灭点，对应于投影平面切割的主轴的数目。例如，如果投影平面仅切割z轴（因此，z轴是投影平面的法线），则只在z轴上有一个主灭点，因为此时平行于x或y坐标轴的直线也平行于投影平面，所以没有灭点。

透视投影是按主灭点的数量来分类的，所以也取决于投影平面切割坐标轴的数目。图6-3
231 显示一个立方体的两种不同的一点透视投影。很清楚的是，它们是一点投影，因为平行x和y轴的直线并不会聚；仅有平行于z轴的直线才会聚于一点。图6-4显示具有一些投影线的一点透视的结构，且其投影平面仅切割z轴。

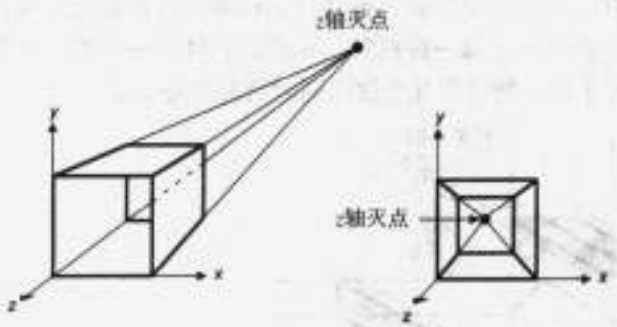


图6-3 立方体投影到切割z轴的平面的一点透视投影，显示垂直于投影平面的直线的灭点

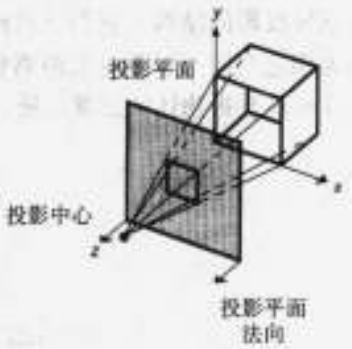


图6-4 立方体投影到切割z轴的平面的一点透视投影的结构。投影平面法线平行于z轴。（摘自[CARL78]，Association for Computing Machinery, Inc.；授权使用。）

图6-5 显示两点透视的结构。注意，在投影中平行于y轴的直线并不会聚于一点。两点透视通常应用于建筑、工程、工业设计和广告绘画等方面。三点透视应用得不多，因为它们并不比两点透视添加更多的真实性。

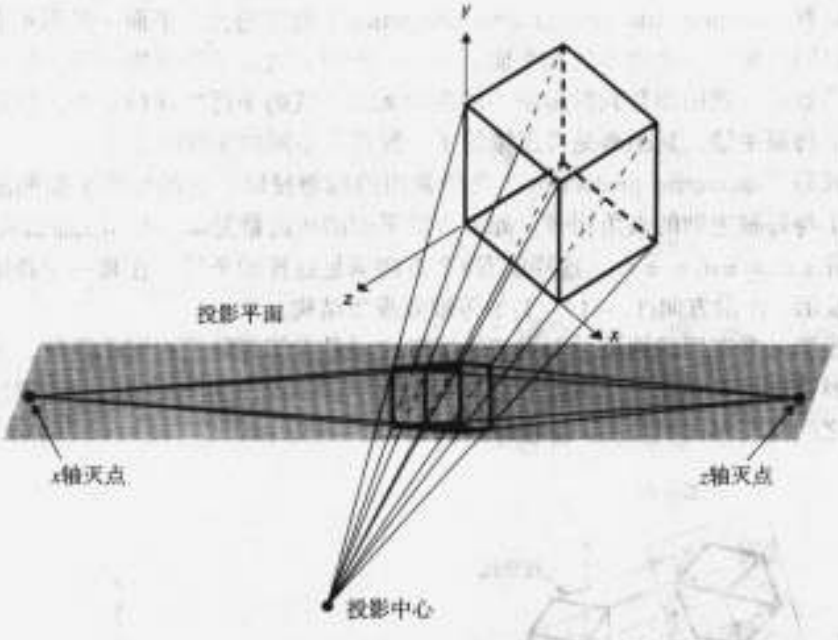


图6-5 立方体的两点透视投影。投影平面切割x轴和z轴

6.1.2 平行投影

平行投影根据投影方向和投影平面法线之间的关系分为两种类型。在正平行投影（orthographic parallel projection）中，这些方向是相同的（或彼此相反），所以投影方向是投影平面的法向。对于斜平行投影（oblique parallel projection），这些方向是不同的。

最常见的正投影类型是：前视图（主视图）投影，顶视图（俯视图，也叫平面视图）投影和侧视图投影。在所有这些投影中，投影平面垂直于一根主轴，所以该轴即为投影方向。图6-6

显示这三种投影的结构，它们通常被用来在工程制图中绘制机械部件、机械零件的装配和建筑物，因为用这些图可以测量出距离和角度。但是由于每一种投影仅描绘出物体的一个面，很难从中推导出被投影物体的三维性质，即使对于同一物体的几个投影同时研究也是如此。

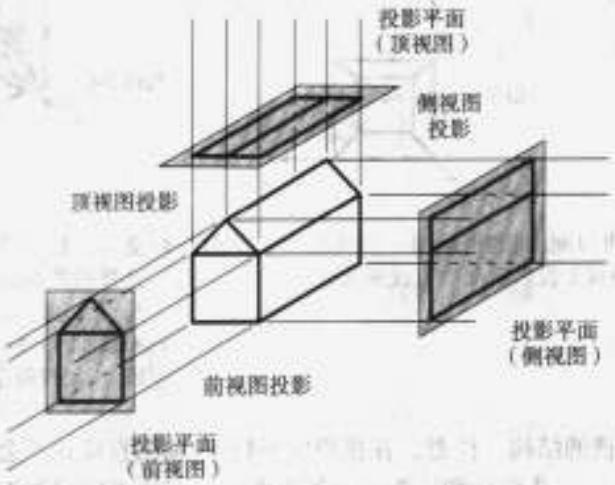


图6-6 三个正投影的结构

轴测正投影（axonometric orthographic projection）使用的投影平面一般不垂直于某一根主轴，所以可以同时显示一个物体的几个面。在这一方面，就像透视投影一样，但是不同的是它的透视缩小系数是一致的而与到投影中心的距离无关。线的平行性保持不变，但是角度不是这样的，同时沿每根主轴，其距离是可度量的（一般具有不同的缩放因子）。

等轴测投影（isometric projection）是最常用的轴测投影。它的投影平面的法线（因此也是投影方向）与每根主轴的夹角相等。如果投影平面的法向量是(d_x, d_y, d_z)，那么我们要求$|d_x| = |d_y| = |d_z|$或者$\pm d_x = \pm d_y = \pm d_z$。这样就有8个方向满足这样的条件（在每一个卦限中有一个方向）。图6-7显示一个沿方向(1, -1, -1)的等轴测投影结构。

等轴测投影具有有用的性质，即沿三个主轴方向具有相等的透视缩小系数，允许沿着三个轴向的量度具有相同的比例（因此才有等轴测的名字：iso代表相等，metric代表度量）。另外，主轴的投影之间具有相同的角度，如图6-8所示。

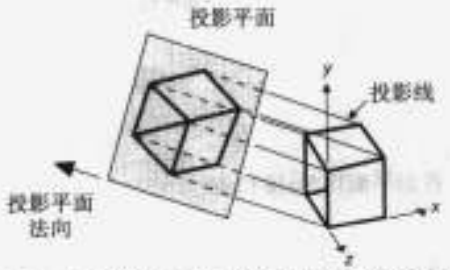


图6-7 一个单位立方体的等轴测投影的结构。（摘自[CARL78]，Association for Computing Machinery，Inc.，授权使用。）

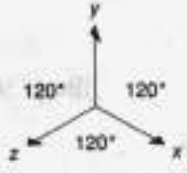


图6-8 三个轴的单位向量的等轴测投影，具有投影方向(1, 1, 1)

232 | 234

斜投影，第二类平行投影，与正投影的不同之处在于投影平面的法向和投影方向不同。斜投影将轴测投影和前、顶、侧正投影的性质结合起来：投影平面垂直于一个主轴，于是平行于投影

平面的物体表面的投影就可以进行角度和距离的测量。物体的另外一些表面也投影，允许沿着主轴测量其距离，但不是角度。斜投影在本书中被广泛（虽然不惟一）使用，这是由于它们本身的属性而且它们易于绘制。图6-9显示斜投影的结构。注意，投影平面法向和投影方向不一致。

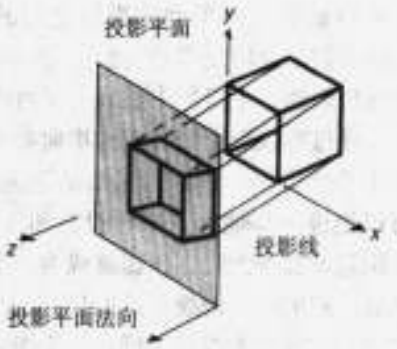


图6-9 斜投影的结构。（摘自[CARL78]，Association for Computing Machinery，Inc.：授权使用。）

两类频繁使用的斜投影是斜等测（cavalier）投影和斜二测（cabinet）投影。对于斜等测投影，其投影方向和投影平面成45° 角，结果是与投影平面垂直的一条直线的投影与该直线本身的长度相等，即不存在透视缩小。图6-10显示一个单位正方体在(x, y)平面上的几种斜等测投影，那些倾斜线是与(x,y)平面相垂直的立方体边的投影，它们与水平轴的夹角为α，一般为30°或45°。

斜二测投影（如图6-11所示）投影方向和投影平面的夹角为arctan(2) = 63.4°，所以与投影平面垂直的那些直线的投影是实际长度的1/2。斜二测投影比斜等测投影更为真实一些，这是因为透视缩小系数为1/2，与我们的其他视觉经验一致。

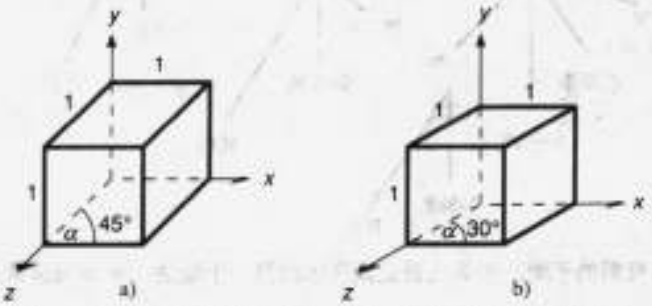


图6-10 单位立方体在$z = 0$平面上的斜等测投影。所有的边投影在单位长度上。在a) 中，投影方向是（$\sqrt{2}/2, \sqrt{2}/2, -1$）；在b) 中，投影方向是（$\sqrt{3}/2, 1/2, -1$）

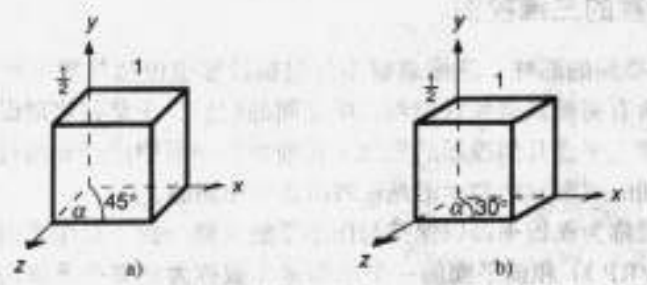


图6-11 单位立方体在$z = 0$平面上的斜二测投影。平行于x和y轴的边投影在单位长度上。在a)中，投影方向是（$\sqrt{2}/4, \sqrt{2}/4, -1$）；在b)中，投影方向是（$\sqrt{3}/4, 1/4, -1$）

图6-12有助于解释这两种投影的投影线与投影平面的角度。(x, y)面是投影平面，点P'是$(0, 0, 1)$在投影平面上的投影。角度α和长度l与图6-10和图6-11中的相同，我们可以通过改变投影方向来控制它们（l是z轴单位向量投影到(x, y)面的长度；α是投影与x轴的夹角）。设定投影方向为$(dx, dy, -1)$，我们可以从图6-12得到$dx = l\cos\alpha, dy = l\sin\alpha$。假设给定$l$和$\alpha$，则投影方向是$(l\cos\alpha, l\sin\alpha, -1)$。

图6-13显示不同类型的投影之间的逻辑关系。连接它们的共同线索是它们包含一个投影平面和透视投影的一个投影中心或者平行投影的投影方向。通过将投影中心看成由指向某个参考点的投影中心的方向与到该参考点的距离所定义，我们可以更进一步地统一平行投影和透视投影。当这个距离增加到无限，投影就成为一个平行投影。因此，我们可以说统一这些投影的共同线索是它们本身包含一个投影平面，一个指向投影中心的方向和一个到投影中心的距离。

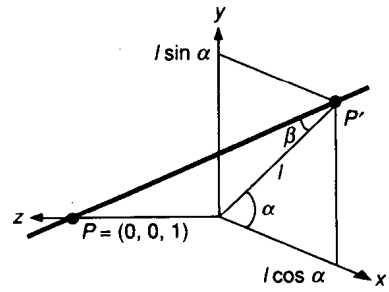


图6-12 $P=(0,0,1)$到$P'=(l\cos\alpha, l\sin\beta, 0)$的斜平行投影，投影方向是$P'-P=(l\cos\alpha, l\sin\beta, -1)$

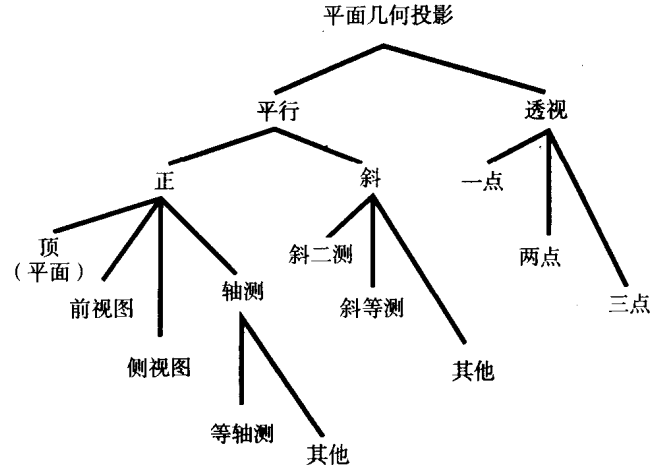


图6-13 平面几何投影的子类。平面视图是顶视图的另一个说法。前和侧经常是不带术语视图被使用

在下一节中，我们考虑怎样混合这些不同类型的投影到三维观察过程。

6.2 指定一个任意的三维视图

如同在图6-1中提到的那样，三维观察不仅包括投影也包括裁剪三维空间的视见体。投影和视见体一起提供所有需要裁剪和投影到二维空间的信息。于是到物理设备坐标的二维变换是直接的。现在我们建立平面几何投影的概念（在前面的一节中已经介绍过）以表明怎样确定一个视见体。这里提到的观察方法和术语是在PHIGS中用到的。

投影平面（以后称为*视图平面*以保持与图形学的文献一致）是用平面上的一个点（该点被称为*视图参考点*（VRP））和该平面的一个法向量（被称为*视图平面法向量*（VPN）[⊖]）定义的。相对于被投影的物体，视图平面可以放在任何位置：可以穿过该物体，也可以放在物体的前面或后面。

235 ~ 237

给定视图平面，我们需要一个在视图平面上的窗口。窗口的作用与二维窗口的作用类似：它的内容被映射到视口，而投影到窗口外的视图平面上三维空间的任何部分是不显示的。我们可以看到窗口在定义视见体的时候也起到重要的作用。

⊖ PHIGS中有另外一个变量，即视图平面距离（VPD）：视图平面可以距离观察点VRP一个距离。VPD在VPN的方向是正的，见习题6.22。

为了在视图平面上定义一个窗口，我们需要一些方法来指定沿两个垂直坐标轴的最小和最大窗口坐标。这些轴是三维观察参考坐标（viewing-reference coordinate,VRC）系的一部分。VRC的原点是VRP。VRC的一个轴是VPN，该轴被称作n轴。VRC的第二个轴从视图上方向量（view up vector, VUP）得到，该向量确定了视图平面上v轴的方向。v轴被定义使得VUP沿着和VPN平行的方向在视图平面上的投影与v轴重合。u轴的方向定义使u,v和n构成一个右手坐标系，如图6-14所示。VRP和两个方向向量VPN和VUP定义于右手世界坐标系中（某些图形程序包将y轴作为VUP，但这样限制太大，如果VPN平行于y轴，将无法定义。在此情况下VUP会是无定义的）。



图6-14 视图平面由VPN和VRP定义；v轴由沿着VPN到视图平面的VUP的投影定义。u轴与VPN和v组成右手观察参考坐标系

定义了VRC系之后，就可以定义窗口的最小和最大的u值和v值，如图6-15 所示。该图说明窗口不需要对称于视图参考点，并且清楚地显示了窗口的中心CW。

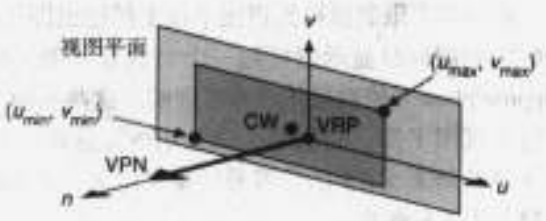


图6-15 观察参考坐标系统（VRC）是一个由u、v和n轴组成的右手系统。n轴总是VPN。CW是窗口中心

投影中心和投影方向（DOP）由投影参考点（projection reference point, PRP）和一个投影类型指针定义。如果投影类型是透视投影，则PRP是投影中心。如果投影类型是平行投影，则DOP从PRP指向CW。CW一般不是VRP,甚至不需要在窗口边界以内。

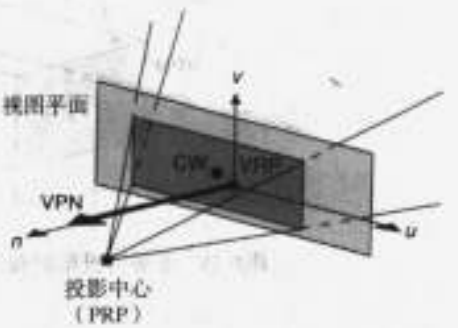


图6-16 透视投影的半无限四棱锥视见体。CW是窗口中心

PRP在VRC系统中定义，而不是在世界坐标系中。因此，PRP 相对于VRP的位置并不随着VUP或VRP的移动而改变。这样的好处是程序员可以指定要求的投影方向，例如，使用斜等测投影，然后改变VPN和VUP(因此改变了VRC),而不必要重新计算需要保持预期投影的PRP。另一方面，移动PRP来获得物体的不同视图会更困难一些。

视见体为世界空间中将被裁剪出来并投影到视图平面的那一部分定出边界。对透视投影，视见体是一个半无限的四棱锥形，其顶点是在PRP，边则穿过窗口的拐角。图6-16显示一个透视投影的视见体。投影中心后面的位置不包含在视见体中，因此不会被投影。当然，在现实中，我们的眼睛看到一个不规则形状的圆锥形的视见体。但是，一个四棱锥形的视见体是数学上更易处理的，且与矩形视口的概念相一致。

对于平行投影，视见体是一个无限长的平行管道，它的各边都平行于投影方向，该方向是从PRP到窗口中心的方向。图6-17和图6-18显示平行投影视见体和它们与视图平面、窗口和PRP的关系。对于正平行投影，但是不对于斜平行投影，视见体的侧面垂直于视图平面。

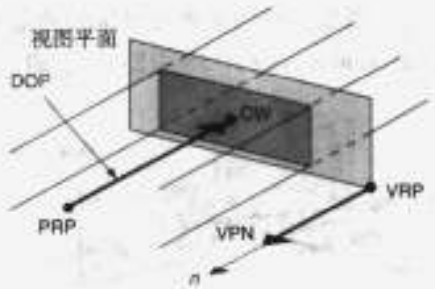


图6-17 平行正投影的无限平行管道视见体。VPN和投影的方向（DOP）是平行的。DOP是从PRP到CW的向量，且平行于VPN

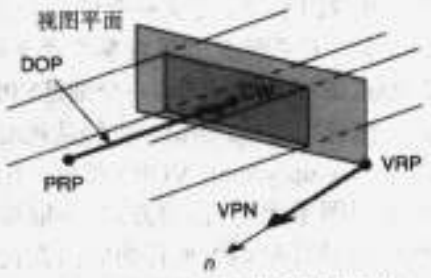


图6-18 斜正投影的无限平行管道视见体。投影方向（DOP）不与VPN平行

有时为了限制投影到视图平面上的输出图元的数量，我们可能需要视见体是有限的。图6-19、图6-20和图6-21显示怎样用一个*前裁剪平面*（front clipping plane）和一个*后裁剪平面*（back clipping plane）使视见体成为有限的。这些平面（有时称为*前截面*（hither）和*后截面*（yon））均平行于视图平面；它们的法线是VPN。这些平面由相对于视图参考点和沿着VPN的称为*前距离*（*F*）或*后距离*（*B*）的带符号的量来定义，正向取在VPN方向。对于正的视见体，前距离必须在代数上大于后距离。

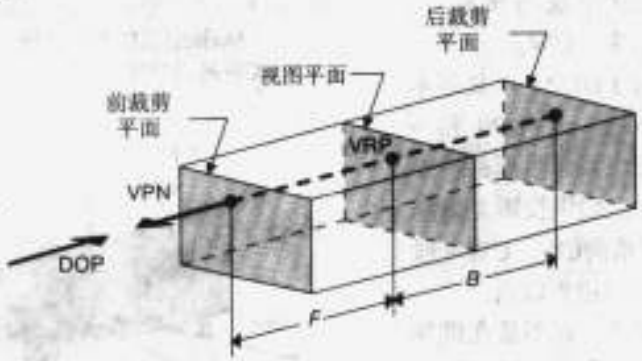


图6-19 正平行投影的被截断的视见体。DOP是投影方向

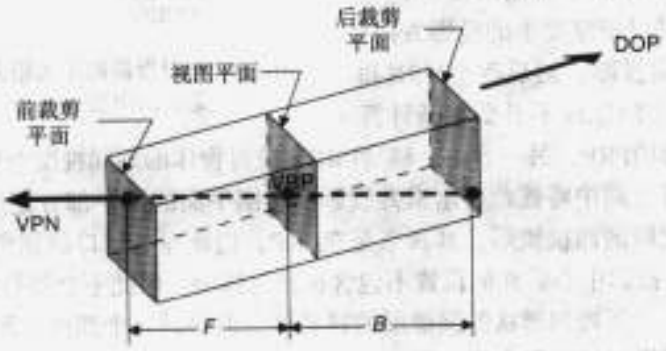


图6-20 显示VPN与投影方向（DOP）相斜的斜平行投影的被截断的视见体；VPN也是前裁剪平面和后裁剪平面的法线

以这种方式限制视见体是有用的，可以消除无关物体同时允许用户关注于空间的一个特别的部分。前后距离的动态修改可以给观察者在物体的不同部分之间的空间关系上一种好的感觉（当这些部分在视野中出现和从视野中消失时，参见第14章）。对于透视投影，则有另外一个动机。恰好处在

投影中心距离上的一个物体在投影平面上成为不可区别形状的斑点。在将这样的物体显示到绘图仪上时，笔可能穿破纸；在向量显示器上，CRT荧光物质可能被电子束烧坏；在胶片记录器上，光线的高度集中导致出现一个模糊的白色区域。同时，非常靠近投影中心的物体可像许多不连接的采集棍一样扩展过窗口，没有可辨别的结构。适当地指定视见体可以消除这些问题。

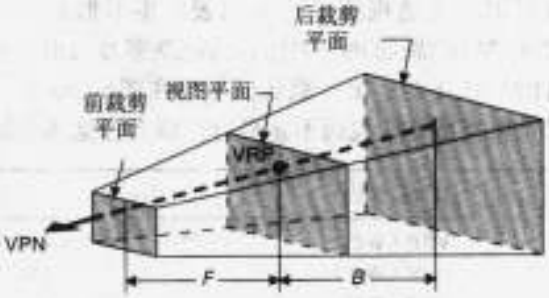


图6-21 透视投影的截断的视见体

视见体的内容怎样映射到显示表面上呢？首先，考虑在规格化投影坐标（normalized projection coordinate）的三个维度都从0到1扩展的单位立方体。视见体被转换为NPC的矩形实体，该实体的定义是沿着x轴从x_{min}到x_{max}，沿着y轴从y_{min}到y_{max}，沿着z轴从z_{min}到z_{max}。前裁剪平面成为z_{max}面，后裁剪平面成为z_{min}面。同样地，视见体的u_{min}侧成为x_{min}面，且u_{max}侧成为x_{max}面。最后，视见体的v_{min}侧成为y_{min}面，且v_{max}侧成为y_{max}面。NPC的矩形实体部分（称为三维视口）在单位立方体内。

238 ↓ 241

接着该单位立方体的$z=1$面被映射到可被输出到显示器上的最大的正方形上。为了创建三维视口的内容的线框显示（它是视见体的内容），每一个输出图元的z分量被简单地抛弃，同时输出图元被显示。我们在第15章将会看到隐藏面的消除使用z分量来确定哪一个输出图元最接近于观察者并因此是可见的。

PHIGS使用两个4×4的矩阵，它们是视图方向矩阵和视图映射矩阵，来表示观察定义的完整的集合。VRP、VPN和VUP被混合来形成视图方向矩阵，该矩阵将在空间坐标系中表示的位置转换为在VRC中表示的位置。这个变换使u、v和n轴分别对应x、y和z轴。

由PRP，u_{min}，u_{max}，v_{min}，v_{max}，F和B定义的视见体定义与由x_{min}，x_{max}，y_{min}，y_{max}，z_{min}，z_{max}定义的三维视口定义结合形成视图映射矩阵，该矩阵将VRC中的点转换为规格化投影坐标的点。形成视图方向矩阵和视图映射矩阵的子程序调用在7.3.4节讨论。

在下一节，我们将看到怎样使用本节介绍的概念获得不同视图。6.4节将介绍平面几何投影的基础数学，而在6.5节将介绍为所有观察操作所需要的数学和算法。

6.3 三维观察的例子

本节我们考虑怎样应用前面介绍的基本观察概念来产生各种投影，如图6-22和图6-23显示的那些投影。因为这些图中显示的房子在本节将被通篇使用，记住它的维度和位置（见图6-24）将是很有帮助的。对于每一个讨论的视图，我们给出一个显示VRP、VPN、VUP、PRP、窗口

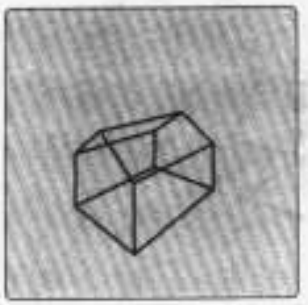


图6-22 房子的两点透视投影

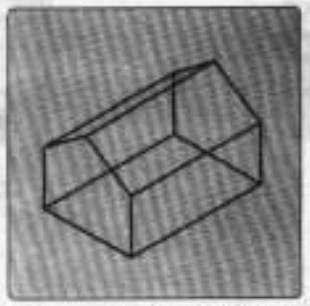


图6-23 房子的等轴测投影

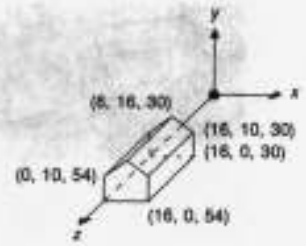


图6-24 房子在z轴上从30延伸到54，在x轴上从0延伸到16，在y轴上从0延伸到16

和投影类型（透视或平行）的表。本节假定NPC中的单位立方体为三维视口默认值。符号(WC)或(VRC)添加到表中作为给定观察参数的坐标系的记号。为PHIGS使用的默认观察定义的表格的形式在此表出。默认值示于在图6-25a中。与这些默认值对应的视见体示于在图6-25b。如果投影类型是透视而不是平行，那么视见体是如图6-25c显示的四棱锥。

观察参数	值	注 释
VRP（WC）	(0, 0, 0)	原点
VPN（WC）	(0, 0, 1)	z轴
VUP（WC）	(0, 1, 0)	y轴
PRP（VRC）	(0.5,0.5,1.0)	
窗口（VRC）	(0,1,0,1)	
投影类型	平行	

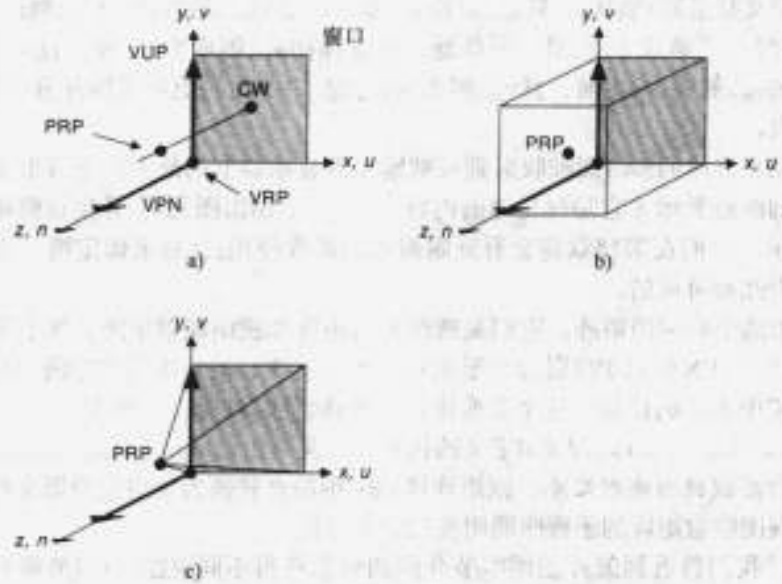


图6-25 a) 默认的观察定义：VRP在原点，VUP是y轴，VPN是z轴。这使得u, v, n的VRC系统与x, y, z的世界坐标系统一致。窗口沿着u和v从0延伸到1，且PRP在(0.5, 0.5, 1.0)；b) 默认的平行投影视见体；c) 默认投影是透视投影的视见体

鼓励想了解所有这些参数怎样相关的读者以辅助标尺的形式构建一所房子、世界坐标系统和VRC系统，如图6-26所示。想法是将VRC系统定位到世界坐标系中，其方式与视图举例中所



图6-26 用于理解三维观察的模型。a) 房子和世界坐标系，b) 房子和VRC系统

用的方式一样，同时假想从房子的线段顶点发出的投影线与视图平面相交。经验表明，这是一种很好的理解（和讲授）三维观察概念的方法。 242 - 244

6.3.1 透视投影

为了得到如图6-27所示的房子（这个图和所有类似的图都用第7章讨论的SPHIGS程序绘制）前部的一点透视图，我们放置投影的中心（这可看成观察者的位置）到$x=8$、$y=6$和$z=84$。x值被选择为房子的水平中心，y的值大约和一个站在（x,z）平面上的观察者的眼睛高度相等；z值任意。在这种情况下，将z值从房子的前面移动了30个单位（$z=54$的平面）。窗口被设置得相当大，以保证房子适合在视见体内。所有其他的观察参数有其默认值，所有观察参数的整个集合如下：

VRP（WC）	(0, 0, 0)
VPN（WC）	(0, 0, 1)
VUP（WC）	(0, 1, 0)
PRP（VRC）	(8, 6, 84)
窗口（VRC）	(-50, 50, -50, 50)
投影类型	透视

尽管图6-27的图像实际是房子的透视投影，但是它非常小且不在视图平面的中间。我们倾向于房子的一个更居中的投影，同时使它的投影几乎覆盖整个视图平面，如图6-28所示。如果视图平面和房子的前平面重合，我们可以更轻松地得到这个效果。现在，由于房子的前平面在x和y两个方向上都从0延伸到16，如果窗口在x和y方向都从-1延伸到17，就可以得到合理的结果。

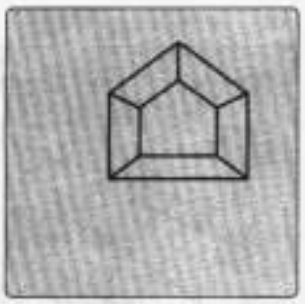


图6-27 房子的一点透视投影

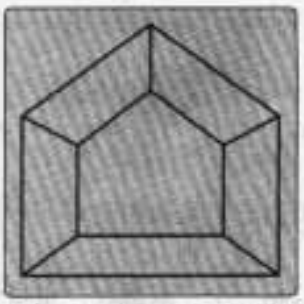


图6-28 房子的中心透视投影

通过在$z=54$平面上放置VRP，我们放置视图平面到房子的前平面上：(0, 0, 54)，即房子的前左下角即可。为使投影中心与图6-27一样，PRP（在VRC系统中）需要处在(8, 6, 30)上。图6-29显示VRC、VRP和PRP的新的安排，它对应于下面的观察参数的集合： 245

VRP（WC）	(0, 0, 54)
VPN（WC）	(0, 0, 1)
VUP（WC）	(0, 1, 0)
PRP（VRC）	(8, 6, 30)
窗口（VRC）	(-1, 17, -1, 17)
投影类型	透视

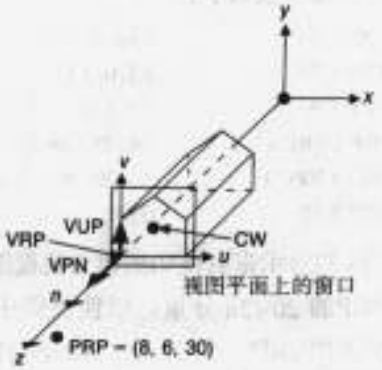


图6-29 图6-28的观察位置

可以用很多其他的方法获得同样的结果。例如，VRP在(8, 6, 54)，如图6-30所示，PRP给定的投影中心成为(0, 0, 30)。窗口必须也被改变，因为它的

定义是基于VRC系统的，其原点为VRP。合适的窗口的u从－9延伸到9，v从－7延伸到11。对于房子，这是与前面例子使用相同的窗口，但是这里它定义于一个不同的VRC系统。因为向上的观察方向是y轴，所以u轴和x轴是平行的，就好像v和y轴是平行的一样。概括来讲，下面的观察参数，如图6-30所示，也产生图6-28：

VRP（WC）	（8，6，54）
VPN（WC）	（0，0，1）
VUP（WC）	（0，1，0）
PRP（VRC）	（0，0，30）
窗口（VRC）	（－9，9，－7，11）
投影类型	透视

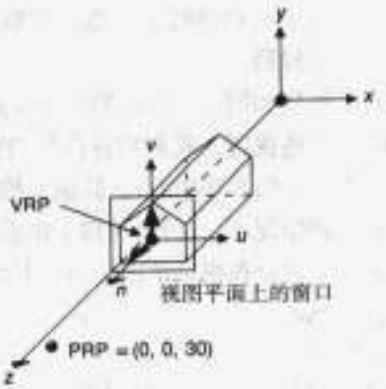


图6-30 图6-28的另一个观察位置

下面，让我们尝试获得如图6-22所示的两点透视投影。投影中心与取世界坐标物体快照的照相机的位置相似。考虑到这个相似性，图6-22的投影的中心好像位于房子的右上部，就好像从正z轴看一样。确切的投影中心是(36，25，74)。现在，如果房子在(16，0，54)的拐角选为VRP，则该投影中心是在相对于该角的(20，25，20)。由于视图平面与房子的前平面（$z=54$平面）重合，一个u从－20到20和v从－5到35的窗口就肯定足够大到包含这个投影。因此，我们可以用如下的观察参数定义图6-31的视图：

VRP（WC）	（16，0，54）
VPN（WC）	（0，0，1）
VUP（WC）	（0，1，0）
PRP（VRC）	（20，25，20）
窗口（VRC）	（－20，20，－5，35）
投影类型	透视

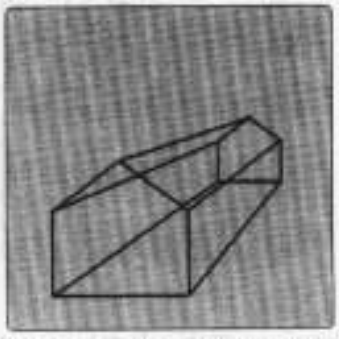


图6-31 从(36，25，74)看到的房子的透视投影，VPN平行于z轴

该视图类似于但是很明显不同于图6-22所示的视图。一方面，图6-22是一个两点透视投影，而图6-31是一个一点透视。很显然地，简单地移动投影中心不能产生图6-22。实际上，我们需要对视图平面重新定向，使得通过设置VPN为(1，0，1)，它可以交于x和z轴。因此，图6-22的观察参数如下：

VRP（WC）	（16，0，54）
VPN（WC）	（1，0，1）
VUP（WC）	（0，1，0）
PRP（VRC）	（0，25，$20\sqrt{2}$）
窗口（VRC）	（－20，20，－5，35）
投影类型	透视

图6-32显示带有该VPN建立的视图平面。使用了PRP的$20\sqrt{2}n$分量，以使投影中心距离在(x,y)平面的VRP的$20\sqrt{2}$处，如图6-33所示。

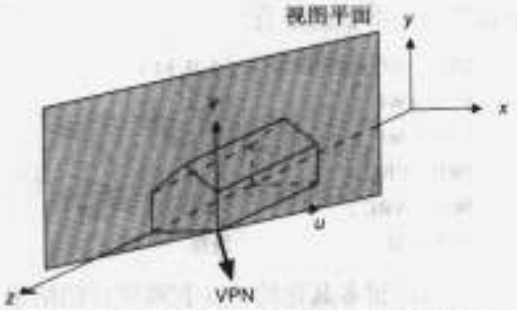


图6-32 对应于图6-22的视图平面和VRC系统

有两种方式选择完全包围投影的窗口，像

图6-22的窗口一样。可以用草图估计房子投影到视图平面的尺寸，如图6-33所示，来计算投影线与视图平面的交集。但是一个更好的方法是允许窗口绑定为程序中的变量，该程序被通过一个定值设备或定位设备交互确定。

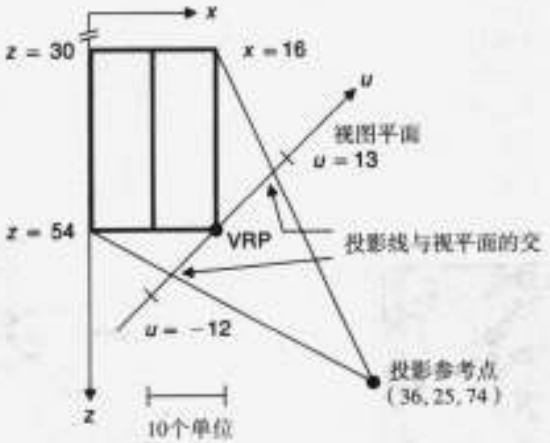


图6-33 用于确定一个合适的窗口大小的房子的俯视（平面）图

图6-34从与图6-22相同的投影获得，但是窗口有一个不同的方向。在前面的所有例子中，VRC系统的v轴平行于世界坐标系中的y轴；因此窗口（其两侧面均平行于v轴）与房子的垂直面很好地一致。图6-34有与图6-22精确一致的观察参数，除了VUP从y轴旋转偏离了10°。

另一个为透视投影定义观察参数的方法在图6-35中给出。该图按照摄影师可能考虑的放置照相机位置的方法构建。需要六个参数：投影中心，它是照相机的位置；注视中心，它是一个照相机瞄准的点（VPN是从注视中心到投影中心的向量）；VUP，向上向量；D，从投影中心到投影平面的距离；W，投影平面上窗口的宽度；H，投影平面上窗口的高度。注视中心不需要在视图平面上。在这个模型中，VPN总是直接指向投影中心，视见体对中心线对称。上述位置都是在世界坐标中被给定的——没有观察坐标的概念。习题6.24要求你将这六个观察参数转换为这里给定的观察模型。

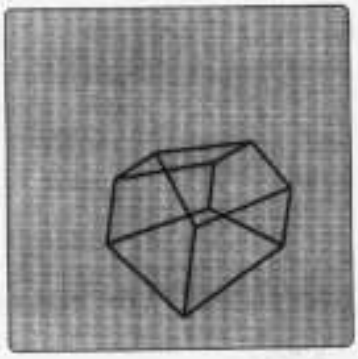


图6-34 由旋转VUP产生的房子的投影

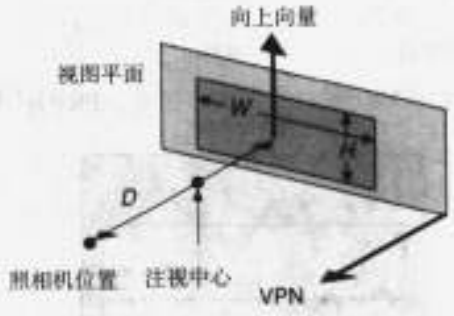


图6-35 定义视图的另一个方法，用照相机位置（投影中心）、注视中心、向上向量、从投影中心到投影平面的距离和投影平面上窗口的高度和宽度。VPN平行于从注视中心到照相机位置的方向

6.3.2 平行投影

我们通过使投影的方向平行于z轴创建一个房子的前平行投影（图6-36）。回忆一下投影的

方向是由PRP和窗口中心确定。采用默认的VRC系统和(-1, 17, -1, 17)的窗口，窗口中心是(8, 8, 0)。在(8, 8, 100)的PRP提供平行于z轴的投影方向。图6-37显示产生图6-36的观察位置。观察参数如下：

VRP (WC)	(0, 0, 0)
VPN (WC)	(0, 0, 1)
VUP (WC)	(0, 1, 0)
PRP (VRC)	(8, 8, 100)
窗口 (VRC)	(-1, 17, -1, 17)
投影类型	平行

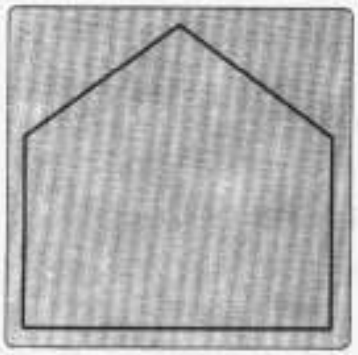


图6-36 房子的前平行投影

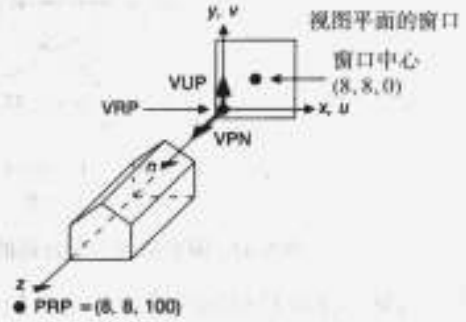


图6-37 产生图6-36所示房子的一个前视图的观察位置。PRP可以是$x = 8$和$y = 8$的任意点

为了创建侧视图（图6-38），我们需要图6-39的观察位置，以(y, z)平面（或任何平行于它的平面）作为视图平面。这种情况与下面的观察参数对应：

VRP (WC)	(0, 0, 54)	
VPN (WC)	(1, 0, 0)	x轴
VUP (WC)	(0, 1, 0)	y轴
PRP (VRC)	(12, 8, 16)	
窗口 (VRC)	(-1, 25, -5, 21)	
投影类型	平行	

窗口中心在VRC的(12, 8, 0)；因此，PRP有同样的u和v坐标。

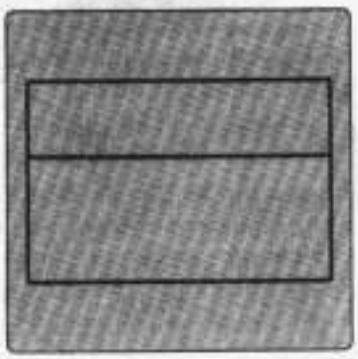


图6-38 从房子侧面的平行投影

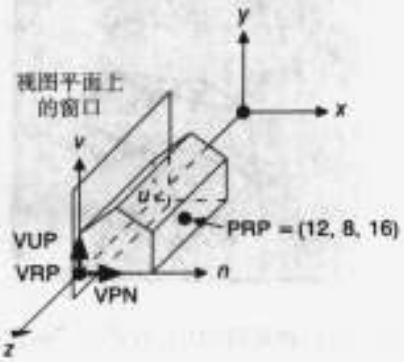


图6-39 图6-38的观察位置

我们通过使用(x, z)平面作为视图平面并且VPN作为y轴来创建一个房子的顶视图。+y的默

认的视图上方方向必须被改变；我们使用负x轴。由于VRP再次定义为房子的一个角，我们获得图6-40所示的观察位置，它由下面的观察参数定义：

VRP（WC）	(16, 0, 54)	
VPN（WC）	(0, 1, 0)	y轴
VUP（WC）	(−1, 0, 0)	负x轴
PRP（VRC）	(12, 8, 30)	
窗口（VRC）	(−1, 25, −5, 21)	
投影类型	平行	

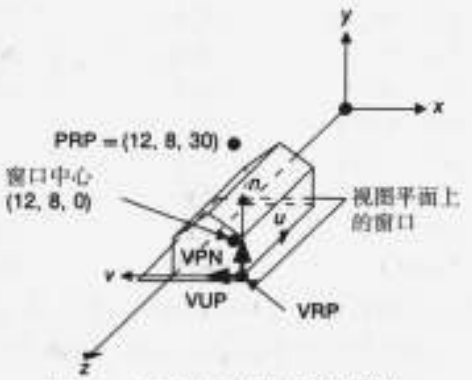


图6-40 房子的俯视图的观察位置。

图6-23是一个在方向(−1, −1, −1)上的等轴测（平行正）投影，是等轴测投影的8个可能的方向之一（见6.1.2节）。下面的观察参数创建这样的一个等轴测投影：

VRP（WC）	(8, 8, 42)
VPN（WC）	(1, 1, 1)
VUP（WC）	(0, 1, 0)
PRP（VRC）	(0, 0, 10)
窗口（VRC）	(−20, 20, −20, 20)
投影类型	平行

具有角度α（见图6-10）的斜等测投影由下列参数定义：

VRP（WC）	(8, 8, 54)	房子的正中
VPN（WC）	(0, 0, 1)	z轴
VUP（WC）	(0, 1, 0)	y轴
PRP（VRC）	(cosα, sinα, 1)	
窗口（VRC）	(−15, 15, −15, 15)	
投影类型	平行	

窗口是关于VRP对称的，这意味着VRP(VRC系统的原点)是窗口的中心。像定义中那样放置PRP意味着投影方向（即从PRP指向窗口中心的向量）是(0, 0, 0) − PRP = −PRP，即它的负PRP方向。从6.1.2节起，我们知道斜等测投影的投影方向在前面的表中给定的：−PRP = −(cosα, sinα, 1)。现在注意如果希望斜等测投影到一个常数x的平面而不是常数z的平面将发生什么：仅VRP、VPN和VUP需要改变来建立一个新的VRC系统。PRP、窗口和投影类型可保持不变。

246
t
252

6.3.3 有限的视见体

至此，在所有的例子中，视见体都被假设为无限的。6.2节描述的前和后裁剪平面可帮助确定一个有限的视见体。这些平面都是平行于视图平面的，且分别在距离视图参考点F和B的地方（沿着VPN从VRP开始度量）。为了避免负视见体，我们必须保证F在代数上大于B。

一个后墙裁剪掉的房子的前透视图（图6-41）是由下述观察定义（其中已经添加了F和B）导致的结果。如果给定距离，就假定是对相应平面做裁剪；否则，就不做。上述观察定义如下：

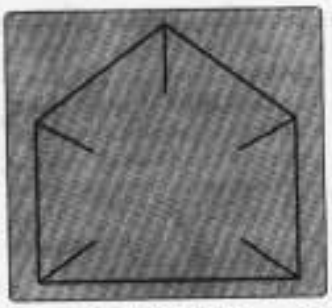


图6-41 后裁剪平面在$z = 31$处的房子的透视投影

VRP（WC）	(0,0,54)	房子的前左下角
VPN（WC）	(0,0,1)	z轴
VUP（WC）	(0,1,0)	y轴
PRP（VRC）	(8,6,30)	
窗口（VRC）	(-1,17,-1,17)	
投影类型	透视	
F(VRC)	+1	在房子前面一个单位，$z = 54 + 1 = 55$
B(VRC)	-23	在房子后面向前一个单位，$z = 54 - 23 = 31$

这种情况的观察位置与图6-29的一样，除了添加了裁剪平面。

如果前后裁剪平面动态移动，则被观察物体的三维结构可较静态观察时更容易识别。

6.4 平面几何投影的数学

在本节中，我们将介绍平面几何投影的基础数学。为了简化起见，我们假设用于透视投影的投影平面与z轴垂直，且位于$z = d$处，而对于平行投影，投影平面是$z = 0$面。每一个投影可
253 被4×4矩阵定义。这样很方便，因为投影矩阵可由变换矩阵组成，允许两个操作（变换，然后投影）被表示为一个矩阵。下一节中我们将讨论任意投影平面。

在本节中，我们从投影平面（该投影平面在距离原点d处，有一个点P被投影到投影平面上）开始推导出几种投影的4×4矩阵。为了计算$P_p = (x_p, y_p, z_p)$，(x, y, z)在$z = d$处的投影平面上的透视投影，我们使用图6-42中的相似三角形来写比率

$$\frac{x_p}{d} = \frac{x}{z}; \qquad \frac{y_p}{d} = \frac{y}{z} \tag{6-1}$$

用d乘以每一侧得到

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d}, \qquad y_p = \frac{d \cdot y}{z} = \frac{y}{z/d} \tag{6-2}$$

距离d正好是应用于x_p和y_p的比例因子。被z除产生透视投影，其中远距离物体的投影比近处的物体的投影更小。除了0以外允许所有z值。投影点可在负z轴上的投影中心后或在投影中心和投影平面之间。

公式(6-2)的变换可表示为一个4×4的矩阵：

$$M_{\text{per}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{6-3}$$

点$P = [x\ y\ z\ 1]^{\mathrm{T}}$与矩阵$M_{\text{per}}$相乘产生一般齐次坐标的点$[X\ Y\ Z\ W]^{\mathrm{T}}$

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M_{\text{per}} \cdot P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6-4}$$

或

$$[X \quad Y \quad Z \quad W]^{\mathrm{T}} = \left[x \quad y \quad z \quad \frac{z}{d}\right]^{\mathrm{T}} \tag{6-5}$$

现在，被W（等于z/d）除，且去掉第四个坐标回到三维，我们有

$$\left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}\right) = (x_p, y_p, z_p) = \left(\frac{x}{z/d}, \frac{y}{z/d}, d\right) \tag{6-6}$$

这些公式是公式(6-1)的正确的结果，加上变换后的d的z坐标，d是沿着z轴的投影平面的位置。

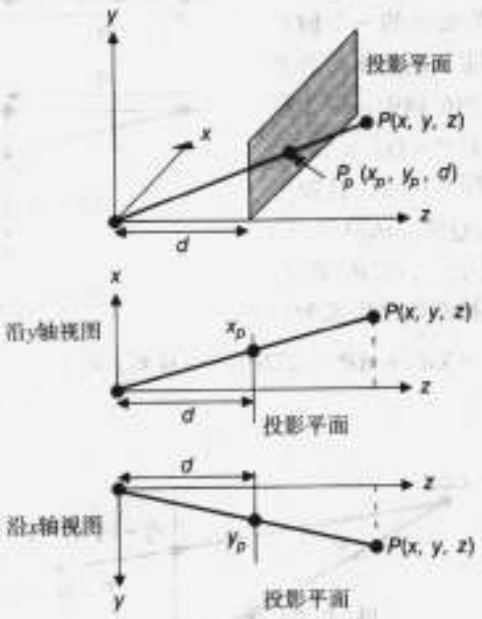


图6-42 透视投影

另一个透视投影的公式将投影平面置于$z=0$的位置，透视投影中心在$z=-d$，如图6-43所示。三角形的相似性现在给出

$$\frac{x_p}{d} = \frac{x}{z+d}, \qquad \frac{y_p}{d} = \frac{y}{z+d} \tag{6-7}$$

与d相乘，我们得到

$$x_p = \frac{d \cdot x}{z+d} = \frac{x}{(z/d)+1}, \qquad y_p = \frac{d \cdot y}{z+d} = \frac{y}{(z/d)+1} \tag{6-8}$$

投影矩阵是

$$M'_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \tag{6-9}$$

该公式允许d（到投影中心的距离）趋于无限。

在位于$z=0$的投影平面上的正投影是直接的。这种情况下，投影的方向与投影平面的法线（z轴）一致。因此，点P投影为

$$x_p = x, \quad y_p = y, \quad z_p = 0 \tag{6-10}$$

该投影用下面的矩阵表示：

$$M_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-11}$$

注意当公式(6-9)中的d趋于无限时，公式(6-9)成为公式(6-11)。

M_{per}仅应用在特殊的场合，即其投影中心在原点的情况。M_{or}仅应用在投影方向平行于z轴的时候。基于[WEIN87]中提出的概念的一个健壮性更强的公式，不仅删除这些限制还将平行和透视投影统一到一个公式中。在图6-44中，一个点$P=(x, y, z)$到投影平面的投影是$P_p=(x_p, y_p, z_p)$。投影平面垂直于z轴，在距离原点z_p处，且投影中心（COP）是距离点$(0, 0, z_p)Q$处。从$(0, 0, Z_p)$到COP的方向被规格化方向向量(d_x, d_y, d_z)给定。P_p在COP和P之间的线上，可被参数化定义为：

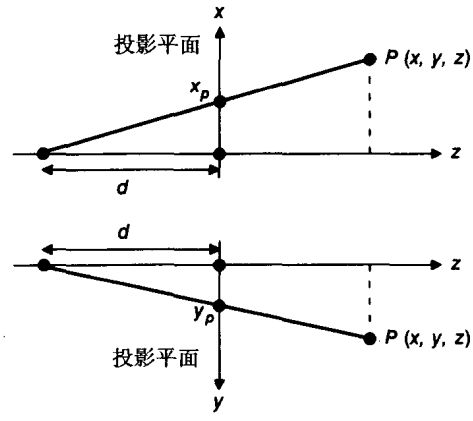


图6-43 另一个透视投影

$$\text{COP} + t(P - \text{COP}), \quad 0 \leqslant t \leqslant 1 \tag{6-12}$$

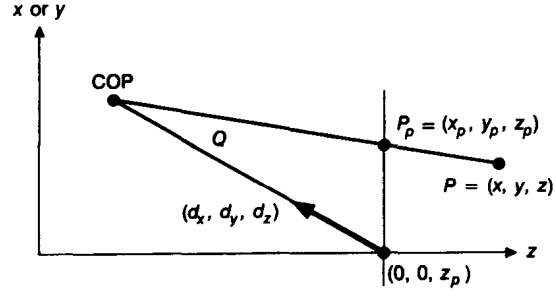


图6-44 从COP到$P=(x, y, z)$的直线与在$z=z_p$处的投影平面的交点是点P的投影。COP是从点$(0, 0, z_p)$开始的距离Q，方向为(d_x, d_y, d_z)

重写公式(6-12)作为在直线上的任意点$P'=(x', y', z')$的单独公式，$\text{COP}=(0, 0, z_p)+Q(d_x, d_y, d_z)$，产生

$$x' = Q\,d_x + (x - Q\,d_x)t \tag{6-13}$$

$$y' = Q\,d_y + (y - Q\,d_y)t \tag{6-14}$$

$$z' = (z_p + Q\,d_z) + (z - (z_p + Q\,d_z))t \tag{6-15}$$

我们求出点P的投影P_p，即在COP和P之间的直线与投影平面的交点，通过代入$z'=z_p$到公式(6-15)，先求出参数t：

$$t = \frac{z_p - (z_p + Q\,d_z)}{z - (z_p + Q\,d_z)} \tag{6-16}$$

将t的值代入公式(6-13)和公式(6-14)，求出$x'=x_p$和$y'=y_p$，这产生

$$x_p = \frac{x - z\dfrac{d_x}{d_z} + z_p\dfrac{d_x}{d_z}}{\dfrac{z_p - z}{Q\,d_z} + 1} \tag{6-17}$$

$$y_p = \frac{y - z\dfrac{d_y}{d_z} + z_p\dfrac{d_y}{d_z}}{\dfrac{z_p - z}{Q\,d_z} + 1} \tag{6-18}$$

将恒等式$z_p = z_p$的右边乘以一个分子和分母都是公式(6-19)的分数：

$$\frac{z_p - z}{Q\,d_z} + 1 \tag{6-19}$$

保持恒等式不变且给予z_p与x_p和y_p同样的分母：

$$z_p = z_p \frac{\frac{z_p - z}{Q\,d_z} + 1}{\frac{z_p - z}{Q\,d_z} + 1} = \frac{-z\frac{z_p}{Q\,d_z} + \frac{z_p^2 + z_p Q\,d_z}{Q\,d_z}}{\frac{z_p - z}{Q\,d_z} + 1} \tag{6-20}$$

现在公式(6-17)、公式(6-18) 和公式(6-20)可被重写为4 × 4的矩阵M_{general}使得M_{general}的最后一行被$[x\ y\ z\ 1]^T$乘产生它们的公分母，它是齐次坐标W且因此是X和Y、Z的约数： 254 ~ 257

$$M_{\text{general}} = \begin{bmatrix} 1 & 0 & -\frac{d_x}{d_z} & z_p\frac{d_x}{d_z} \\ 0 & 1 & -\frac{d_y}{d_z} & z_p\frac{d_y}{d_z} \\ 0 & 0 & -\frac{z_p}{Q\,d_z} & \frac{z_p^2}{Q\,d_z} + z_p \\ 0 & 0 & -\frac{1}{Q\,d_z} & \frac{z_p}{Q\,d_z} + 1 \end{bmatrix} \tag{6-21}$$

当给出下面的值，M_{general}就特殊化为三个前面推导出来的矩阵M_{per}, M'_{per}和M_{ort}：

	z_p	Q	$[d_x$	d_y	$d_z]$
M_{ort}	0	∞	[0	0	−1]
M_{per}	d	d	[0	0	−1]
M'_{per}	0	d	[0	0	−1]

当$Q \neq \infty$时，M_{general}定义一个一点透视投影。透视投影的灭点通过乘以z轴上无限远点计算得出，z轴上的无穷远点由M_{general}用$[0\ 0\ 1\ 0]^T$的齐次坐标给出。计算这个乘积，且被W相除，得出

$$x = Q\,d_x, \quad y = Q\,d_y, \quad z = z_p$$

给定一个预期的灭点 (x, y)和一个已知的到投影中心的距离Q，这些等式惟一定义$[d_x\ d_y\ d_z]$，这是因为$\sqrt{d_x^2 + d_y^2 + d_z^2} = 1$。

类似地，易于表明到 (x, y)面的斜等测投影和斜二测投影，具有图6-10和图6-11 所示的角度α时：

	z_p	Q	$[d_x$	d_y	$d_z]$
斜等测	0	∞	[cosα	sinα	−1]
斜二测	0	∞	$\left[\frac{\cos\alpha}{2}\right.$	$\frac{\sin\alpha}{2}$	$\left.-1\right]$

本节中，我们已经看到怎样表示M_{per}, M'_{per}和 M_{ort}，所有这些是更一般的M_{general}的特殊情况。但在所有这些情况中，投影平面都垂直于z轴。在下一节中，我们去除这个限制且考虑有限视见体隐含的裁剪。

6.5 实现平面几何投影

给定一个视见体和一个投影，让我们考虑裁剪和投影的观察操作怎样被实际应用。正如观察过程的概念模型（图6-1）所提示的那样,我们可以通过计算定义视见体的六个面的每一个与 258

直线的交点来裁剪直线。在裁剪后仍存在的线将被投影到视图平面上，这一过程要同时求解投影线与视图平面的交点方程。然后交点坐标从三维世界坐标转化到二维设备坐标。但是，该过程需要的大量计算过程（很多线需要重复）包括相当大的计算量。令人高兴的是，有一个更有效的过程，基于分治策略将一个困难的问题分解为一系列简单的问题。

某些视见体较一般的空间更容易被裁剪（裁剪算法在6.5.3节被讨论）。例如，计算直线和被六个面

$$x = -1,\quad x = 1,\quad y = -1,\quad y = 1,\quad z = 0,\quad z = -1 \tag{6-22}$$

定义的平行投影视见体的平面的交点是很简单的。这对于下述平面定义的透视投影视见体也成立：

$$x = z,\quad x = -z,\quad y = z,\quad y = -z,\quad z = -z_{\min},\quad z = -1 \tag{6-23}$$

这些规范视见体示于图6-45。

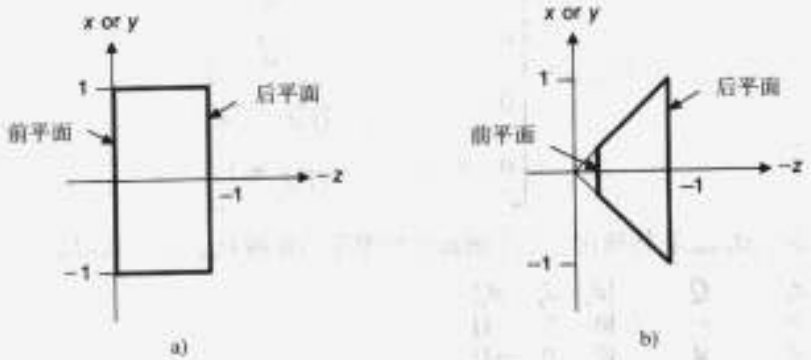


图6-45 两个规范视见体。a) 为平行投影，b) 为透视投影。注意，$-z$方向指向右侧

我们的策略是找到规格化变换N_{par}和N_{per}，它们将任意的平行投影或透视投影视见体分别转换为平行和透视规范视见体。然后执行裁剪，接着通过6.4节的矩阵投影到二维。该策略的风险是需要浪费很多随后要被裁剪操作抛弃的点也要做变换，但是至少裁剪变得容易做了。

图6-46显示这里包括的一系列过程。我们可以通过组合步骤3和4为一个变换矩阵将它简化为变换–裁剪–变换序列。对透视投影，还需要一个除法来从齐次坐标映射回三维坐标。该除法在
259 组合序列的第二个变换后完成。另一个可选择的策略，在齐次坐标中裁剪，将在6.5.4节中讨论。

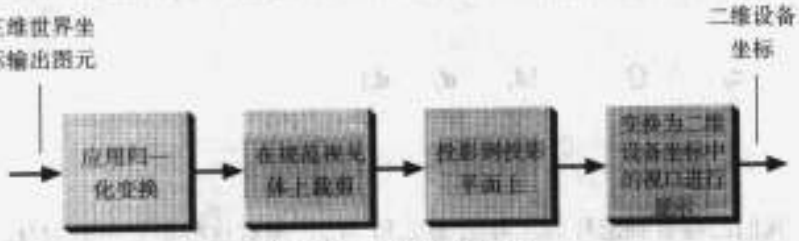


图6-46 三维观察的实现

熟悉PHIGS的读者会注意到公式(6-22)和公式(6-23)的规范视见体与PHIGS的默认视见体不同：对平行投影，是x, y, z从0到1的单位立方体，而对于透视投影，是顶点在(0.5, 0.5, 1.0)且侧面穿过在$z = 0$的平面上x和y从0到1的单位正方形的四棱锥。这里的规范视见体的定义是为简化裁剪公式且提供6.5.4节讨论的平行投影和透视投影之间的一致性。另一方面，PHIGS默认视见体的定义使二维观察成为三维观察的一种特殊情况。习题6.26涉及PHIGS的各种默认定义。

在下面的两节中，我们为透视投影和平行投影推导规格化变换，它用做变换-裁剪-变换序列的第一步。

6.5.1 平行投影

本节中，我们为平行投影推导规格化变换N_{par}来变换世界坐标位置，使得视见体变换为由公式(6-22)定义的规范视见体。变换后的坐标按照该规范视见体裁剪，其结果投影到$z=0$平面，然后变换到视口来显示。

变换N_{par}按最一般的情况（即斜（而不是正）平行投影）推导。因此N_{par}包括一个导致观察坐标的投影方向平行于z的错切转换，即使在（u, v, n）坐标系它也并不平行于VPN。通过包括该错切，我们可以仅通过设置$z=0$来实现在$z=0$面上的投影。如果平行投影是正平行投影，则该规格化变换的错切分量是恒等的。

组成N_{par}的一系列变换如下：

1) 平移VRP 到原点。

2) 旋转VRC使n轴（VPN）成为z轴，u轴成为x轴，v轴成为y轴。

3) 错切变换使投影方向平行于z轴。 260

4) 平移和缩放变换到由公式(6-22)给出的平行投影规范视见体。

在PHIGS中，步骤1和2定义视图方向矩阵（view-orientation matrix），同时步骤3和4定义视图映射矩阵（view-mapping matrix）。

图6-47显示该变换序列应用到平行投影视见体和一个房子的轮廓的情形；图6-48 显示上述变换序列产生的平行投影。

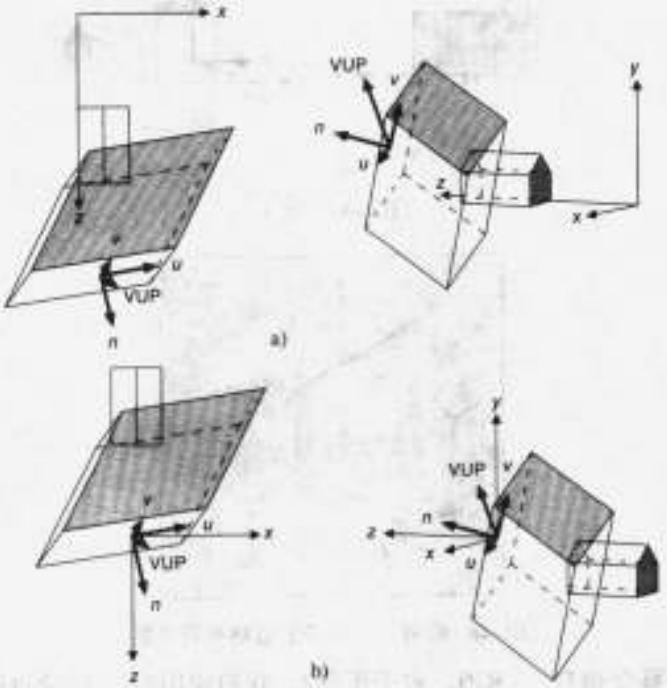


图6-47 在平行投影观察流水线的各个阶段的结果。在每种情况都显示一个俯视的偏轴平行投影。a) 最初的观察位置，b) VRP平移到原点，c) (u, v, n) 坐标系旋转到与（x, y, z）系统一致，d) 视见体错切使得投影方向(DOP)与z轴平行，e) 视见体平移和缩放变换到规范平行投影视见体。观察参数是VRP = (0.325, 0.8, 4.15)，VPN = (0.227, 0.267, 1.0)，VUP = (0.293, 1.0, 0.227)，PRP = (0.6, 0.0, -1.0)，窗口 = (-1.425, 1.0, -1.0, 1.0)，$F=0.0$，$B=-1.75$。（图由乔治·华盛顿大学的L.Lu编程绘出。）

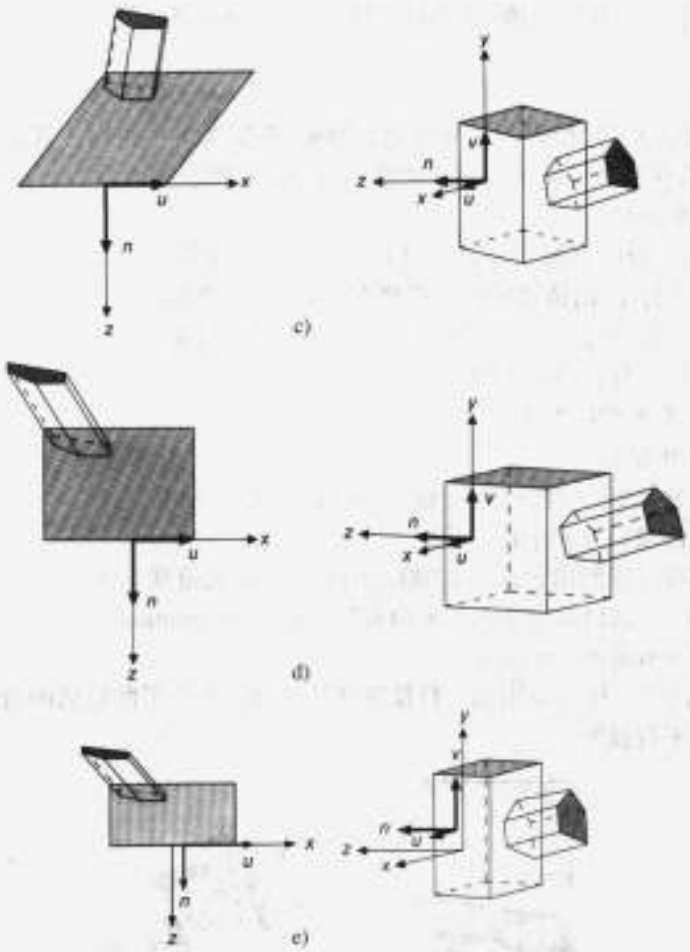


图6-47（续）

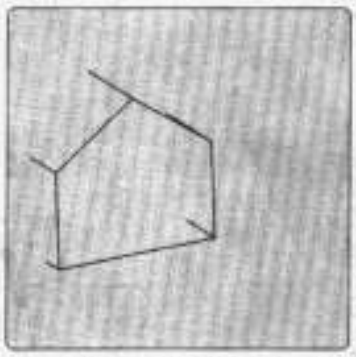


图6-48 裁剪后的房子的最终平行投影

步骤1正好是平移变换$T(-\mathrm{VRP})$。对于步骤2，我们使用5.5节讨论过的并在公式(5-66)和(5-67)推导中表明的正交矩阵的性质。执行步骤2的旋转矩阵的行向量是单位向量，它们被R旋转到x，y和z轴。VPN旋转到z轴，于是

$$R_z = \frac{\mathrm{VPN}}{\|\mathrm{VPN}\|} \tag{6-24}$$

u轴（垂直于VUP和VPN，因此是单位向量沿着VUP 和R_z（与VPN方向相同）的叉积。）被旋转到x轴，于是

$$R_x = \frac{\text{VUP} \times R_z}{\|\text{VUP} \times R_z\|} \tag{6-25}$$

类似地，v轴（垂直于R_z和R_x）,旋转到y轴，于是

$$R_y = R_z \times R_x \tag{6-26}$$

因此，步骤2的旋转由下述矩阵给出：

$$R = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-27}$$

其中r_{1x}是R_x的第一个元素，依此类推。

第三步是沿着z轴错切视见体，使得它的所有平面垂直于坐标轴之一。我们通过确定错切应用到投影方向（DOP）使其与z轴重合来实现该步骤。回忆一下DOP是从PRP 到窗口中心（CW）的向量，并且PRP定义于VRC系统。最前面的两个变换步骤使VRC与世界坐标系一致，所以PRP自身现在是在世界坐标中。因此，DOP是CW－PRP。给定

$$\text{DOP} = \begin{bmatrix} dop_x \\ dop_y \\ dop_z \\ 0 \end{bmatrix}, \quad \text{CW} = \begin{bmatrix} \frac{u_{\max} + u_{\min}}{2} \\ \frac{v_{\max} + v_{\min}}{2} \\ 0 \\ 1 \end{bmatrix}, \quad \text{PRP} = \begin{bmatrix} prp_u \\ prp_v \\ prp_n \\ 1 \end{bmatrix} \tag{6-28}$$

261 ~ 263

于是

$$\begin{aligned} \text{DOP} &= \text{CW} - \text{PRP} \\ &= \begin{bmatrix} \frac{u_{\max} + u_{\min}}{2} & \frac{v_{\max} + v_{\min}}{2} & 0 & 1 \end{bmatrix}^{\text{T}} - [prp_u \quad prp_v \quad prp_n \quad 1]^{\text{T}} \end{aligned} \tag{6-29}$$

图6-49显示这样定义的DOP和预期的DOP'。

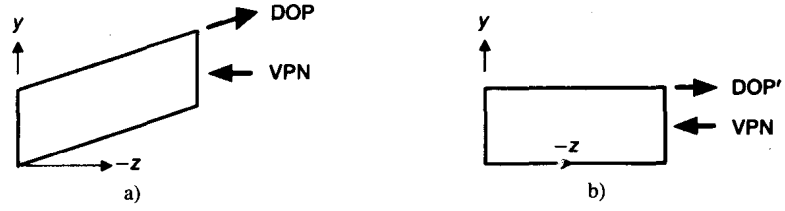


图6-49 使用视见体的侧视图作为错切的例子。a) 中的平行四边形错切到 b) 中的矩形；由于VPN平行于z轴，所以不变

错切可用5.6节的公式(5-47)的(x, y)错切矩阵完成。使用系数shx_{par}和shy_{par}，矩阵是

$$SH_{\text{par}} = SH_{xy}(shx_{\text{par}}, shy_{\text{par}}) = \begin{bmatrix} 1 & 0 & shx_{\text{par}} & 0 \\ 0 & 1 & shy_{\text{par}} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6-30}$$

如5.6节描述的那样，当将$z \cdot shx_{\text{par}}$和$z \cdot shy_{\text{par}}$添加到x和y时，SH_{xy}使z不受影响。我们要求出

shx_{par}和shy_{par}使得

264

$$\mathrm{DOP}' = [0 \quad 0 \quad dop_z \quad 0]^{\mathrm{T}} = SH_{par} \cdot \mathrm{DOP} \tag{6-31}$$

执行公式(6-31)的乘法，随后进行代数处理，表明等式发生在

$$shx_{par} = -\frac{dop_x}{dop_z}, \quad shy_{par} = -\frac{dop_y}{dop_z} \tag{6-32}$$

注意到，对于正投影，$dop_x = dop_y = 0$,于是$shx_{par} = shy_{par} = 0$,且错切矩阵简化为恒等矩阵。

图6-50显示这三个变换步骤应用后的视见体。视见体的范围是

$$u_{min} \leqslant x \leqslant u_{max}, \qquad v_{min} \leqslant y \leqslant v_{max}, \qquad B \leqslant z \leqslant F \tag{6-33}$$

这里F和B分别是从VRP沿着VPN到前裁剪平面和后裁剪平面的距离。

在过程中的第四步即最后一步是把错切变换后的视见体变换到规范视见体。我们通过平移公式(6-33)给出的视见体的前中心到原点，然后缩放变换到公式(6-22)给出的$2 \times 2 \times 1$的最终规范视见体来实现该步骤。这两个变换是

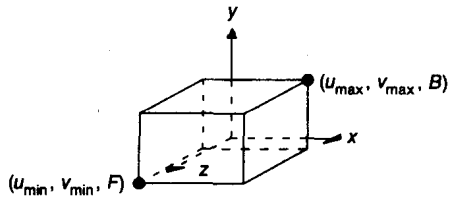


图6-50 经过变换步骤1到3后的视见体

$$T_{par} = T\left(-\frac{u_{max} + u_{min}}{2}, -\frac{v_{max} + v_{min}}{2}, -F\right) \tag{6-34}$$

$$S_{par} = S\left(\frac{2}{u_{max} - u_{min}}, \frac{2}{v_{max} - v_{min}}, \frac{1}{F - B}\right) \tag{6-35}$$

如果F和B没有定义（因为前平面裁剪和后平面裁剪不起作用），那么任意满足$B \leqslant F$的值可被使用。值0和1是满足的。

总起来说，我们有

$$N_{par} = S_{par} \cdot T_{par} \cdot SH_{par} \cdot R \cdot T(-\mathrm{VRP}) \tag{6-36}$$

N_{par}把任意的平行投影视见体变换到平行投影规范视见体，因此允许输出图元在平行投影规范

265 视见体上裁剪。

6.5.2 透视投影

我们现在为透视投影推导规格化变换N_{per}。N_{per}变换世界坐标位置，使得视见体成为透视规范视见体，即截断的四棱锥，其顶点是由公式(6-23)定义的原点。在应用N_{per}后，裁剪在规范视见体进行，裁剪结果用（6.4节推导的）M_{per}投影到视图平面。

组成N_{per}的变换序列如下：

1) 平移VRP到原点。

2) 旋转VRC 使得n轴（VPN）成为z轴，u轴成为x轴，v轴成为y轴。

3) 平移使得PRP给定的投影中心（COP）在原点。

4) 错切使得视见体的中心线成为z轴。

5) 缩放变换使得视见体成为规范透视视见体，即由公式(6-23)的六个平面定义的截断四棱锥。

图6-51显示这个变换序列应用于透视投影视见体和一个房子。图6-52显示结果透视投影。

步骤1和2 是与平行投影一致的：$R \cdot T(-\mathrm{VRP})$。步骤3是投影中心(COP)到原点的平移，为规范视见体所要求。COP由$\mathrm{PRP} = (prp_u, prp_v, prp_n)$在VRC中相对于VRP定义。观察参考坐标由步骤1和2变换到世界坐标，于是在VRC中COP的定义现在也在世界坐标中。因此，步骤3的平移正好是$T(-\mathrm{PRP})$。

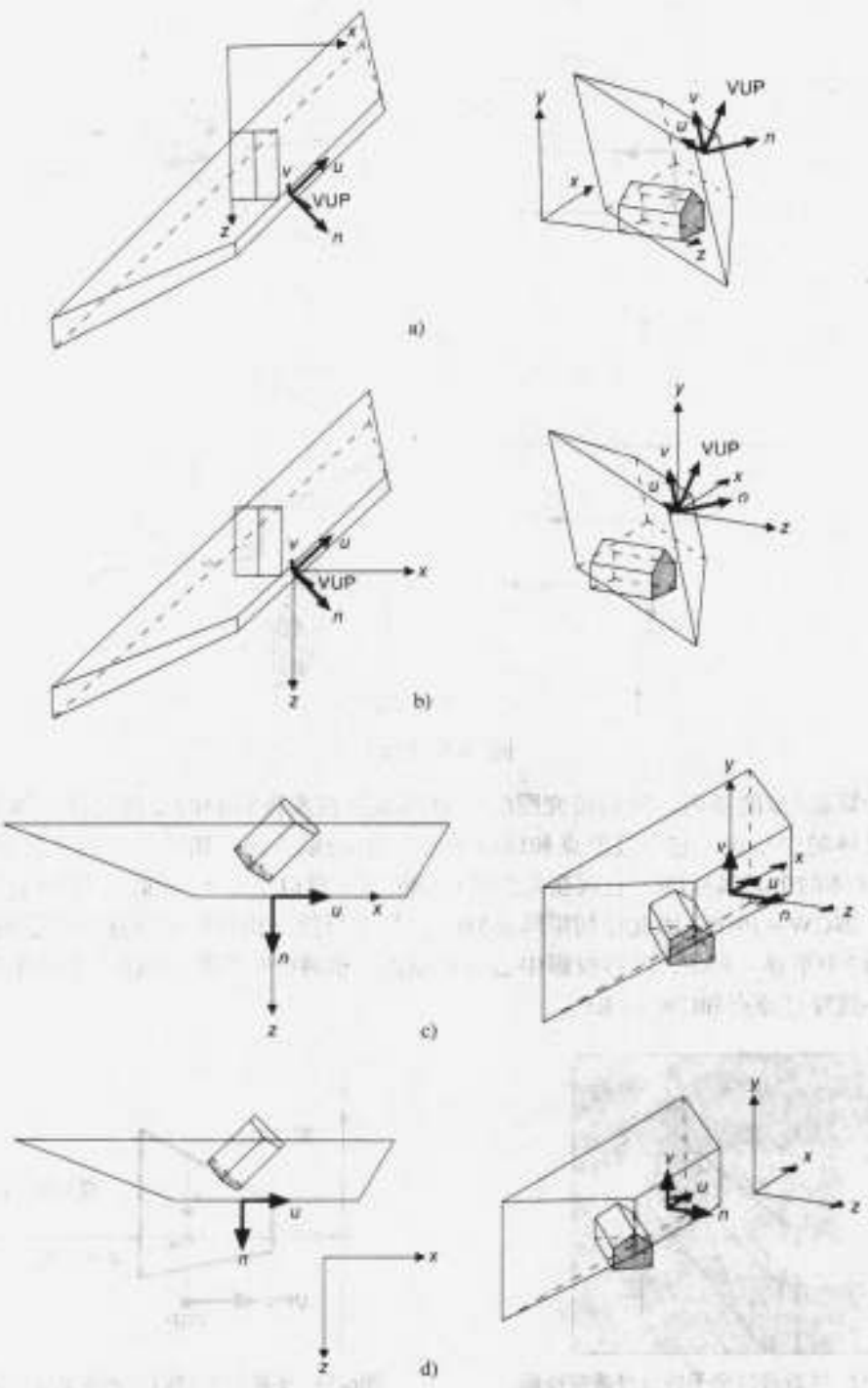


图6-51 透视投影观察流水线的各个阶段的结果。在每种情况下都显示一个俯视偏轴平行投影。a)最初的观察位置，b)VRP平移到原点，c)(u, v, n) 坐标系统旋转到与(x, y, z)系统一致。d)投影中心（COP）平移到原点，e)视见体错切使得投影方向(DOP)与z轴平行，f)缩放变换视见体到规范透视投影视见体。观察参数是VRP = (1.0, 1.275, 2.6)，VPN = (1.0, 0.253, 1.0)，VUP = (0.414, 1.0, 0.253)，PRP = (1.6, 0.0, 1.075)，窗口 = (− 1.325, 2.25, − 0.575, 0.575)，F = 0，B = − 1.2。（此图由乔治·华盛顿大学的L.Lu编程绘出。）

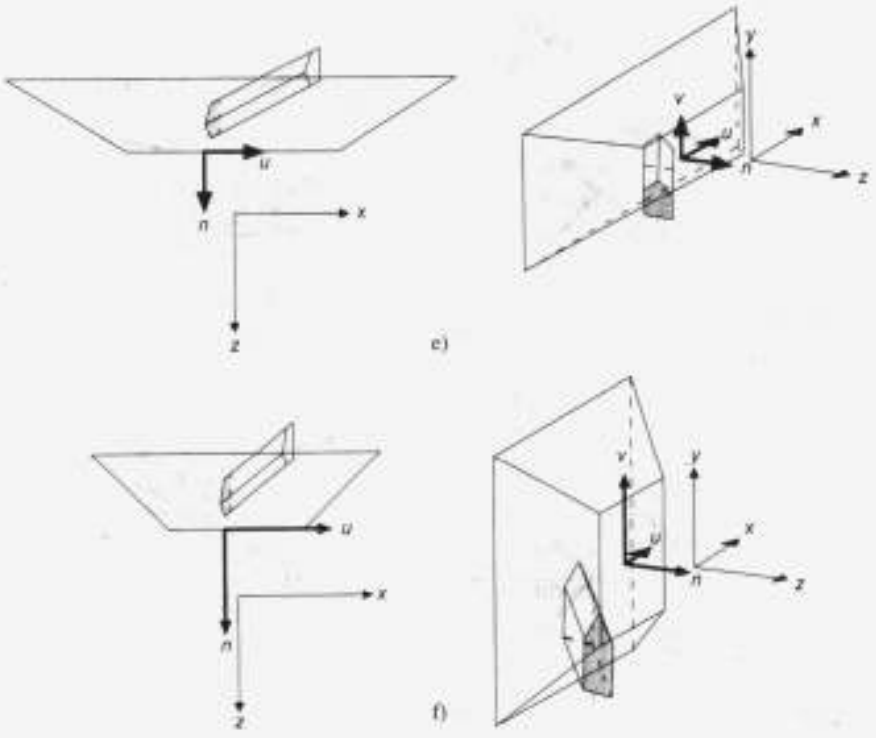


图 6-51（续）

为了计算第4步的错切，我们研究图6-53,该图显示在步骤1到3的变换后视见体的侧视图。注意，视见体的中心线（它穿过原点和窗口中心）与 $-z$ 轴不同。错切的目的是变换中心线为 $-z$ 轴。视见体的中心线从PRP(它现在是在原点)到CW（窗口中心）。因此它与平行投影的投影方向相同，即CW－PRP。因此错切矩阵是 SH_{par}，与平行投影相同！考虑这个问题的另一种方式是在步骤3中平移－PRP，即将投影中心放在原点，也将CW平移－PRP，于是步骤3后，视见体的中心线穿过原点和CW－PRP。

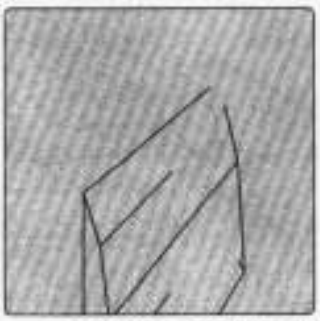


图6-52 被裁剪的房子的最终透视投影

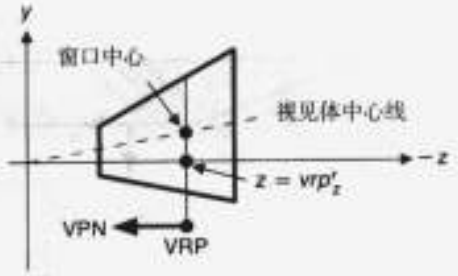


图6-53 变换步骤1到3后的视见体的横截面

在应用错切后，窗口（因此是视见体）集中在 z 轴上。在投影平面上窗口的范围是

$$-\frac{u_{max}-u_{min}}{2} \leqslant x \leqslant \frac{u_{max}-u_{min}}{2} \tag{6-37}$$

$$-\frac{v_{max}-v_{min}}{2} \leqslant y \leqslant \frac{v_{max}-v_{min}}{2}$$

VRP在步骤3前是在原点，现在由步骤3平移且由步骤4错切。定义VRP'作为在步骤3和4的变换后的VRP，

$$\text{VRP}' = SH_{\text{par}} \cdot T(-\text{PRP}) \cdot [0\ 0\ 0\ 1]^{\text{T}} \tag{6-38}$$

VRP'的z分量（表示成vrp_z'）与$-prp_n$相等，这是因为(x, y)平面的错切SH_{par}并不影响z坐标。

最后一个步骤是沿着三个轴缩放变换以产生由公式(6-23)定义并示于图6-54的规范视见体。因此，缩放变换最好想像成是在两个子步骤里实现。在第一个子步骤里，我们按不同比例变换x和y，以给出界定视见体单位斜度边界的倾斜面。我们通过缩放窗口使得它的半高度和半宽度都是$-vrp_z'$来完成该子步骤。合适的x和y缩放因子分别是$-2 \cdot vrp_z'/(u_{\max} - u_{\min})$和$-2 \cdot vrp_z'/(v_{\max} - v_{\min})$。在第二个子步骤里，我们沿着三个轴（以保持单位斜率）均匀缩放使得在$z = vrp_z' + B$的后裁剪平面成为$z = -1$平面。该子步骤的缩放因子是$-1/(vrp_z' + B)$。该缩放因子具有一个负号，因为$vrp_z' + B$本身为负，使缩放因子成为正的。

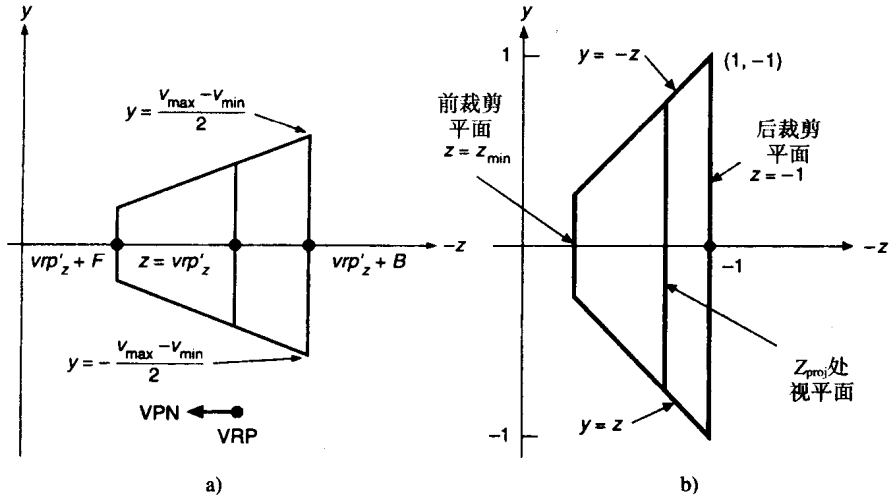


图6-54 在最后的缩放步骤前后的视见体的横截面。在这个例子中，F和B的符号相反，所以前裁剪平面和后裁剪平面在VRP的相反侧面。a) 缩放前，b) 缩放后

将这两个子步骤合在一起，我们得到透视缩放变换：

$$S_{\text{per}} = S\left(\frac{2\ vrp_z'}{(u_{\max} - u_{\min})(vrp_z' + B)}, \frac{2\ vrp_z'}{(v_{\max} - v_{\min})(vrp_z' + B)}, \frac{-1}{vrp_z' + B}\right) \tag{6-39}$$

应用该缩放到z改变投影平面和裁剪平面的位置到新的位置：

$$z_{\text{proj}} = -\frac{vrp_z'}{vrp_z' + B}, \quad z_{\min} = -\frac{vrp_z' + F}{vrp_z' + B}, \quad z_{\max} = -\frac{vrp_z' + B}{vrp_z' + B} = -1 \tag{6-40}$$

总起来说，将透视投影视见体变换为透视投影规范视见体的规格化观察变换是：

$$N_{\text{per}} = S_{\text{per}} \cdot SH_{\text{par}} \cdot T(-\text{PRP}) \cdot R \cdot T(-\text{VRP}) \tag{6-41}$$

类似地，回忆将平行投影视见体变换为平行投影规范视见体的规格化观察变换是：

$$N_{\text{par}} = S_{\text{par}} \cdot T_{\text{par}} \cdot SH_{\text{par}} \cdot R \cdot T(-\text{VRP})$$

这些变换发生在齐次空间中。在什么条件下我们现在可以回到三维进行裁剪呢？只要我们知道$W>0$。该条件很容易理解。负W意味着：当我们用W除的时候，Z的符号和z将相反。具有负Z的点将有正z值，并且可能显示时看上去它们已经裁剪掉了。

什么时候可以肯定我们将具有$W>0$呢？被应用到点、线和面的旋转、平移、缩放和错切（如第5章定义的那样）将保持$W>0$；实际上，它们将保持$W=1$。因此，N_{per}或者N_{par}都不会影响变换点的齐次坐标，所以被W除正常情况将不是映射回三维所必需，同时在适当的规范视见体上的裁剪可以进行。在根据透视投影规范视见体进行裁剪后，透视投影矩阵M_{per}（包含除法）必须应用。

266 ~ 270

如果输出图元包括表示成齐次坐标函数的曲线和曲面，并显示为连接起来的直线段，则可能得到$W<0$。例如，如果X的符号不改变，W的函数的符号从曲线的一个点到下一个点发生变化，那么X/W将在曲线两个点上有不同的符号。在第11章讨论的有理B样条是其中一例。$-W$也产生于使用一些在第5章讨论的变换之外的变换，例如伪影（"fake" shadow）[BLIN88]。

在下一节，将讨论几个三维空间裁剪算法。然后，在6.5.4节我们讨论当我们不能确保$W>0$时怎样裁剪。

6.5.3 用三维规范视见体进行裁剪

对于平行投影来说，规范视见体是单位立方体，对于透视投影来说则是截断的规则四棱锥。在第3章讨论的Cohen-Sutherland和Cyrus-Beck裁剪算法很容易扩展到三维。

对于规范平行视见体的二维Cohen-Sutherland算法的扩展使用一个6位的外码。当满足条件时，某一位为真(1):

第1位—点在视见体的上面——$y>1$

第2位—点在视见体的下面——$y<-1$

第3位—点在视见体的右面——$x>1$

第4位—点在视见体的左面——$x<-1$

第5位—点在视见体的后面——$z<-1$

第6位—点在视见体的前面——$z>0$

与二维的情况类似，如果一条直线的两个端点的编码是全零，则这条直线就被简单地接受；如果两个端点的编码逐位求逻辑"与"后并非全为零，则该直线就可被简单地拒绝。否则，将开始进一步分割处理。可能要进行多达六个交点的计算，每一个对应于视见体的一个侧面。

求交计算使用从$P_0(x_0, y_0, z_0)$到$P_1(x_1, y_1, z_1)$的线段的参数表示：

$$x = x_0 + t(x_1 - x_0) \tag{6-42}$$

$$y = y_0 + t(y_1 - y_0) \tag{6-43}$$

$$z = z_0 + t(z_1 - z_0) \qquad 0 \leqslant t \leqslant 1 \tag{6-44}$$

当t可以从0变到1时，这三个方程给出了直线段上从P_0到P_1所有点的坐标。

为了计算一条直线与视见体的$y=1$平面的交点，我们用常数1替代公式(6-43)中的变量y，从而求出t，$t=(1-y_0)/(y_1-y_0)$。如果t在0到1这一区间的外面，则交点在从点P_0到P_1的直线的无限远处，但是不在P_0和P_1之间的直线段上，因此不是所求的。如果t在[0, 1]区间，则将t代入求x和z，从而得到交点的坐标：

271

$$x = x_0 + \frac{(1 - y_0)(x_1 - x_0)}{y_1 - y_0}, \quad z = z_0 + \frac{(1 - y_0)(z_1 - z_0)}{y_1 - y_0} \tag{6-45}$$

该算法使用外码使得t是否在[0,1]内的检测成为不必要的。

用于规范化透视视见体裁剪的外码位如下：

第1位—点在视见体的上面——$y>-z$

第2位—点在视见体的下面——$y<z$

第3位—点在视见体的右面——$x > -z$

第4位—点在视见体的左面——$x < z$

第5位—点在视见体的后面——$z < -1$

第6位—点在视见体的前面——$z > z_{min}$

计算直线和斜平面的交点是简单的。在$y = z$面上，公式(6-43)必须与公式(6-44)相等，$y_0 + t(y_1 - y_0) = z_0 + t(z_1 - z_0)$。于是

$$t = \frac{z_0 - y_0}{(y_1 - y_0) - (z_1 - z_0)} \tag{6-46}$$

将t代入公式(6-42）和公式(6-43)，则得到

$$x = x_0 + \frac{(x_1 - x_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)}, \quad y = y_0 + \frac{(y_1 - y_0)(z_0 - y_0)}{(y_1 - y_0) - (z_1 - z_0)} \tag{6-47}$$

我们知道$z = y$。选择该规范视见体的原因现在很清楚：平面的单位斜率使得交点的计算比任意斜率平面时的计算简单。

Cyrus-Beck裁剪算法用于在一个一般的凸三维多面体上裁剪一条直线，同时被专门用于三维观察四棱锥[CYRU78]。随后Liang和Barsky独立发展了一种用于竖直的二维和三维裁剪区域[LIAN84]的更有效和更专用的版本。在二维中，至多t的四个值被计算，每个对应于窗口的4条边的每一条；在三维中，至多六个值。按照与二维情况完全相同的准则决定取舍直到精确地保留t的两个值（该值将在[0, 1]区间）。对于既不是0也不是1的t的任意值，式(6-42)、式(6-43)和式(6-44)用来计算x, y和z的值。在平行投影规范视见体中，我们通过用3.12.4节给出的代码来完成到三维的扩展。在透视投影规范视见体中，需要求出6个面的N_i, P_{Ei}, $P_0 - P_{Ei}$的新值和t；它们在表6-1中给出。Liang和Barsky开发但被修改以使其与我们的变量名一致、同时使用外向而不是内向法线实际的代码在图6-55中给出。通过核对可以看出表6-1的最后一列的分数的分子和分母与代码中的术语之间的一致性。$-N_i \cdot D$（它是t的分母）的符号被图6-55中的过程CLIPt用来确定交点是可能进入还是可能离开。因此，在表中所做的代数简化过程中，t的分母的符号总是保持不变的。 272

表6-1 Cyrus-Beck三维规范透视投影视见体裁剪的关键变量和公式

裁剪边	向外法线 N_i	边上的点 P_{Ei}	$P_0 - P_{Ei}$	$t = \frac{N_i \cdot (P_0 - P_{Ei})}{-N_i \cdot D}$		
right: $x = -z$	(1, 0, 1)	$(x, y, -x)$	$(x_0 - x, y_0 - y, z_0 + x)$	$\frac{(x_0 - x) + (z_0 + x)}{-(dx + dz)}$	=	$\frac{x_0 + z_0}{-dx - dz}$
left: $x = z$	(−1, 0, 1)	(x, y, x)	$(x_0 - x, y_0 - y, z_0 - x)$	$\frac{-(x_0 - x) + (z_0 - x)}{dx - dz}$	=	$\frac{-x_0 + z_0}{dx - dz}$
bottom: $y = z$	(0, −1, 1)	(x, y, y)	$(x_0 - x, y_0 - y, z_0 - y)$	$\frac{-(y_0 - y) + (z_0 - y)}{dy - dz}$	=	$\frac{-y_0 + z_0}{dy - dz}$
top: $y = -z$	(0, 1, 1)	$(x, y, -y)$	$(x_0 - x, y_0 - y, z_0 + y)$	$\frac{(y_0 - y) + (z_0 + y)}{-dy - dz}$	=	$\frac{y_0 + z_0}{-dy - dz}$
front: $z = z_{min}$	(0, 0, 1)	(x, y, z_{min})	$(x_0 - x, y_0 - y, z_0 - z_{min})$	$\frac{(z_0 - z_{min})}{-dz}$	=	$\frac{z_0 - z_{min}}{-dz}$
back: $z = -1$	(0, 0, −1)	$(x, y, -1)$	$(x_0 - x, y_0 - y, z_0 + 1)$	$\frac{-(z_0 + 1)}{dz}$	=	$\frac{-z_0 - 1}{dz}$

注：变量D（它是$P_1 - P_0$）被表示为（d_x, d_y, d_z）。每条边上的点P_{Ei}的精确坐标与计算无关，所以它们由变量x, y和z表示。右边（$z = -x$）上的点如第1行第3项所示。

```
void Clip3D (double *x0, double *y0, double *z0, double *x1, double *y1, double *z1,
    double *zmin, boolean *accept)
{
    double tmin = 0.0, tmax = 1.0;
    double dx = *x1 - *x0, dz = *z1 - *z0;
    *accept = FALSE;        /* 初始时假设没有边可见 */
    if (CLIPt (-dx - dz, *x0 + *z0, &tmin, &tmax))                    /* 右侧 */
        if (CLIPt (dx - dz, -*x0 + *z0, &tmin, &tmax)) {              /* 左侧 */
            /* 如果到这儿，直线的部分在 -z≤x≤z中 */
            double dy = *y1 - *y0;
            if (CLIPt (dy - dz, -*y0 + *z0, &tmin, &tmax))            /* 底部 */
                if (CLIPt (-dy - dz, *y0 + *z0, &tmin, &tmax))        /* 顶部 */
                    /* 如果到这儿，直线的部分在 -z≤x≤z, -z≤y≤z中 */
                    if (CLIPt (-dz, *z0 - *zmin, &tmin, &tmax))       /* 前 */
                        if (CLIPt (dz, -*z0 - 1, &tmin, &tmax)) {     /* 后 */
                            /* 如果到这儿，直线的一部分在 -z≤x≤z, -z≤y≤z */
                            /* -1≤z≤zmin中可见 */
                            *accept = TRUE;                           /* 直线的部分是可见的 */
                            /* 如果端点1(t = 1)不在该区域内，计算交线 */
                            if (tmax < 1.0) {                         /* 公式(6-37)至公式(6-39) */
                                *x1 = *x0 + tmax * dx;
                                *y1 = *y0 + tmax * dy;
                                *z1 = *z0 + tmax * dz;
                            }
                            /* 如果端点0(t = 0)不在该区域内，计算交线 */
                            if (tmin > 0.0) {                         /* 公式(6-37)至公式(6-39) */
                                *x0 += tmin * dx;
                                *y0 += tmin * dy;
                                *z0 += tmin * dz;
                            }
                        } /* 计算交线 */
        }
} /* Clip3D */
```

图6-55 扩展到三维规范透视投影视见体的Liang-Barsky二维裁剪算法。代码来自[LIAN84]。函数CLIPt在第3章的图3-45中

Sutherland-Hodgman 多边形裁剪算法可很容易地适应三维。我们通过使用六个而不是四个对S_H_CLIP(第3章)的调用来使用六个裁剪面而不是四个。

一旦裁剪完成，剩下的输出图元被使用M_{ort}或M_{per}投影到投影平面上，同时为了显示被变换到物理设备坐标视口。

273
~
274

6.5.4 在齐次坐标中裁剪

在齐次坐标中进行裁剪有两个原因。第一个与效率有关：有可能将透视投影规范视见体变换为平行投影规范视见体，所以总可使用为平行投影规范视见体而优化的单裁剪过程。但是，这个裁剪必须在齐次坐标中完成以保证正确的结果。在观察操作的硬件实现（第18章）中可提供这种单裁剪过程。第二个原因是作为非寻常齐次变换的结果出现的和来自使用有理参数样条（第11章）的点可能具有负的W，这些点可以在齐次坐标而不是三维中被正确裁剪。

对于裁剪，可以证明从透视投影规范视见体到平行投影规范视见体的变换是

$$M=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & \dfrac{1}{1+z_{\min}} & \dfrac{-z_{\min}}{1+z_{\min}}\\ 0 & 0 & -1 & 0\end{bmatrix},\qquad z_{\min}\neq -1 \tag{6-48}$$

回想式(6-40) $z_{\min}=-(vrp_z'+F)/(vrp_z'+B)$和公式(6-38)$\mathrm{VRP}'=SH_{\mathrm{par}}\cdot T(-\mathrm{PRP})\cdot[0\ 0\ 0\ 1]^{\mathrm{T}}$。图6-56显示将$M$应用到透视投影规范视见体的结果。

矩阵M与透视投影规格化变换N_{per}集成：

$$N'_{\mathrm{per}}=M\cdot N_{\mathrm{per}}=M\cdot S_{\mathrm{per}}\cdot SH_{\mathrm{par}}\cdot T(-\mathrm{PRP})\cdot R\cdot T(-\mathrm{VRP}) \tag{6-49}$$

通过对透视投影使用N'_{per}代替N_{per}，同时通过对平行投影连续使用N_{par}，我们可以在平行投影规范视见体中进行裁剪而不是在透视投影规范视见体中进行裁剪。

三维平行投影视见体由$-1\leqslant x\leqslant 1,-1\leqslant y\leqslant 1,-1\leqslant z\leqslant 0$定义。通过用$X/W$代替$x$，$Y/W$代替$y$，$Z/W$代替$z$，我们求出齐次坐标中的对应的不等式，它可导致

$$-1\leqslant X/W\leqslant 1,\ -1\leqslant Y/W\leqslant 1,\ -1\leqslant Z/W\leqslant 0 \tag{6-50}$$

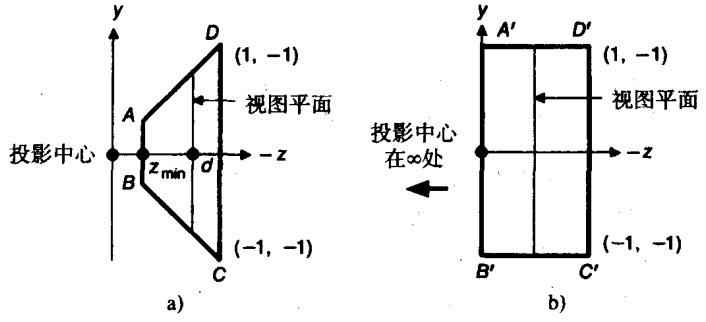


图6-56 在应用矩阵M前a)和后b)的规格化透视视见体的侧视图

对应的平面方程是

$$X=-W,X=W,Y=-W,Y=W,Z=-W,Z=0 \tag{6-51}$$

为了理解怎样使用这些约束和平面，我们必须分别考虑$W>0$和$W<0$的情况。在第一种情况中，我们可以将W与公式(6-50)的不等式相乘而不改变不等式的方向。在第二种情况下，相乘改变了不等式的方向。结果可表示为

$$W>0:\ -W\leqslant X\leqslant W,\ -W\leqslant Y\leqslant W,\ -W\leqslant Z\leqslant 0 \tag{6-52}$$

$$W<0:\ -W\geqslant X\geqslant W,\ -W\geqslant Y\geqslant W,\ -W\geqslant Z\geqslant 0 \tag{6-53}$$

在下面的情况中，即裁剪一般的线和点，仅需要使用公式(6-52)给定的区域，这是因为在使用M以前，所有的可见点都有$W>0$（通常$W=1$）。

但是，正如我们将在第11章看到的那样，有时需要直接在齐次坐标中用任意的W坐标表示点。因此，我们可能有$W<0$,意味着裁剪必须在公式(6-52)和公式(6-53)给出的区域内实现。图6-57将这些区域显示为A和B，同时也表明为什么必须使用两个区域。

在区域A中的点$P_1=[1\ 3\ 2\ 4]^{\mathrm{T}}$变换到三维点(1/4, 3/4, 2/4)，该点在规范视见体$-1\leqslant x\leqslant 1$，$-1\leqslant y\leqslant 1$，$-1\leqslant z\leqslant 0$中。点$P_2=-P_1=[-1\ -3\ -2\ -4]^{\mathrm{T}}$（不在区域$A$内，但在区域$B$内），

变换到与P_1同样的三维点，即（1/4，3/4，2/4）。如果仅对区域A裁剪，则P_2将被不正确地抛弃。这个可能性是存在的，由于齐次坐标点P_1和P_2相差一个常数乘子（-1），同时我们知道这些齐次点对应于同样的三维点（在齐次空间的$W=1$平面上）。

对于区域B上的点，这个问题有两种解决方法。一种是裁剪所有的点两次，一次裁剪一个区域。但是做两次裁剪是很昂贵的。更好的一个解决方法是先用$-W$使点反号，如对于P_2，然后裁剪它们。同样地，我们可以通过对每个端点乘以-1来正确地裁剪一条端点都在图6-57的区域B内的线，将点放在区域A内。

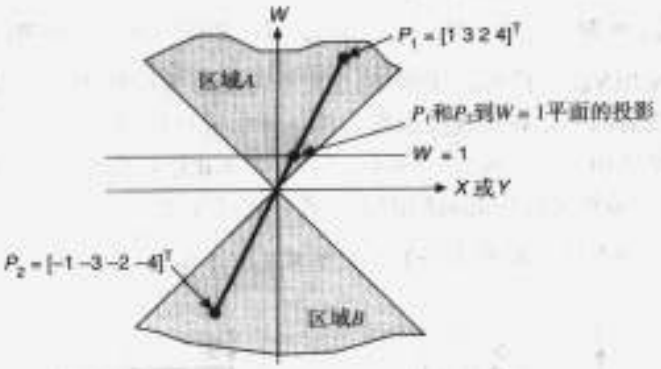


图6-57 点P_1和P_2都映射到$W=1$平面上的同一个点，在穿过原点和上述两点的直线上所有的点都类似。在齐次坐标中按照区域A的裁剪将不正确地抛弃P_2

对于直线，如P_1P_2（如图6-58所示，其端点有相反符号的W值），另一个问题产生了。直线到$W=1$平面的投影是两条线段，一条趋于正无穷，另一条趋于负无穷。现在的解决方法是裁剪两次，一次对一个区域进行裁剪。有可能每一次裁剪将返回一个可见线段。一个简单的方法是在区域A内裁剪该直线，再使直线的两个端点反号，再次对区域A裁剪。该方法保留了原来在齐次坐标中裁剪的目的之一：使用一个简单的裁剪区域。有兴趣的读者可参考[BLIN78a]做进一步的讨论。

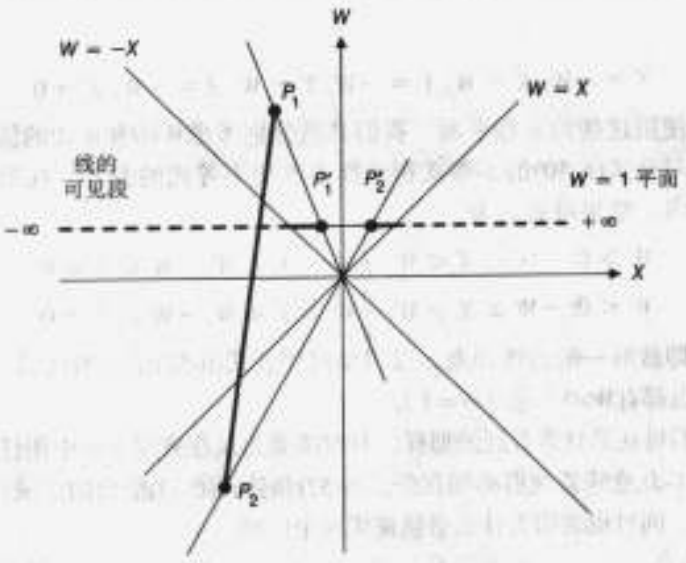


图6-58 线段P_1P_2投影到两个直线段，一个从P_2'到正无穷，另一个从P_1'到负无穷（实粗线表示在裁剪区域内，虚粗线表示在裁剪区域外）。该线段必须裁剪两次，对一个区域裁剪一次

275 | 277

给定公式(6-51)，Cohen-Sutherland 或Cyrus-Beck算法可被实际裁剪所使用。([LIAN84]给出Cyrus-Beck方法的代码；也可参见习题6.25。）惟一的不同是裁剪在四维中，不是三维。

6.5.5 映射到一个视口

在规格化投影坐标系统（也称为三维屏幕坐标系统）中，输出图元被裁剪。为本节的讨论，我们假设规范平行投影视见体已经被用于裁剪（如果该假设不正确的话，透视投影M转换透视投影视见体到平行投影视见体）。因此所有保留下来的输出图元的坐标在视见体$-1 \leqslant x \leqslant 1$，$-1 \leqslant y \leqslant 1$，$-1 \leqslant z \leqslant 0$中。

PHIGS程序员指定一个三维视口（该视见体的内容被映射到该视口）。三维视口包含在单位立方体$0 \leqslant x \leqslant 1$，$0 \leqslant y \leqslant 1$，$0 \leqslant z \leqslant 1$中。单位立方体的$z = 1$前平面被映射到可放到显示屏幕上的最大的正方形中。我们假设正方形的左下角在(0, 0)。例如，在水平分辨率为1024，垂直分辨率为800的显示设备上，正方形是区域$0 \leqslant x \leqslant 799$，$0 \leqslant y \leqslant 799$。通过抛弃其$z$坐标，单位立方体内的点被显示。因此点(0.5, 0.75, 0.46)可能会显示在设备坐标(400, 599)上。在可见面确定（第15章）时，每一个输出图元的z坐标被用来决定哪一个图元可见，哪一个被其他z值较大的图元隐藏。

在单位立方体内用坐标$x_{v.min}$和$x_{v.max}$等给定三维视口，于是从规范平行投影视见体到三维视口的映射可看成一个三步的过程。在第一步，规范平行投影视见体被平移，它的角点$(-1, -1, -1)$成为原点。这一过程受平移$T(1, 1, 1)$影响。接着，被平移的视见体缩放变换到三维视口的大小，缩放系数为

$$S\left(\frac{x_{v.max} - x_{v.min}}{2}, \frac{y_{v.max} - y_{v.min}}{2}, \frac{z_{v.max} - z_{v.min}}{1}\right)$$

最后，适当地缩放变换后的视见体由平移$T(x_{v.min}, y_{v.min}, z_{v.min})$平移到视口的左下角。因此，合成的规范视见体到三维视口的变换是

$$M_{VV3DV} = T(x_{v.min}, y_{v.min}, z_{v.min}) \cdot S\left(\frac{x_{v.max} - x_{v.min}}{2}, \frac{y_{v.max} - y_{v.min}}{2}, \frac{z_{v.max} - z_{v.min}}{1}\right) \cdot T(1, 1, 1) \quad (6\text{-}54)$$

注意，这与5.4节推导的窗口到视口的变换M_{wv}类似，但不相同。

6.5.6 实现小结

在全部的观察变换中有两种通用的实现。第一个（在图6-46中已经描述，且在6.5.1节到6.5.3节中已经讨论过）适合于输出图元定义在三维并且应用到输出图元的变换从不产生一个负的W的情况。其步骤如下： 278

1) 扩展三维坐标到齐次坐标。

2) 应用规格化变换N_{par}或N_{per}。

3) 被W除并映射回三维（在一些情况下，已知$W = 1$,所以不需要除法）。

4) 对平行投影或透视投影规范视见体（无论哪一个适合）进行三维裁剪。

5) 扩展三维坐标到齐次坐标。

6) 使用M_{ort}，公式(6-11)，执行平行投影，或使用M_{per}，具有$d = -1$的公式(6-3)，执行透视投影（因为规范视见体沿着$-z$轴）。

7) 使用公式(6-54)平移和缩放到设备坐标中。

8) 被W除从齐次坐标映射到二维坐标；除法影响透视投影。

步骤6和7使用一个矩阵乘法实现，且与图6-46中的阶段3和4对应。

第二种实现观察操作的方法的需求产生于当输出图元定义于齐次坐标中且可能$W<0$时，或

者当应用到输出图元的变换可能产生一个负W，或者当执行一个裁剪算法时。如6.5.4节所讨论的那样，其步骤如下：

1) 扩展三维坐标到齐次坐标。

2) 应用规格化变换N_{pu}或N'_{per}（包括M，公式(6-48)）。

3) 如果$W>0$，在公式(6-52)定义的空间上对齐次坐标进行裁剪；否则在公式(6-52)和公式(6-53)定义的两个视见体上对齐次坐标进行裁剪。

4) 使用公式(6-54)平移和缩放变换到设备坐标。

5) 被W除从齐次坐标映射到二维坐标；该除法影响透视投影。

6.6 坐标系统

在第5章和第6章已经使用了几种不同的坐标系统。在本节，我们总结所有的系统，并讨论它们之间的关系。还给出在各类参考文献和图形子程序包使用的同义词。图6-59显示坐标系统的接替关系，使用了本书通用的术语；因此，在任何特定的图形子程序包中，仅使用某几个坐标系统。我们已经为不同坐标系统选择名称以反映共同的用法，因此一些名称之间逻辑上并不
279 是一致的。注意术语空间（space）有时作为系统（ystem）的同义词。

从图6-59左边与实际的显示设备距离最远的坐标系开始，每个物体被定义在一个物体坐标系中。PHIGS称之为模型坐标系；术语局部坐标系也常用。如我们将在第7章深入讨论的一样，经常有一个模型坐标系的层次。

物体被变换到世界坐标系中，在该系统中，一个场景或整个物体通过模型变换表示在计算机中。该坐标系有时被称为问题坐标系或应用坐标系。

观察参考坐标系被PHIGS用作为一个定义视见体的坐标系。它也称为(u, v, n)系统或(u, v, VPN)系统。Core系统[GSPC79]使用一个类似的但是未命名的左手系，该左手系被使用后，在原点用眼睛或照相机朝$+z$看，离眼睛越远，z值越大，x指向右边，y朝上。

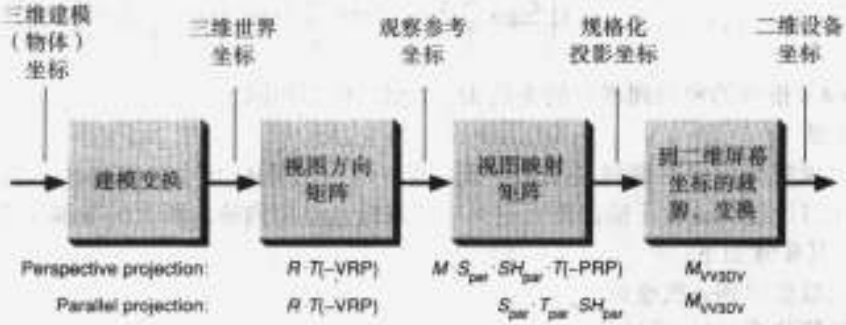


图6-59 坐标系统和它们彼此之间的关联。对于透视投影和平行投影，每个阶段的矩阵影响在该阶段应用的变换

其他图形包，例如Pixar公司的 RenderMan[PIXA88]，给观察参考坐标系加上约束，要求原点在投影中心，同时视图平面法线是z轴。我们称之为眼睛坐标系；RenderMan和某些其他图形系统使用照相机坐标系这一术语。参考6.5节，透视投影规格化变换的前三个步骤从世界坐标系转换到眼睛坐标系。眼睛坐标系统有时是左手的。

从眼睛坐标，我们接着到规格化投影坐标系或三维屏幕坐标，平行投影规范视见体的坐标系（和透视变换后透视投影规范视见体的坐标系）。Core系统称该系统为三维规格化设备坐标。有时，

该系统被称为三维逻辑设备坐标。术语规格化一般意味着所有的坐标值或者在0到1之间，或者在-1到1之间，但是术语逻辑一般表示坐标值在其他预先指定的范围，例如[0，1023]，该范围被典型地定义成对应于一些广泛使用的设备的坐标系统。在某些情况下，该系统不是规格化的。 280

从三维投影到二维产生我们所说的二维设备坐标系，也称为规格化设备坐标系，被[SUTH74]称为图像坐标系，或者RenderMan称为屏幕坐标系。其他使用的术语包括屏幕坐标、设备坐标、二维设备坐标、物理设备坐标（与前面提到的逻辑设备坐标相反）。RenderMan称空间物理形式为光栅坐标。

不幸的是，对这些术语没有一个标准用法。例如，术语屏幕坐标系被不同的作者用来意指前面讨论的最后三个系统，覆盖了二维和三维坐标以及逻辑和物理坐标。

习题

6.1 试写一个接受观察定义的程序，并计算出N_{par}或N_{per}，同时显示这个房子。

6.2 写出用于平行投影和透视投影的三维裁剪算法的程序。

6.3 假设$F = -\infty$且$B = +\infty$,试证明：对于平行投影，先在三维裁剪然后投影到二维的结果与先投影到二维再在二维裁剪的结果相同。

6.4 假设所有的物体在投影中心的前面，同时如果$F = -\infty$且$B = +\infty$,证明在透视投影规范视见体上对三维的裁剪然后进行透视投影的结果与首先透视投影到二维然后在二维裁剪的结果相同。

6.5 证明：S_{per}(6.5.2节)变换图6-54a的视见体为图6-54b的视见体。

6.6 写出以单位立方体进行裁剪的程序。将该程序推广到以任何长方体进行裁剪的一般情况，该长方体的表面垂直于各主轴。试问：该通用程序比适合于特定单位立方体的裁剪程序效率高还是低？

6.7 试写出用规范透视投影视见体进行三维裁剪的程序，并推广到由下式定义的视见体：

$$-a \cdot z_v \leqslant x_v \leqslant b \cdot z_v, \quad -c \cdot z_v \leqslant y_v \leqslant d \cdot z_v, \quad z_{min} \leqslant z_v \leqslant z_{max}$$

这是经过规格化透视变换第1到4步以后的视见体的一般形式。试问，这两种情况哪一种效率更高？

6.8 试写出用一般的六面视见体进行三维裁剪的程序，六个表面由下式定义：

$$A_i x + B_i y + C_i z + D_i = 0, \ 1 \leqslant i \leqslant 6$$

比较下面的两种情况所需要的计算工作量：

a. 对每一个规范视见体进行裁剪。

b. 应用N_{par}，然后用单位立方体进行裁剪。 281

6.9 考虑三维空间中的一条直线，从世界坐标点$P_1(6, 10, 3)$到$P_2(-3, -5, 2)$，一个在区域$-z \leqslant x \leqslant z$，$-z \leqslant y \leqslant z$定义的半无限的观察四棱锥由下面平面界定：$z = +x$，$z = -x$，$z = +y$，$z = -y$。投影平面在$z = 1$处。

a. 在三维空间裁剪该直线（用直线参数方程），然后将其投影到投影平面上。试问，在投影平面上裁剪后的端点是什么？

b. 将该线段投影到投影平面上，然后用二维计算对该投影进行裁剪。试问：在投影平面上裁剪后的端点是什么？

（提示：如果a和b的答案不一致，再试一下！）

6.10 当一个位于投影中心的“后面”的该物体用M_{per}进行投影，然后再进行裁剪时，则发生

什么样的情况呢？你的答案应该证明为什么一般不能先投影再裁剪。

6.11 考虑带有一个旋转窗口的二维观察操作。设计一个规格化变换，将窗口变换成单位正方形。如图6-60所示，窗口是在VRC坐标系统由u_{min}, v_{min}, u_{max}, v_{max}指定的。试证明，这一变换与一般的3D N_{par}相同，条件是投影平面是(x, y)平面，VUP有$-\sin\theta$的x分量和$\cos\theta$的y分量（也就是说，VUP到视图平面的平行投影是v轴）。

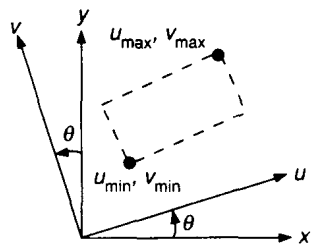


图6-60 一个旋转的窗口

6.12 6.4节的矩阵M_{per}定义一个一点透视投影。试问定义二点透视投影的4×4矩阵是什么形式呢？定义三点透视投影呢？（提示：可将M_{per}乘以不同的旋转矩阵。）

6.13 将M_{per}作用于那些z坐标值小于0的点，其效果是什么？

6.14 设计一个圆锥形（有圆形截面）视见体的裁剪算法。圆锥的顶点在原点，其内角为90°，圆锥体的轴是正的z轴，可考虑采用球坐标系统。

6.15 设计并实现一套实用的子程序，它用来产生一个由基本变换R, S, T的任意序列组成的4×4变换矩阵。

6.16 画一个用来确定在创建图像时应用的投影的类型的判定树，将这个判定树应用到本章中三维投影的图中。

6.17 评价使用Cohen-Sutherland算法与使用Cyrus-Beck算法执行三维裁剪的速度均衡。首先，写出一个汇编语言程序的版本。计算算法处理的每一个基本不同的情况执行的操作，同时相等地加权这些情况。（例如，在Cohen-Sutherland算法中，简单接受（trivial accept）、简单拒绝（trivial reject）和计算1，2，3或4交点是基本上不同的情况）忽略子程序调用
282 的时间（假设使用程序内联代码）。通过指令需要的周期数目分辨指令。

6.18 在习题6.17中，我们假设所有的情况都相同。这是一个好的假设吗？解释你的答案。将两个算法中的任何一个用到各类真实物体，使用各种视见体，来了解在真实的设置中各种情况的混合是怎样的不同。

6.19 平行投影的规范视见体作为$2\times2\times1$矩形平行管道形状。假设单位立方体在第一象限位置，一个角在原点。

a. 找到该视见体的规格化N'_{par}。

b. 找到对应的齐次坐标视见体。

6.20 给出VRP在窗口中间时房子的前视图、顶视图和侧视图的观察参数。对每个视图，PRP一定不同吗？为什么？

6.21 证明公式(6-48)确实将图6-56a的规范透视视见体变换成图6-56b的规范平行视见体。

6.22 在PHIGS中，观察定义允许视图平面在距离VRP有VPD的地方。重新推导规格化观察变换N_{par}和N_{per}以包括VPD。令VPD = 0确保你的结果特例化为本书中的结果。

6.23 重新用公式描述N_{per}和N_{par}，使之合成为矩阵$M_{general}$。

6.24 表明怎样从图6-35的六个观察参数变换为本章讨论的观察参数。写出并测试一个实用子程序来实现变换。

6.25 对于齐次裁剪平面$X=-W$，$X=W$，$Y=-W$，$Y=W$，$Z=0$，$Z=-W$，完成一个类似于表6-1的表，对由$W>0$，$-W\leqslant X\leqslant W$，$-W\leqslant Y\leqslant W$，$-W\leqslant Z\leqslant0$给定的$W>0$区域使用

Cyrus-Beck裁剪方法。

6.26 PHIGS使用图6-61显示的默认视见体，这些与图6-45显示的本章使用的规范视见体有所不同。修改N_{par}和N_{per}变换来映射视见体为PHIGS默认视见体。

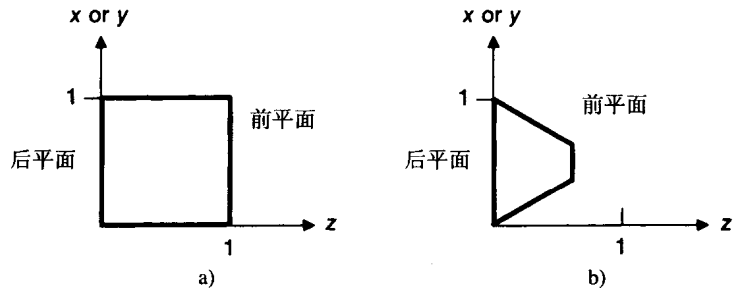


图6-61 由PHIGS使用的默认视见体，它与图6-45给出的本章使用的视见体稍有不同。a)平行，b)透视

6.27 立体对是同一场景的两个视图，产生于略有不同的投影参考点，但是具有同样的视图参考点。设d是立体偏差，即两个参考点之间的距离。如果我们将参考点作为眼睛，于是d是我们眼睛之间的距离。令P是我们眼睛中间的点。给定P, d, VRP, VPN和VUP，推导两个投影参考点的表达式。 283

第7章 对象的层次结构和简单的PHIGS系统

图形软件包是一个应用程序和图形硬件间的中介。一个软件包可以只支持最基本的输出图元和交互设备，也可以提供非常丰富的功能。在第2章，我们介绍了相对简单和低级的SRGP软件包，并指出了其局限性。在本章，我们将介绍一个相对丰富也更复杂的标准图形软件包PHIGS（程序员的层次交互图形系统[⊖]）。诸如PHIGS和GKS（图形核心系统）这样的标准软件包是由国际或各国官方标准化组织制定的；GKS和PHIGS就是由ANSI（美国国家标准研究所）和ISO（国际标准化组织）制定和发布的。这些标准的主要目的是使应用程序和软件开发过程具有可移植性。非官方标准是由个体公司或由公司和学校的联合组织开发、推动及授权使用；Adobe的PostScript和MIT的X Window就是两个这样的工业标准。

这里描述的软件包实际上是PHIGS的一个子集，因而称为SPHIGS（Simple PHIGS，读"ess-figs"）。它保留了PHIGS的主要功能，但作了简化和修改以适应一些简单的应用。SPHIGS也包括了一些PHIGS + 扩展中的增强功能。我们设计SPHIGS的目的是用最简单的方式介绍有关概念，而不是要提供一个和PHIGS严格地向上兼容的软件包。不过，一个SPHIGS应用程序通过简单的修改就可以在PHIGS下使用。脚注说明了SPHIGS和PHIGS之间的一些主要区别，但总的来说，除非有特别的说明，否则SPHIGS的特点在PHIGS中都有。

在SPHIGS和集成的光栅图形软件包（如 X Window系统的SRGP或Xlib软件包）之间有三个主要的区别。首先，为适应工程和科学应用的要求，SPHIGS使用三维的浮点坐标系统，并采用了第6章讨论的三维观察流水线。

其次，更大的不同是SPHIGS维持一个结构的数据库。所谓结构是一个基本图元、属性和其他信息的逻辑的集合体。程序设计人员可以通过少量的编辑命令修改数据库中的结构。SPHIGS保证屏幕图像是所存储的数据库内容的精确显示。结构不仅包含图元及其属性的定义，而且包括对从属的子结构的调用。这样，SPHIGS显示了类似编程语言中的过程的某些特点。特别是，就像由过程调用子过程，从而引入过程的层次一样，由结构调用其子结构，也就引入了结构的层次。当我们可以控制所调用的子结构的几何属性（大小、方向和位置）和外观（颜色、线型和线宽等）时，这种层次化的结构就显得有用。

第三个不同在于SPHIGS是在抽象的三维世界坐标系中，而不是在二维的屏幕坐标系中进行操作，所以它不支持对像素的直接操作。由于这些不同，SPHIGS和SRGP适应不同的需要和应用，正如我们在第2章所指出的那样，每一个软件包有它适用的地方，同时任何一个软件包都不能满足所有的需要。

因为其支持结构层次的特点，SPHIGS特别适合于基于构件和子构件的层次模型的应用；事实上，SPHIGS的结构层次可以看成是特殊功能的造型层次。因而在讨论利用SPHIGS进行几何造型的特点之前，我们在7.1节先讨论通常的几何造型。在7.2节至7.9节，我们讨论如何创建、显示

⊖ 本章中PHIGS这个术语，也包括PHIGS的扩展——PHIGS +，PHIGS + 支持多面体、曲线和曲面等高级几何图元，也包括使用光照、明暗处理、深度提示等的绘制技术，详见第14~16章。

和编辑SPHIGS的结构数据库。7.10节讨论交互，尤其是关联拾取。余下的部分讲述SPHIGS中所不包含的PHIGS属性，讨论实现的问题，并以SPHIGS和其他层次编程方法的评价作为结尾。

7.1 几何造型

7.1.1 什么是模型

在自然科学和社会科学的教程中我们遇到过很多模型的例子。例如，大家可能熟悉原子的波尔模型，在此模型中，电子围绕由中子和质子组成的原子核运动。其他的例子包括生物学中的非约束的指数增长模型，试图描述经济领域某些问题的宏观和微观经济学模型。模型是具体
286 或抽象的实体的一些（无须全部）属性的表示。实体模型的作用是让实体的结构或行为可以被形象化和理解，并且提供一个可实验的手段，以及预测输入或变化对模型的影响。在自然科学、社会科学以及工程中的定量模型常常表达为方程组，可以通过改变独立变量、系数和指数的值进行实验。通常模型会对所描述的实体的结构和行为进行简化，以使模型易于形象化，或使由方程组表示的模型易于通过计算推导。

本书中，我们只讨论基于计算机的模型，特别是那些借助于图形学解释的模型。图形学可用于创建和编辑模型，获得模型参数的值，并使模型的结构和行为可视化。模型本身和创建模型并使之可视化的图形方法是不同的，模型，如人口模型，无须任何内在的图形属性。在通常的模型类型中，用到计算机图形学的模型有：

- 组织模型：用层次的方法表示组织机构和分类，如图书馆的分类目录和生物学分类法。这些模型有多种有向图表示，如组织结构图表。
- 定量模型：用方程来描述经济、金融、社会、人口、气候、化学、物理和数学系统。它们常用图表和统计图表描述。
- 几何模型：有明确的几何定义的构件以及构件之间的关系构成的集合，包括工程和建筑结构、分子和其他化学结构、地理结构以及车辆等。这些模型通常由结构图或模拟现实的合成图像表示。

计算机辅助建模可以帮助医药研制人员模拟针对某种疾病的新混合物的化学行为，帮助航空工程师预测超音速条件下机翼的变形，帮助飞行员学习驾机飞行，帮助核反应堆专家预测各种设备发生故障的影响并提出适当的补救措施，以及帮助汽车设计师测试在撞车时乘客车厢的完整性。在这些以及其他的更多的例子中，利用模型进行实验比用实物进行实验更容易更经济也更安全。事实上，在许多情况下，如航天飞机飞行员的训练和核反应堆安全性的实验，建模和仿真是惟一可行的获取系统知识的方法。正是因为这些原因，计算机建模正在逐渐代替传统的方法，如风洞实验。工程师和科学家可以利用数字的风洞、显微镜、望远镜等进行许多实验。这种利用数值分析的方法进行模型的模拟和动态显示的方法正成为科学中的一种新范例，逐渐代替传统的理论和物理实验分支。建模和模拟的成功取决于一个好的模型和输入，在建模中尤其要避免发生所谓的“输入垃圾产出垃圾”的情况。

287 模型并不一定需要包含内在的几何数据；像组织机构模型这样的抽象模型并不是面向空间的。尽管如此，许多这样的模型能被几何地表示，例如，组织机构能用机构图表示，临床药剂评价的结果可以用直方图表示。即使是表示内在的几何物体的模型，也未必要采用图形表示或模型的视图来描述。例如，我们可以选择用多面体或是用曲面的集合来表示机器人，同时我们也能规定诸如以什么视角、以何种投影关系以及要求达到怎样的真实效果，来对其进行真实感的描述。我们也可以选择以图形的方法来显示模型的结构或者模型的行为，比如，我们也许需要显示一个超大

规模集成电路芯片上的物理电路，同时也需要显示其作为输入和时间函数的电子和逻辑行为。

7.1.2 几何模型

几何模型或图形模型由于使用内在的几何性质描述构件，很自然地可用图形表示。几何模型可表示为如下成分：

- 构件的空间位置分布和形状（即实体的几何属性）和其他影响外表的属性，如颜色。
- 构件间的连接关系（即实体的结构与拓扑属性），这些连接属性可以抽象定义（如用网络的关联矩阵或层次关系的树结构）；也可以由其内在的几何定义（如集成电路的通道的维数）。
- 和构件相联系的由应用所定义的数据值和属性，如电子文本和描述文字。

和几何模型相关的是其处理算法，如对离散电路模型的线性电路分析、对机械结构的有限元分析以及原子模型的极小能量方法。

直接被存储在模型中的部分和在分析显示前通过计算得到的部分之间通常有一个平衡，即通常所说的空间-时间的平衡。例如，在一个计算机网络的模型中，可直接存储所用的连接关系，也可以每次当需要一个新的视图时，通过连接矩阵用一个简单的图布局算法计算得出。为了满足分析和显示的要求，一个模型需要保持足够的信息，但是信息表达的格式和编码技术的选择，取决于应用和时间-空间的平衡。

7.1.3 几何模型中的层次

几何模型通常具有一个由自底而上的构造过程决定的层次结构：构件被作为构成高一层次实体的部件（building block），同时，新的实体又作为更高层次的实体的部件。如同大型的编程系统一样，其层次结构很少是以严格的自底向上或自顶向下的方法建成，其根本的是最后的层次结构而非其建构的过程。物体具有层次结构的属性非常普遍，因为几乎没有实体是完全独 288
自为一体的，一旦我们将一个物体分为其零件的集合，我们已经至少创立了一个两层的层次结构。在一些特殊情况下，每个物体只被其上一层次的物体包含一次，这种层次结构可用树来表示，其中每个物体是一个节点，物体间的包含关系是连接节点的边。在更常见的情况下，物体可能被包含多次，这种情况的层次结构可用无环有向图（DAG）表示。图7-1是一个物体层次结构的简单例子，表示一个基本的机器人透视图；图7-2a用DAG显示了机器人的结构。我们可以复制多重包含的物体，把无环有向图转化为图7-2b所示的树。为表示方便，单向的箭头被省略，因为节点间的关系可以通过树中节点的相对位置表示，在上面的节点包含在下面的节点。

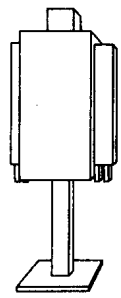


图7-1 一个简单机器人的透视图

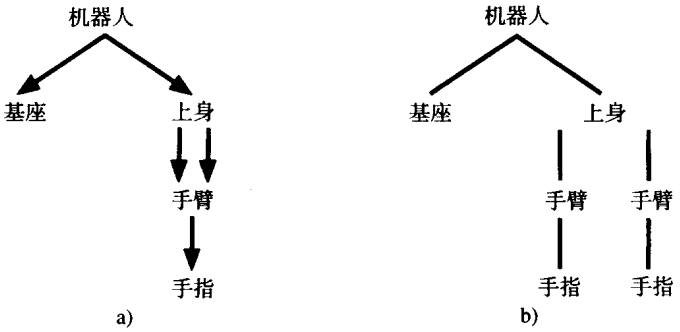


图7-2 机器人构件的层次结构。a) 无环有向图，b) 树结构

机器人由一个在基座及其上上可转动的上身组成。上身由头和躯干组成，头可绕躯干转动；躯干还和两个独立的手臂相连，手臂可分别绕过肩膀的水平轴转动。手臂由一个固定的部分即手部和一个与手保持平行并可滑动的手指组成以完成基本的抓取动作。这样，一个手指构件在手臂中被包含一次，而在一个躯干中则包含两个手臂构件。在本章中我们将讨论这个机器
289 人的创建，它的形状以正投影视图的形式显示在7-7b给出的屏幕的2~4窗口中。

虽然具有层次结构的物体既可以由几何图元组成，也可以由其低层次的包含物组成，但表示机器人层次结构的有向图或树只显示了其和所包含的低层物体之间的关系。这如同在高级的过程语言中，过程的层次关系图用来显示程序的调用结构一样。重要的是，一个复合对象如何被构造成层次结构是由设计人员决定的。例如，机器人可以以两层的层次结构来构造，根对象由作为几何图元的基座、头、躯干，以及同样由几何图元表示的相连的两个手臂构成。

许多系统（如计算机网络和化学设备）可以用网络图表示，其中的对象不仅被多次地包含，而且还可以拥有任意的连接关系。这样的网络可以表示为可能包含环的图，但在层次结构中，子系统多次出现时，系统依然能显示对象间的包含层次。

为了简化构造复杂对象及其模型的任务，我们通常用由应用所决定的原子构件作为基本的部件。在二维中，这些构件通常用标准符号形状的塑料模板或计算机绘图模板来绘制。在绘图程序中，这些形状相应地由基本几何图元，如直线、长方形、多边形、椭圆、圆弧等构成。在三维中，圆柱体、平行六面体、球体、锥体、旋转曲面等常被用来作为部件。这些三维形体部件可以由低层次的几何图元（如三维多边形）来定义；在这种情况下，光滑曲面以牺牲其精度为代价，可以由多个多边形来逼近。在可以直接处理自由曲面和实体的高级几何造型系统中，诸如参数多项式曲面这样的形状以及诸如圆柱体、球体、圆锥体这样的实体，可以将其自身作为图元，并可以直接解析地定义，从而不损失精确性（参见第11、12章）。本章中，我们将用“对象”的概念描述由其自身的模型坐标系以几何图元或低层的物体定义的构件，它不仅包含几何数据，也包含相关的应用数据。这样，一个对象就是一个作为组成部分的形状以及其所用的数据。

于是，我们可以构建适于多种用途的层次结构：

- 用模块的方法建立复杂的对象，通常可以通过反复地调用具有不同几何属性和外观属性的部件来实现。
- 提高存储的效率，因为只存储反复使用的对象的索引，而不必每次存储完整的对象定义。
- 允许方便的更新传播，因为在一个作为部件的对象定义中的修改将自动传播到所有使用这个部件的高层次的对象中去（既然它们已代表一个已更新的版本）。这和编程语言中的过程层次类似，对一个过程体的改变将会反映到那个过程的所有调用中。

应用程序可以使用多种技术来编码层次模型。例如，一个网络或关系数据库可以被用来存
290 储对象的信息以及对象之间关系的信息。另一种更有效的方法，是由应用程序维持一个自定义的链表结构，包括存储对象的记录和描述关系的指针。在一些模型中，对象之间的关系也是对象，它们必须在模型中以数据记录的形式来表示。另外一种方法是使用面向对象的数据库。面向对象的编程环境，如SmallTalk[GOLD83]、MacApp[SCHM86]和ET + + [WEIN88]正在被越来越多地应用于图形应用程序中，用于存储几何对象的模型信息。

1. 相互关联

在许多网络中，对象被置于特定的位置（或由用户交互或由应用程序自动决定），然后被相互关联。这种关联可以是抽象的，从而是任意形状的（在层次结构图或网络图，如组织机构图或工程进度图中）；也可以是本身具有意义的几何结构（如超大规模集成电路芯片）。如果

是抽象的关联，我们可以用多种标准协定得出层次结构图或网络图，并且我们也能用诸如线型、线宽或颜色等属性来表示各种不同的关系（如组织机构图中的虚线表示）。对于那些形如集成电路中半导体部件间的电路连接情况的关联，本身就是对象。抽象的和非抽象的关联通常都被约束在水平或垂直的方向（有时称为曼哈顿布局模式）上以简化可视化和物理构建过程。

2. 对象层次间的参数传递

作为部件调用的对象必须置于其父对象的适当的位置，为了适应此要求，通常必须改变对象的大小和方向。齐次坐标矩阵用于第5章的图元变换以及第6章的视见体规格化；在层次结构的模型中，对低层对象使用缩放、旋转和平移矩阵是常见的操作。Sutherland首先在Sketchpad系统[SUTH63]中使用这种图形造型的能力，并使用了*宿主*（master）这个术语来表达对象的定义，*实例*（instance）表达调用几何变换的情况。如在4.3.3中讨论的那样，在使用层次结构显示列表的图形系统（又被称为结构显示文件）中，在硬件中实现了宿主实例的层次结构，使用子循环调用和浮点算术单元来实现变换。为了区分用于视见体规格化的变换和用于建构层次结构对象的变换，我们通常将后者称为*模型变换*。在数学上，模型变换和规一化变换间没有区别。

和过程层次结构相比，我们有时称一个父对象“调用”层次结构中的一个子对象，并且将父对象坐标系中的比例、方向、位置等“几何参数”传递给子对象。我们可以看到，像SPHIGS这样的支持对象层次结构的图形软件包，能对图元的顶点和实例化的子对象的顶点做存储、组合和使用变换矩阵。但在7.5.3节我们将看到SPHIGS的参数传递机制并不像面向过程语言那样普遍。 291

7.1.4 模型、应用程序和图形系统间的关系

至此，我们大致地讨论了模型，并着重讨论了层次结构的几何模型和模型变换。在讨论SPHIGS前，让我们简要地回顾一下图1-5显示且后来在图3-2中进一步解释的图形的概念模型，以进一步揭示模型、应用程序和图形系统间的内在关系。在图7-3中，应用程序分为五个子系统，记为a)到e)：

a) 通过增加、删除和替换模型中的信息来建立、修改和维护模型。

b) 遍历（扫描）模型以提取用于显示的信息。

c) 遍历模型以提取信息用于分析模型的行为和性能。

d) 显示信息（如绘制几何模型、输出分析结果）和用户界面工具（如菜单、对话框）。

e) 执行不直接用到模型和显示的各种应用任务（如内务处理）。

术语*子系统*并不一定指编码的主要模块——少量的调用或一个短的过程都足以实现一个子系统。并且，一个子系统可以贯穿应用程序的始终，而不是集中在一个独立的程序模块中。图7-3只是简单地显示了逻辑构件而不是程序结构构件，并且不同于建立、修改、分析或显示模型的过程，至于调用的是模型的特定模块还是模型维持代码的特定部分，并不总是清楚的。这可用以下例子说明，比如，电路分析模块是模型定义的一部分，因为它描述了模型的行为。对于一个使用传统的过程语言（如C和Pascal）的程序员来说，如果认为模型主要包括数据，图7-3会更好理解，而熟悉面向对象的编程语言的人认为模型同时包括数据和过程会更自然一点。

在许多应用程序中，特别是工业应用中，存在着“80/20”规则：程序的主要部分（80%）处理实体的建模，只有少数部分（20%）处理模型的显示。换句话说，在许多如CAD那样的应用程序中，模型的图形表达只是一种达到目的的手段，用于分析、物理建构、数控加工、以及其他的后处理。当然也有许多应用就是用于图形本身，如画图、绘图、电影和视频制作、飞行模拟场景的动画等，这其中除绘画外，都需要一个模型以便于产生绘制的图像。简言之，大多数的图形涉及到模型（和模拟），说“图形学就是建模”是有道理的。第11、12和20章将讨论这个重要的内容。

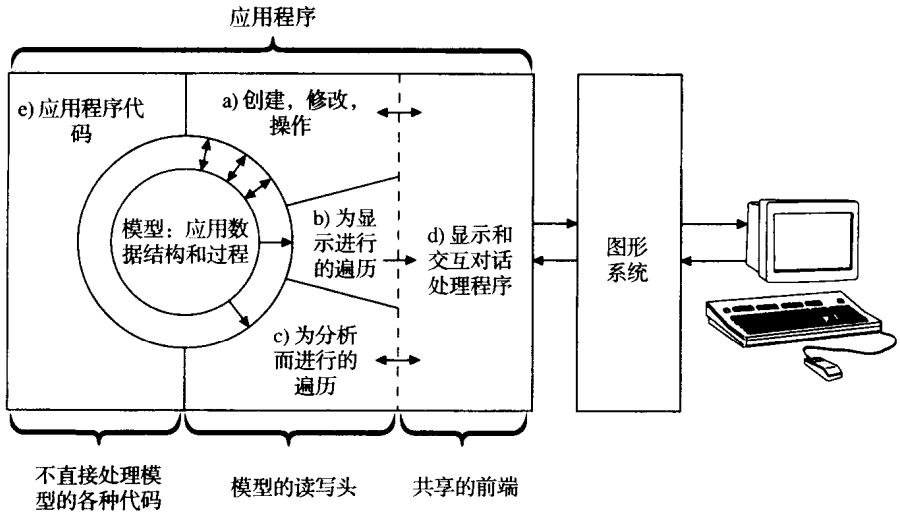


图7-3 应用模型及其读写

7.2 保留模式图形包的特点

在讨论应用软件、应用模型和图形软件包的作用时，我们先看一下图形软件包的功能和模型改变后的影响。如第2章中提到的，SRGP在瞬时模式下进行操作，不能保持传递给它的图元和属性的记录。这样，对一个应用对象的修改或删除需要除去屏幕上的所有相关信息，然后有选择地改变或全部重新生成屏幕信息，这些都需要应用程序从模型中重新定义图元。另一方面，PHIGS在保留模式下工作：它保持一个所有图元和相关信息的记录，以允许其后的编辑和自动更新显示，这样它就减轻了应用程序的负担。

7.2.1 中央结构存储库及其优点

PHIGS在一个叫作*中央结构存储器*（CSS）的特殊用途数据库中存储信息。PHIGS中的*结构*是一些*元素*的序列，这些元素包括图元、外观属性、变换矩阵和对从属结构的调用，其目的是定义一致的几何对象。这样，PHIGS有效地存储了一个特殊用途的造型层次结构，并把造型变换和其他属性作为参数传递给从属结构。请注意CSS造型的层次结构与存储宿主-实例的层次结构的硬件分层显示列表的相似性。事实上，PHIGS可以看成一个与设备无关的层次化显示列表的软件包规范；虽然给定的实现为特定的显示设备而被优化，但编程人员不需要关心其细节。尽管许多PHIGS的应用是纯软件的，但越来越多的应用利用硬件来实现部分或全部的软件包的功能。

292
~
293

和显示列表一样，CSS复制存储在应用程序的通用模型/数据库内的几何信息，以实现快速的*显示遍历*，即用于计算模型的新视图的遍历。CSS的主要优势在于当应用更新CSS时，能自动进行快速的屏幕刷新。仅这个属性就使得在应用数据库和CSS间复制几何数据变得物有所值，尤其当PHIGS应用程序使用独立的处理器作为“遍历引擎”以使CPU减轻显示所需遍历的计算负担（见第18章）的时候。诸如改变变换矩阵等较小的编辑功能，也能在PHIGS中有效地完成。

CSS的第二个优势是可以自动地关联拾取：软件包确定由用户拾取的图元在层次结构中的实体和位置（见7.10.2节和7.12.2节）。关联拾取功能展示了将使用频率高的功能纳入基本软件包的常用技巧。

CSS的第三个优势是其编辑功能，与层次建模的特点相结合，使用户能容易地创建各种动态效果，如运动过程的动力学，其中时变的变换可用于在父对象中对子对象进行缩放、旋转和定位。例如，我们可以用施加在子结构上的旋转表示我们的简单的机器人模型的每一个关节

（如手臂是上身的一个可旋转的子结构），并通过编辑一个旋转矩阵来动态地旋转手臂。

7.2.2 保留模式软件包的局限性

尽管CSS（作为一个针对显示和快速增量式刷新的特殊用途的实体）实现了一些常用的造型操作，但是它不是对每一种造型的应用都是必需的和充分的。CSS不总是必需的，因为应用程序在模型被修改时能自行完成屏幕刷新，能自行完成关联拾取（即使需要可观的工作量），并且能通过定义对象、传递变换以及其他参数的过程来实现对象的层次化结构；CSS通常是不充分的，因为在大多数的应用中，仍然需要单独地来建立和更新应用的数据结构，以记录每个应用对象的所有相关的数据。这样，将存在一个所有几何数据的副本，并且这两个表示间必须保持同步。因为这些原因，一些图形软件包不需要进行任何类型的结构存储，就可支持浮点坐标以及规一化的二维和三维观察功能。这种瞬时模式的软件包的合理性在于维持CSS的所需的代价，因为通常应用程序本身都维护一个应用模型，这个模型已足以满足屏幕刷新的要求。

对于那些在相继的图像之间有很大的结构改变的应用，保留模式软件包就不能满足需要。例如，在“数字风洞”中进行机翼分析时，其表面是由三角形网格表示的，当机翼受空气动力作用时，其三角网格的大多数顶点将发生位置改变。在这种情况下编辑结构数据库就没有意义，因为每个新图像中的大多数数据都发生了变化。事实上，我们并不推荐编辑PHIGS的结构数据 294
库，除非被编辑的元素的数目远小于整个被显示的网格。PHIGS所提供的编辑工具只是最基本的，比如，可以很方便地改变一个模型变换；但是要改变一个多边形的顶点就需要先删除原多边形然后再重新定义新的版本。通常情况下，应用程序会为显示遍历，而不是为大规模编辑做优化，因为显示遍历是最常用的操作。另外，任何情况下应用模型都必须要更新，而更新一个数据库总是比更新两个数据库更方便快捷。

由于这些局限性，一些PHIGS的实现提供了瞬时模式输出的功能，尽管这些功能由于技术原因并没有成为官方PHIGS规范的一个部分。

7.3 定义和显示结构

上一节已经讨论了PHIGS和SPHIGS的基本的属性，本节中，我们开始详细描述SPHIGS软件包，除非特别注明，否则所有讨论的内容都适用于PHIGS。SPHIGS结构上允许的操作包括以下几项：

- 打开（启动编辑）和关闭（终止编辑）。
- 删除。
- 插入*结构元素*（三种主要类型的结构元素是图元、属性（包括定义模型变换的属性）和调用子结构的元素）。一个元素是每当一个*元素生成过程*被调用时生成并被插入当前结构中的一个数据记录，它存储了那个过程的参数。
- 删除结构元素。
- 提交显示（可以类比于在布告栏提交照片），由定义如何将浮点坐标映射到屏幕坐标的视图操作决定。

7.3.1 打开和关闭结构

创建一个结构，比如构造一个形成图7-2中机器人手臂的图元和属性的集合，我们把对元素生成过程的调用与以下调用归为一类：

```
void SPH_openStructure (int structureID);
void SPH_closeStructure (void);
```

这些过程对结构进行的操作，类似于标准打开和关闭文件命令对磁盘文件进行的操作。但和磁盘文件所不同的是，每次只能打开一个结构，并且在打开时定义的所有元素都存储在此结构中。一旦关闭，结构可重新被打开以供编辑（见7.9节）。

结构有另外的两个特点：首先，图元和属性可以仅仅被确定为结构的元素（如同C程序中所有语句必须在过程或函数中定义）。对于一个结构中可以存储多少个元素并无规定；结构既
295 可以是空的也可以包含任意多个元素，仅受存储空间的限制。当然，构成一个结构的所有元素通常应该是定义了一个对象的逻辑上相互关联的集合。

其次，结构的标识号是整数，由于这些标识通常只由应用程序使用，而非交互用户使用，所以编程人员可以用符号常量表示这些标识，而一般并不需要用字符串这样的形式。整数标识可以允许在应用程序数据结构中的对象和相应对象的结构标识间建立方便的映射。

7.3.2 定义输出图元及其属性

生成输出图元元素的过程和在SRGP中的看起来很相似，但两者间有一些重要的差别。首先，点由三个双精度的坐标（x、y、z）定义，其次，这些过程把元素放在CSS中当前开放的结构中，而不是直接改变屏幕的图像——显示结构是一个独立的操作，将在7.3.3节中讲述。在本章，*图元*这个术语是三个相关实体的简称：元素生成过程（如SPH_polyLine），由该过程生成的结构的元素（如polyLine元素），以及在对中央结构存储器的显示遍历过程中，执行元素所产生的显示图像。SPHIGS通过模型变换和视图操作转换图元的坐标来实现图元元素的执行，这些操作包括将坐标裁剪、转换到视见体，然后*光栅化*（转换到像素）。SPHIGS中的属性比SRGP定义得更细，即每个图元都有其自己的属性。这样，像颜色和线型这样的属性是分类的，以便程序员可以在保留多边形和文本颜色的同时重置直线的当前颜色。

1. 图元

SPHIGS比SRGP有更少的图元，因为与一些SRGP的图元（如椭球体）等价的三维实体在实现时，尤其在变换、裁剪和扫描转换中，是耗费计算时间的。

大多数SPHIGS的图元和SRGP相应的图元具有相同的定义方法（除了点是三维的情况）：

```
void SPH_polyLine (int vertexCount, point *vertices);
void SPH_polyMarker (int vertexCount, point *vertices);
void SPH_fillArea (int vertexCount, point *vertices);     /* 和SRGP_polygon相似 */
void SPH_text (point origin, char *str);                  /* 不是完整的三维，参见7.7.2节 */
```

注意，SPHIGS不验证填充区域（或面片，下面将进行描述）是否是平面的，如果不是平面的将会产生不可预知的结果。

现在，我们考虑图7-4所示的简单房屋的定义。我们可以通过把每个面（也称为*面片*）定义为一个填充区域在SPHIGS中描述房屋，以不必要地重复定义和存储（在CSS中）每一个共
296 享的顶点为代价。这种重复还会减慢显示的产生速度，因为视图操作的计算必须对每一个顶点处理多次。利用共享顶点的间接面片索引可以大大提高存储和处理时间上的效率。我们把一个多面体看成面片的集合，每个面片由一个顶点索引表表示，每个索引是一个指向顶点表的指针。我们可以用以下的记号描述一个多面体的说明：

```
Polyhedron = {VertexList, FacetList}
VertexList = {V1, V2, V3, V4, V5, V6, V7, V8, V9, V10}
V1 = (x1, y1, z1), V2 = (x2, y2, z2) . . .
FacetList = {front = {1, 2, 3, 4, 5}, right = {2, 7, 8, 3}, . . . bottom = {. . .}}
```

SPHIGS提供这种有效的形式来定义*多面体*图元。在SPHIGS术语中，多面体是面片的集合，

它可以封闭也可以不封闭一个体。在一个封闭的多面体中，如我们的房屋，每个顶点通常至少被三个面片所共享，所以利用间接方法进行定义的效率是很高的。多面体的外观也受应用到填充区域的同样属性的影响。

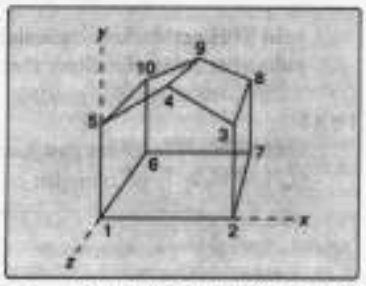


图7-4 一栋由点集和面片集定义的简单房屋

多面体元素生成过程中的面片表是以存储面片描述的连接集合的整数（SPHIGS中的类型"vertexIndex*"）数组形式表示的。每个面片描述是$V+1$个整数的序列，其中V是面片包含的顶点数，前V个整数表示顶点表的索引，最后一个整数(-1)是面片的结束标志。这样，我们可以通过发送数组1, 2, 3, 4, 5, -1, 2, 7, 8, 3, -1, …来描述房屋的面片（通过过程的第四个参数，下面将进行描述）：

```
void SPH_polyhedron (int vertexCount, int facetCount, point *vertices, vertexIndex *facets);
```

注意SPHIGS的绘制算法要求区别一个面的两面，即朝内的和朝外的。这样，一个面片的顶点应该以逆时针顺序（右手规则）定义，视点位于面片外侧。[⊖] 297

作为一个简单的例子，下面的C程序创建了由单个多面体组成的结构，对图7-4的房屋进行造型。

```
SPH_openStructure (HOUSE_STRUCT);
    SPH_polyhedron (10, 7, houseVertexList, houseFacetDescription);
SPH_closeStructure();
```

基本上，SPHIGS只支持多边形几何图元，更高级的三维造型图元将在以后讨论——第11章的多项式平滑曲线及曲面和第12章的实体图元。

2. 属性

图7-5所列出的过程生成属性元素。在显示遍历过程中，属性元素的执行按以下模式改变属性的值：新的值将一直起作用，直到被显式地改变。如同在下一节和7.7节中讨论的那样，显示遍历过程中，属性和图元是联系在一起的。

```
polyLine:
    void SPH_setLineStyle (CONTINUOUS / DASHED / DOTTED / DOT_DASHED lineStyle);
    void SPH_setLineWidthScaleFactor (double scaleFactor);
    void SPH_setLineColor (int colorIndex);

fill area and polyhedron:
    void SPH_setInteriorColor (int colorIndex);
    void SPH_setEdgeFlag (EDGE_VISIBLE / EDGE_INVISIBLE flag);
    void SPH_setEdgeStyle (CONTINUOUS / DASHED / DOTTED / DOT_DASHED lineStyle);
    void SPH_setEdgeWidthScaleFactor (double scaleFactor);
    void SPH_setEdgeColor (int colorIndex);

polyMarker:
    void SPH_setMarkerStyle (MARKER_CIRCLE / MARKER_SQUARE/... markerStyle);
```

图7-5 生成属性元素的过程

⊖ SPHIGS要求每个面片的一面被定义为朝外，即使这个多面体的面不封闭。另外，我们认为朝内的面是不可见的。

```
    void SPH_setMarkerSizeScaleFactor (double scaleFactor);
    void SPH_setMarkerColor (int colorIndex);

text:
    void SPH_setTextFont (int fontIndex);
    void SPH_setTextColor (int colorIndex);
```

图 7-5（续）

填充区域的属性不同于在SRGP中的属性，填充区域和多面体图元都有分别单独定义了属性的内部和边界。内部只有颜色属性，而边界有颜色、线型、宽度等属性。另外，边界的可见性可以通过边界标志属性加以关闭，这一点对多种绘制模式都有用，这将在7.8节讨论。

线/边界宽度和标记大小是以非几何的方式定义的：它们不是通过世界坐标系单位定义的，因而不支持几何变换。因此模型变换和观察操作可以改变一条直线的显示长度，但不能改变其宽度。类似地，在不连续线型中的长划线的长度也是和作用于其上的变换相独立的。然而不同
298 于SRGP的是，像素不作为测量的单位，因为其大小是设备相关的。一般是对每一设备设置一个标称宽度/大小，从而使得在每个输出设备上对宽度/大小单位有基本相同的外观。SPHIGS的应用规定的单位为该标称宽度的倍数（非整数）。

SPHIGS因为三个原因而不支持图案：首先，SPHIGS保留图形以在二值显示系统中模拟彩色的光照效果；其次，对大多显示系统而言，实现彩色系统图案区域的光滑光照效果需要太多的计算量；最后，PHIGS中，称为“剖面线”的几何图案即使在有实时变换硬件的显示系统中也是非常耗时间的。

7.3.3 提交结构进行显示遍历

SPHIGS记录在CSS中新生成的结构，但直到应用程序提交了服从特定观察规范的结构后，才显示这个新的结构[⊖]。SPHIGS然后在CSS中对结构元素进行显示遍历，从第一个到最后一个按顺序执行每个元素。如果图元的一部分在视图中可见，则图元将显现在屏幕图像上。执行一个属性元素（包括几何变换和外观属性）将改变存储在状态向量（属性遍历状态）中的属性集，该属性集将模式地作用于其后遇到的图元上。这样，属性以显示遍历的顺序作用于每个图元。

下面的过程在由SPHIGS内部维护的提交结构表中添加一个结构：

```
void SPH_postRoot (int structureID, int viewIndex);
```

术语根表示在提交一个调用子结构的结构S时，我们实际上提交的是一个以S为根的层次化的DAG，称为结构网络。即使提交的结构不调用子结构，我们也称之为根，所有被提交的结构都是根。

*viewIndex*参数选择视图表（将在下一节讨论）中的项，这个项规定了结构的图元的坐标如何映射到屏幕的整数坐标空间。

我们可以通过删除CSS中的结构（或元素）的方式从屏幕上删除一个对象的图像（见7.9节），也可以通过使用SPH_unpostRoot过程从提交的根表中去除特定的根，而不从CSS中删除结构：

```
void SPH_unpostRoot (int structureID, int viewIndex);
```

⊖ PHIGS和SPHIGS提交的结构显示的方式完全不同。PHIGS通常的机制是，视图规范是一个结构元素，从而在显示遍历时，可以像其他元素一样被改变和编辑。许多现有的PHIGS实现也支持SPHIGS型简单的结构提交机制。

7.3.4 观察

1. 合成照相机

把三维软件包看成一个合成照相机，来获得由几何定义了的对象所占据的三维世界的“快照”。生成一个结构等价于把一个对象定位于照相空间，提交一个结构可以类比为激活预先设置在某个场景中的照相机对物体进行拍照并将快照提交于布告栏。我们可以看到，每当场景中 299
任何对象发生改变时，我们的照相机都将自动产生一个更新的图像以代替旧的。为了产生动画效果，快速地显示多幅静止图像，就如同一个照相机一样。

继续我们的比喻，让我们来考虑合成的图像是如何产生的。首先，照相机操作者必须定位照相机并调整其方向。其次，必须决定哪些场景应该被显示：例如，是对象近景的特写还是远距离的全景。接下来，摄影者必须决定做一个多大的照片以便提交在布告栏中。最后，要决定将照片提交在布告栏的什么位置。在SPHIGS中，这些原则在视图中表示，包括观察操作的定义，该操作的视口定义了照片的大小及其在布告栏的位置。并不是结构数据库中的所有对象都用同样的“照相机设置”来显示。事实上，我们很快会看到，在一个布告栏中可以有多个视图。

2. 视口

如同上一章讨论的，视口在NPC系统中定义了一个平行六面体，在VRC中定义的视见体的内容被映射到这个区域内。由于NPC系统以一种固定的方式映射到物理设备的整数坐标系，因此视口也定义了图像将显示在屏幕的什么位置。三维的NPC系统以下面的方式映射到二维的屏幕坐标系：NPC单位立方体中位于(0, 0, 0)和(1, 1, 1)的两个对角点分别对应于屏幕上能表示的最大正方形顶点，而z坐标将直接被忽略。例如，一个显示设备的分辨率为水平1024竖直800，一个NPC下点$(0.5, 0.75, z)_{NPC}$对应设备坐标系下的$(512, 599)_{DC}$。为增强可移植性，应用程序不应使用位于单位立方体外的NPC系统空间；通常，为了充分利用非正方形屏幕的优势，为实现可移植性所牺牲的代价是值得的。

3. 视图表

SPHIGS维持一个视图表，其表项的数量与具体实现相关。每个视图包含一个称为视图表示的视见体和视口的定义，以及一个被提交的根列表（初始为空）。视图表中的第0项定义了一个默认视图，它是一个如图6-25b所示的视见体，前后平面分别位于$z = 0$和$z = -\infty$处。其默认视图的视口为NPC单位立方体。

视图表中所有项的视图表示（除视图0）都可以用以下过程来编辑：

```
void SPH_setViewRepresentation (
    int viewIndex, matrix_4x4 voMatrix, matrix_4x4 vmMatrix,
    double NPCviewport_minX, double NPCviewport_maxX,
    double NPCviewport_minY, double NPCviewport_maxY,
    double NPCviewport_minZ, double NPCviewport_maxZ);
```

两个4 × 4的齐次坐标矩阵是第6章所述的视图方向矩阵和视图映射矩阵，它们由图7-6所示的过程生成。 300

4. 多视图

在提交时定义的视图索引指向一个特定的NPC视口，该视口描述了在屏幕（布告栏）上出现结构图像（照片）的位置，如同人们可以在布告栏上提交多幅照片一样，一个应用程序可以将屏幕分成几个视口。

多视图的用途在许多方面都很强大。我们通过以不同的视图提交，可以在单个屏幕上不同的区域内同时显示几个不同的结构。在图7-7a中，我们示意性地表示一个视图表，只显示了指向每个视

图的被提交的结构网络列表的指针。我们可以看到有一个视图显示了一个街道的场景，有三个独立的视图显示了一个机器人。机器人的结构被显示了三次，每次有不同的视图索引。图7-7b显示了结果的屏幕图像。机器人的多个视图不仅可以具有不同的视口定义，也可以具有不同的视见体定义。

```
/* 建立UVN观察参考坐标系 */
matrix_4x4 SPH_evaluateViewOrientationMatrix (
        point viewRefPoint,
        vector_3D viewPlaneNormal, vector_3D viewUpVector,
        matrix_4x4 voMatrix);

/* 建立视体并描述它怎样映射到NPC空间 */
matrix_4x4 SPH_evaluateViewMappingMatrix (
        /* 首先，我们在VRC中定义视体 */
        double umin, double umax, double vmin, double vmax,     /* 视平面边界 */
        PARALLEL / PERSPECTIVE projectionType,
        point projectionReferencePoint;                          /* 在VRC中 */
        double frontPlaneDistance, double backPlaneDistance,     /* 裁剪平面 */
        /* 然后，我们指定NPC视口。 */
        double NPCvp_minX, double NPCvp_maxX,
        double NPCvp_minY, double NPCvp_maxY,
        double NPCvp_minZ, double NPCvp_maxZ,
        matrix_4x4 vmMatrix);
```

图7-6 用于计算观察变换矩阵的实用程序

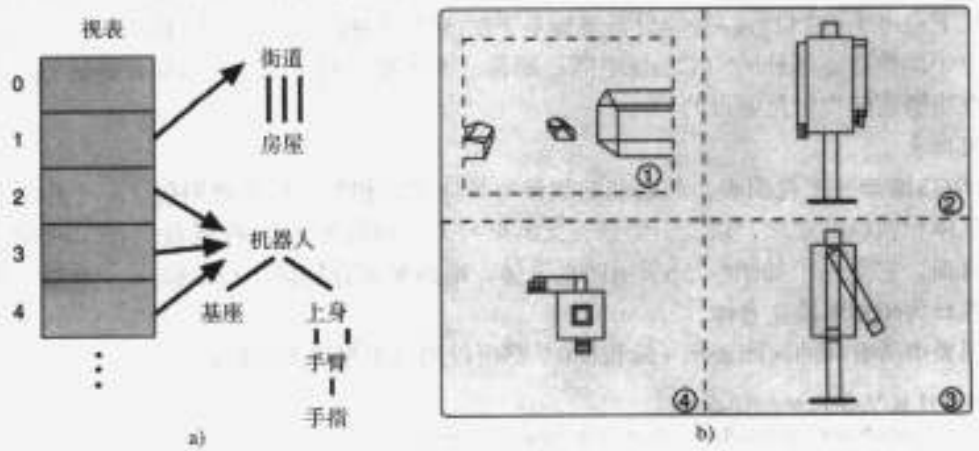


图7-7 共享同一屏幕空间的多视图。a)视图表的示意图，表中的每一项都指向一个提交给该视图的根列表；b)结果图像，虚线框定的视口范围和圆圈中的数字显示了视口和与之对应的视图索引

前述内容都表明每个视图至多有一个被提交的结构，但事实上，任意数目的根都能提交给单一的视图。因此，我们可以通过把所有根提交给一个视图来在一个统一的视图中显示不同的根结构。这种情况下，我们可比喻为用照相机拍摄一张包含许多物体的场景的合成快照。

视口不同于实际的快照或窗口管理视窗的另一个特性在于它是透明的[⊖]。在实际应用中，许多程序为避免重叠而将视口平铺开，但有时叠放也有其优势。如我们可以合成两个由不同的观察变换产生的或在不同测量单位下显示的图像，这样，我们就可以在建立一个机器零件的近视图时插入一个整个机器的小图（供参考）重叠在大的零件图上。（为避免混乱，我们可以选

⊖ 有些PHIGS的版本提供不透明的视口用于视口平铺，人们用屏蔽（shielding）一词来描述这种不规范的特点。

择近视图中只有背景的区域。）

为刷新屏幕，SPHIGS通过以视图索引值递增的顺序遍历在视图表中的每个视图所提交的根来显示所提交的元素网络，通常从视图表中的视图0开始。这样，一个提交给视图N的对象图像将比一个提交给索引值小于N的视图的对象图像有更高的显示优先级，因而可能遮盖住它们。当然，只有当视口实际上重叠时，它们间的顺序才是十分重要的。[⊖]

一个应用程序能产生许多独立的世界坐标（WC）系空间，并能使用任意所需的测量单位。例如，在图7-7中，街道的结构在世界坐标空中定义，此坐标系的轴向的一个单位代表10码，而机器人则定义在一个以厘米为度量的完全独立的世界坐标中。虽然每个根结构在其自己的世界坐标系中定义，但每一个显示设备只有一个NPC空间，由所有被提交的结构共享，因为这个坐标空间是显示设备的一个抽象。

7.3.5 通过窗口管理共享屏幕的图像应用

20世纪70年代早期，当设计第一个图形软件包时，在给定的时间内只能运行一个图形应用软件，并且使用整个屏幕。PHIGS的设计开始于20世纪70年代后期，当时这种独占屏幕的操作模式仍占主流，并且窗口管理程序还没有被广泛使用。因此，NPC空间的单位立方体就习惯地被映射到整个屏幕。

现代的拥有多任务操作系统的图形工作站允许同时运行多个图形应用程序，在窗口管理程序的控制下，共享工作站的资源、屏幕和输入设备。在这种环境中，每个应用程序被分配了一个自己的窗口，这个窗口就像一个“虚拟的屏幕”一样工作。用户通过调用窗口管理器的函数，能够改变窗口大小或移动窗口位置。其主要的优点在于，每个应用程序就像控制整个窗口一样工作，而不必关注其显示屏幕只是实际显示设备的一部分。因而在窗口管理器环境下，一个SPHIGS的应用程序不需要做其他的调整，软件包和窗口管理器合作将一个NPC空间映射到所分配的窗口而不是整个屏幕。[⊖]图7-8显示两个SPHIGS应用和一个终端仿真程序同时运行在一个图形工作站上。由于SPHIGS将NPC空间映射到适合窗口管理器所提供的窗口的最大正方形区域，非正方形窗口的一些部分对SPHIGS应用是不能用的，如图示的SPHIGS窗口所表现的桌椅的场景。

301 / 302

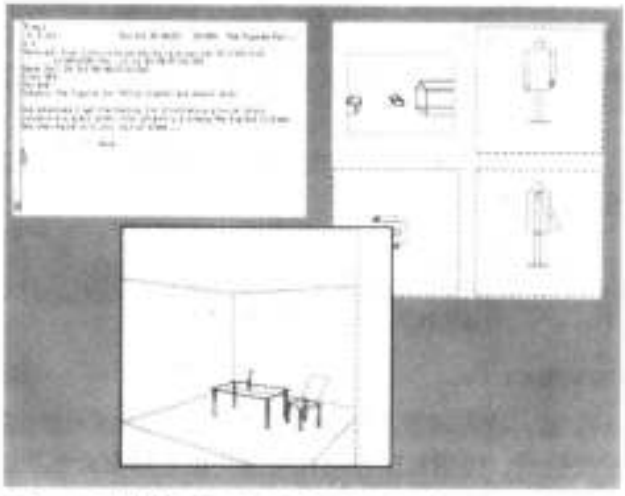


图7-8 在视窗管理器的视窗中运行的两个SPHIGS应用程序

303

⊖ 这种平凡的视图优先系统不如PHIGS复杂，允许应用程序显式地指定视图优先级。

⊖ 有时只需要在窗口管理器窗口中显示NPC空间的部分。PHIGS支持NPC工作站窗口的设置，用于裁剪显示遍历产生的图像，被裁剪的部分按外表比例映射到物体设备坐标中定义的工作站视口。这种工作站的变换也可用于将NPC空间的矩形部分映射到物理显示设备，并允许使用非正方形屏幕区域。

7.4 模型变换

7.3.2节包含了一段C程序，此程序创建一个简单的结构用于房屋的建模。出于简化考虑，我们把房屋的一个角放在坐标原点，房屋的边与主轴对齐，并设置整坐标单位的大小。我们把定义在坐标原点的并与主轴对齐的对象称为是标准化的。标准化的对象不仅较具有任意空间位置的对象易于定义（确定其顶点坐标），也易于进行几何操作，如改变大小、方向、位置等。

如果我们要将房屋放在不靠近原点的不同的位置，我们当然可以自己重新计算房屋的顶点，然后用7.3.2节所示的相同的Pascal代码重新创建一个房屋的结构，只是改变了顶点的坐标。但是，我们现在介绍一种强大的变换技巧实现对标准化部件对象的位置和大小的改变。

如同我们在第5章所看到的，我们可以通过将以列向量$[x, y, z, 1]^T$表示的每个顶点乘一个4×4的齐次坐标变换矩阵，来变换一个多边形这样的图元。以下的实现函数生成一个这样的矩阵：

```
matrix_4x4 SPH_scale (double scaleX, double scaleY, double scaleZ, matrix_4x4 result);
matrix_4x4 SPH_rotateX (double angle, matrix_4x4 result);
matrix_4x4 SPH_rotateY (double angle, matrix_4x4 result);
matrix_4x4 SPH_rotateZ (double angle, matrix_4x4 result);
matrix_4x4 SPH_translate (double deltaX, double deltaY, double deltaZ, matrix_4x4 result);
```

对不同的轴可以有不同的缩放因子，这样一个对象可以被非均匀地被拉伸和扭曲。对于旋转，用度表示的角度参数代表其绕设定的主轴按逆时针运动的角度，主轴的方向沿坐标轴的正无穷指向原点。

矩阵可以用于生成一个结构中的变换元素。下面是元素生成过程：

```
void SPH_setLocalTransformation (
    matrix_4x4 matrix, REPLACE / PRECONCATENATE / POSTCONCATENATE  mode);
```

“Local”这个前缀词表示SPHIGS如何显示一个结构。当SPHIGS遍历一个结构时，它存储一个*局部矩阵*，作为已经遍历过的结构的状态信息的一部分。局部矩阵通常都被默认地被初始化为单位矩阵。每当碰到一个setLocalTransformation元素，局部矩阵以某种方式修改：它可以被由mode参数定义的多重操作改变或者代替。在遍历过程中每碰到一个图元，其顶点都将由局部变换矩阵进行变换，然后再进行观察变换以便显示。（我们在后面将会看到，层次结构使这个过程变得更复杂。）

304 下面的代码创建一个在任意位置的房屋的结构，并利用默认视图提交该结构以用于显示。房屋维持它原来的标准化的大小和朝向。

```
SPH_openStructure (HOUSE_STRUCT);
    SPH_setLocalTransformation (SPH_translate (...), REPLACE);
    SPH_polyhedron (...);      /* 这里的顶点像前面一样被标准化 */
SPH_closeStructure ();
SPH_postRoot (HOUSE_STRUCT, 0);
```

像这样简单的变换并不常见。我们通常不仅希望平移对象，而且希望能影响它的大小和朝向。当要求对图元进行多重变换时，应用程序可以将每个变换矩阵依次相乘以得到一个局部的合成变换矩阵。通常，一个标准化的部件可以经过先缩放，再旋转，然后平移到达指定位置。就像在第5章中所述，这种顺序可以避免不需要的平移和剪切。

下面的代码创建并提交一个房屋的结构，它从原点移开并被旋转到我们可以看到其侧面而非其前面的位置：

```
SPH_openStructure (MOVED_HOUSE_STRUCT);
```

```
    SPH_setLocalTransformation (SPH_rotateY (...), REPLACE);
    SPH_setLocalTransformation (SPH_translate (...), PRECONCATENATE);
    SPH_polyhedron (...);     /* 这里的顶点像前面一样被标准化 */
SPH_closeStructure ();
SPH_postRoot (MOVED_HOUSE_STRUCT, 0);
```

对平移矩阵使用PRECONCATENATE模式保证左乘用于合成平移矩阵和旋转矩阵，从而先转动再平移。左乘要远比右乘普遍，因为它和单个变换元素的顺序相对应。因为SPHIGS相对于主轴进行旋转和缩放，所以编程人员必须生成一个将任意轴映射到主轴所需的变换矩阵，正如第5章所讨论的。

SPHIGS在遍历时间完成变换元素的合成，这样，每次屏幕刷新时，都执行变换单元合成的过程。另一种通过定义连续的变换序列的方法则提高了显示遍历过程的效率，即我们自己在一定义时间（定义阶段）合成这些变换，并生成一个变换元素。下面的函数在定义时间内执行矩阵乘法：

```
matrix_4x4 SPH_composeMatrix (matrix_4x4 mat1, matrix_4x4 mat2, matrix_4x4 result);
```

在前面代码中的两个setLocalTransformation可以用以下代码代替：

```
SPH_setLocalTransformation (
    SPH_composeMatrix (SPH_translate (...), SPH_rotateY (...), result), REPLACE);
```

305

这种方法的缺点在于不能对图元的大小和方位进行动态的改变，因为这种改变需要对一系列setLocalTransformation元素的成员做选择性的编辑；另外，整个合成矩阵必须重新计算和指定。提高效率的经验规则是：除非其中的某个变换发生了选择性的更新而不得不单独指定外，在其余的情况下，我们就在定义时使用合成。

让我们来看一下由三个简单房子（如图7-7所示）组成的街道结构。在如图7-9a所示的透视图中，左边是一所民房，右边是一幢大厦，中间是一个小村舍。我们画出平行于x坐标轴的虚线并在x轴上做了标记，指出这些房子的相对位置，并且我们使用SPHIGS的显示模式表示出多面体线框边并进行了消隐（见7.8节）。最左边的房子是没有经过任何变换的标准化的房，而另外两个房子则是大小、方向和位置不同的副本。

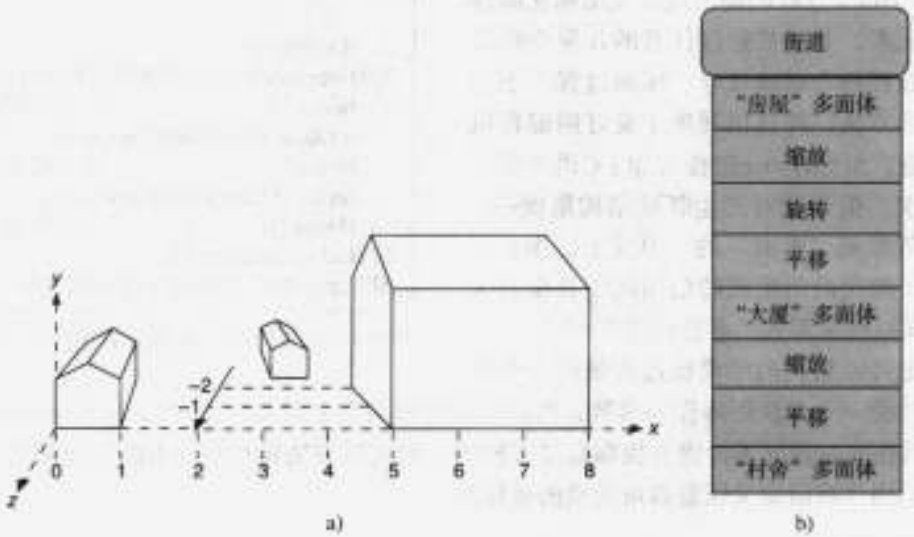


图7-9 构建一个具有三所房屋的街道模型。a) 透视图，b) 结构

如图7-9b的示意图所示，生成这个街道结构的直接的方法就是先定义三个标准化房子多面体，然后进行适当的变换。我们把顺序进行的一组变换元素看成一个单元；其中的第一个元素采用了REPLACE模式，其余均采用PRECONCATENATION方式，相邻的变换之间用乘号（·）隔开。生成整个街道结构的代码如图7-10所示。

```
SPH_openStructure (STREET_STRUCT);
    SPH_polyhedron (...);      /* 使用其标准方式定义第一所房屋 */

    /* 大厦就是将房屋在x, y, z方向上分别放大2倍、3倍和1倍，绕y轴旋转90°  */
    /* 然后平移；请注意它的左面现在是朝前的，  */
    /* 并处于(x, y)平面中 */
    SPH_setLocalTransformation (SPH_scale (2.0, 3.0, 1.0, result), REPLACE);
    SPH_setLocalTransformation (SPH_rotateY (90.0, result), PRECONCATENATE);
    SPH_setLocalTransformation (SPH_translate (8.0, 0.0, 0.0, result), PRECONCATENATE);
    SPH_polyhedron (...);

    /* 村舍就是将房屋均匀缩放0.75倍，不旋转，设置在z轴的负方向、x轴的正方向 */
    SPH_setLocalTransformation (SPH_scale (0.75, 0.75, 0.75, result), REPLACE);
    SPH_setLocalTransformation (SPH_translate (3.5, 0.0, −2.5, result), PRECONCATENATE);
    SPH_polyhedron (...);
SPH_closeStructure ();
SPH_postRoot (STREET_STRUCT, 0);
```

图7-10 生成图7-9的代码

如图7-11所示，我们可以定义了一个Pascal过程来生成标准化房子多面体，以消除该标准房子多面体定义中的冗余。因为本例中的房子是由一个简单的多面体调用定义的，这种方法的效果并不明显；如果房子的构造更复杂一点，包含多个图元及属性的定义，这种方法就能明显地减少代码量。除此之外，该方法还有利于程序的模块化：当房子的形状或式样改变时，只需编辑House过程，而不必编辑构造街道结构的代码。

```
void House (void)
{
    SPH_polyhedron (...);
}

/* Mainline */
SPH_openStructure (STREET_STRUCT);
    House ();                  /* 第一所房子 */
    set local transformation matrix;
    House ();                  /* 大厦 */
    set local transformation matrix;
    House ();                  /* 村舍 */
SPH_closeStructure ();
SPH_postRoot (STREET_STRUCT, 0);
```

图7-11 使用一个模板程序对街道进行建模

我们把House这样的能生成定义结构化部件的一系列元素，并能被经过任意的几何变换反复调用的过程称为模板过程。模板过程对于程序员来说很方便，并且也展现了良好的编程风格。请注意，虽然House过程增加了C语言程序的过程层次，但是没有产生新的结构层次——街道模型依然是“平面”的。事实上，图7-11与图7-10中的代码所生成的结构网络是没有区别的。生成结构所需的元素数也基本相同。

306
~
307

我们也可以对我们的模板过程做的一个改变是让它接受一个变换矩阵作为参数，然后模板过程用它来定义一个setLocalTransformation元素。[⊖] 虽然在某些情况下传递变换参数是方便的，但这种方法相对于原来的方法而言，丧失了其可以在调用模板前定义任意数量变换的通用性。

⊖ Newman定义显示过程为模板过程，以缩放、旋转和平移为参数。

7.5 层次式结构网络

7.5.1 两层层次结构

到目前为止，我们已经讨论了三种类型的结构元素：输出图元、外观属性和几何变换。下面，我们将介绍SPHIGS系统是如何从结构层次中获得强大功能，以及这种分层结构是如何实现的。这种结构层次是通过使一个元素在运行时遍历调用子结构而实现的，不要把它同前面介绍的模板过程层次结构相混淆。模板过程的结构层次是在对CSS进行编辑时（定义时间）形成的，它产生的是嵌入式元素，而非子结构调用。与此相反，子结构调用引起的结构层次是在显示时对CSS进行遍历而形成的。在7.15节，我们会对二者的联系和各自的优缺点进行分析。

调用子结构的结构执行元素以如下方式创建：

void SPH_executeStructure (**int** *structureID*);

让我们用一个在CSS中生成一所房屋的过程来替换前面的模板过程（如图7-12所示）。该过程在主函数中只调用了一次，并且HOUSE_STRUCT从未提交过；它是作为街道结构的一个子对象被调用并显示出来的。注意，这里STREET_STRUCT定义的惟一区别就在于增加了对建造房屋结构的过程的调用，以及把每一个对模板过程的调用都替换为对executeStructure的调用。虽然显示图像跟图7-9a是一模一样的，但结构网络不一样，如图7-13所示，图中每个executeStructure元素都引出了一个箭头。

```
void BuildStandardizedHouse (void)
{
    SPH_openStructure (HOUSE_STRUCT);
        SPH_polyhedron (...);
    SPH_closeStructure ();
}

/* Mainline */
BuildStandardizedHouse ();
SPH_openStructure (STREET_STRUCT);
    SPH_executeStructure (HOUSE_STRUCT);     /* 第一所房子 */
    设置局部变换矩阵
    SPH_executeStructure (HOUSE_STRUCT);     /* 大厦 */
    设置局部变换矩阵
    SPH_executeStructure (HOUSE_STRUCT);     /* 村舍 */
SPH_closeStructure ();
SPH_postRoot (STREET_STRUCT, 0);
```

图7-12 利用一个从属结构对街道进行建模

提交STREET_STRUCT使得SPHIGS系统通过遍历STREET_STRUCT的结构网络来更新屏幕；遍历采用了深度优先的方式，正如过程/子程序层次结构被执行一样。在前述的例子中，遍历过程先把街道结构的局部矩阵规格化为单位矩阵，接着对房屋子结构做第一次调用，调用时将街道结构的局部矩阵应用到房子的每个顶点，就好像构成房屋的多面体本身是街道结构的图元一样。第一次调用返回后，局部矩阵被置为想要得到的变换的合成矩阵，然后做第二次调用，并将这个新的合成矩阵应用到房屋

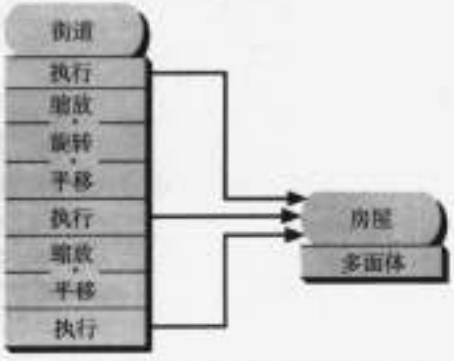


图7-13 表明从属结构的调用的结构网络

顶点，以生成房子的第二个实例。第二次调用返回后，局部矩阵被再次改变；新的合成矩阵再次被应用于房子的顶点以得到第三个房子实例。

我们把一个结构看成是一个独立的实体，其中的图元都定义在它自己的浮点造型坐标系统（MCS）中；这样考虑是为了便于建立低层的标准化的部件。正如我们在5.8节所指出的那样，变换将某个坐标系统下的顶点映射到另一个坐标系统中；这里，SPHIGS采用结构*S*的局部矩阵把子结构中的图元变换到*S*本身的MCS中。

308 ↓ 309

7.5.2 简单的三层层次结构

作为一个三层层次结构的简单例子，我们对街道例子中的房屋进行了扩展。新的房子由原来的标准化的房子（改称为SIMPLE_HOUSE_STRUCT）和一个烟囱组成，烟囱经过了适当的缩放和平移并直立在房子的右上角。有两种方法可以修改房子的结构，一是把组成烟囱的面直接加在原来的多面体上，二是再在结构中增加一个多面体。我们的选择是把房子分解为两个子体从而引入三层层次结构。这种模块化的好处在于我们可以在烟囱本身的MCS中以标准化的方式（位于原点，单位尺寸）来定义烟囱（如图7-14a所示），然后再经过缩放和平移将其放到房子的MCS中并位于屋顶。要是我们把烟囱直接定义到屋顶并映射到房子的MCS而不进行缩放，很明显我们就不得不对其顶点进行繁杂的计算。然而采用模块化处理，我们只需定义一个标准化的烟囱并使它底面的坡度与屋顶相同；只要满足这个条件，就能进行均匀的缩放和任意的平移。

修改后的房屋结构由以下代码生成：

```
SPH_openStructure (HOUSE_STRUCT);
    SPH_executeStructure (SIMPLE_HOUSE_STRUCT);
    set local matrix to scale/translate standardized chimney onto roof of simple house;
    SPH_executeStructure (CHIMNEY_STRUCT);
SPH_closeStructure ();
```

当我们用街道结构以变换来实例化这个两层的房子生成如图7-14b所示的三层层次结构会

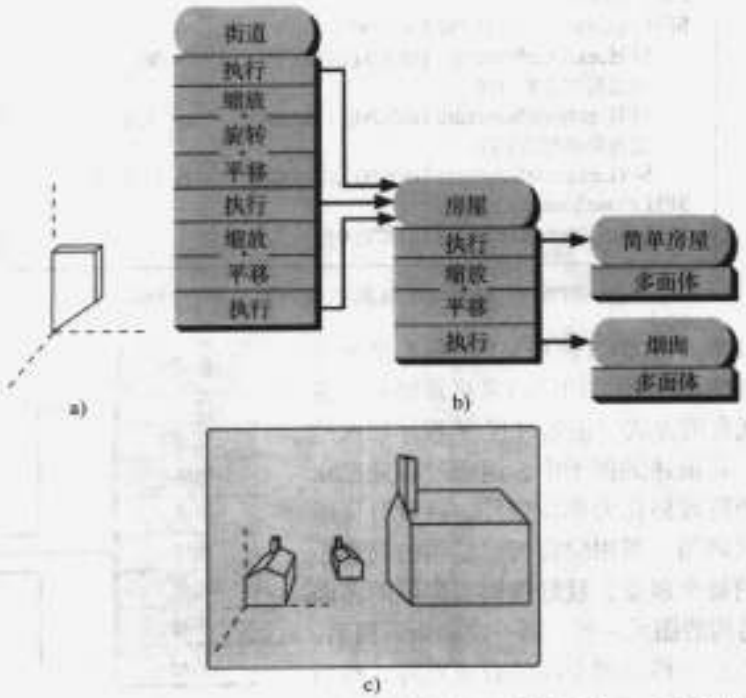


图7-14 街道的三层层次结构。a) 标准化的烟囱，b) 结构网络，c) 结果图像

怎样呢？由于SPHIGS是通过变换其组成元素和子结构来变换一个父结构的，因此我们可以确保这两个图元构件(简单房子和烟囱)是作为一个整体而得到变换的（如图7-14c所示）。这里的关键之处在于街道结构的定义完全不用改变。因此，街道结构的设计者不必关心内部的细节，比如房子是怎样构成的等等，对于设计者来说，这是一个黑匣。

7.5.3 自底向上构造的机器人

作为一个更加有趣的典型例子，我们来看一下简单机器人，它采用结构层次造型，并通过不断修改变换以达到动态效果。对于复杂的物体或系统层次结构，通常采用自顶向下的方式来进行构思和描述。比如，一幢计算机系的大楼由各个楼层组成，每个楼层都包括一些办公室和实验室，而每个办公室或实验室都有大量的器具和设备，依此类推。让我们回想一下，这个机器人由上身和下部底座组成；上身包括躯干、头和两支一样的手臂，其中每支手臂又由一个固定的手和一个可以滑动（平移）的手指（作为钳子）组成。

虽然我们设计时采用了自顶向下的方式，但实现时却采用了自底向上的方式，定义一些构造块用于逐层生成高层的构造块的定义，从而形成部件的层次结构。因此，在构造机器人时，我们就定义手指构造块来定义手臂，然后把两个手臂部件的实例拼在上身上，依此类推，部件层次的示意图如图7-2所示，上身的详细结构网络图如图7-15所示。

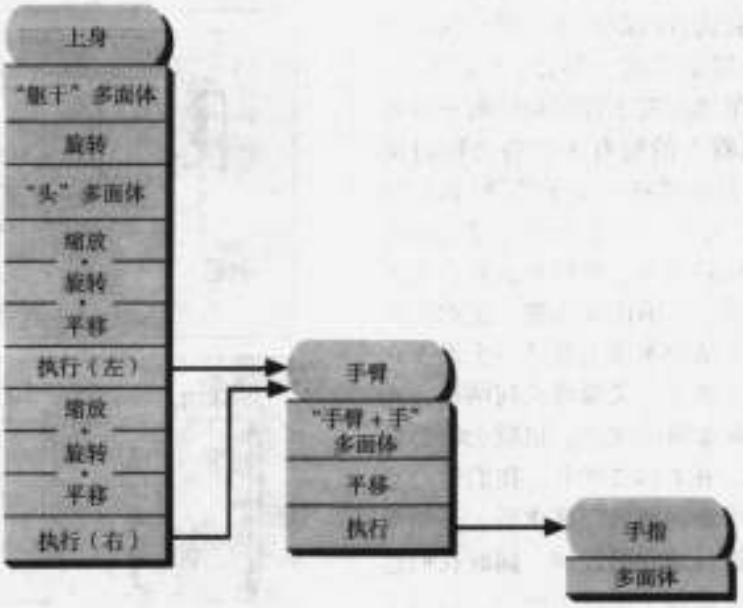


图7-15 机器人上身的结构层次

让我们来仔细分析一下这个自底向上的构造过程所包括的几何结构和变换。用相同的计量单位来定义手臂和手指使得二者之间更容易组合。我们把手指结构定义在它自身的MCS中的标准化位置，在MCS中它沿y轴"悬垂"（如图7-16a所示）。定义手臂结构时采用了与手指相同的计量单位，它由一个手臂+手的多面体（标准化的，沿y轴悬垂，如图7-16b所示）和一个经过平移的手指结构的调用组成。手指调用之前的平移元素负责将手指从标准化的原点位置平移到手腕上的适当位置（图7-16c）。

手臂调用手指构成手腕的过程是一个两层结构，与房屋-街道的例子很相似。通过编辑手

臂结构的平移元素，我们可以沿着手腕滑动手指（如图7-16d）[⊖]。

接下来，我们构造上身。由于我们想要头部可以转动，故而先定义躯干多面体，然后旋转，再定义头部多面体（图7-16e）。下一步是让这个上身结构调用手臂结构两次。在调用前应当做什么样的变换呢？如果我们主要关心的是使手臂准确定位于上身的MCS中，我们将得到如图7-16f所示的结果：手臂和上身的设计明显不合比例。这个问题很容易解决：我们可以在此之前先进行一个缩放变换（如图7-16g所示）。但是，如果我们要使手臂能够绕连接双肩的轴线转动（就像是行进中的士兵的动作一样），光靠一个缩放变换和一个平移变换是不够的；为达此目的，平移前，我们在结构中增加了一个旋转元素。到此为止，我们完成了对整个上身结构的定义；习题7.1就是装配整个机器人。图7-16h展示了左臂旋转元素不为0时的情况。

因为每支手臂在调用前都进行了一系列独立变换，因此每支手臂的动作都能得到独立的控制。当一支手臂悬垂时，另一支手臂可以摆动，如图7-16h所示。事实上，正是变换的不同导致了左、右臂的差异。然而，请注意，对于左臂和右臂来说，可滑动的手指不仅位于固定手的同一侧，而且离开的距离也一样，这是因为手指是手臂内部定义的一部分。（实际上，如果程序只是简单地改变手臂结构中的平移元素，那么所有机器人的所有手指将会同时动作！）因此，我们必须将一支手臂绕y轴旋转180°，使两手臂对称。一个调用手臂的结构可以控制整个手臂的尺寸、方向和位置，却无论如何也无法改变手臂的内部构造。正如我们前面所说，一个子结构本质上就是一个黑匣：调用者需要知道子结构定义哪些几何结构，却不必知道子结构是如何生成的，也就无法影响到子结构的内部。在7.14.2节中，我们讨论了为什么当一个手臂结构被调用两次后，不能对其两个手指分别进行独立的控制，同时我们也展示了如何通过生成多个形式上统一、内部变换各异的手臂结构来解决这个问题。

总而言之，在定义任何结构时，我们只需处理如下问题：该结构所包括的图元和低层子结构，定位该结构的组成部件时所需用到的造型变换，以及影响子结构外观所需用到的属性。我们不必也不能改变低层子结构的内部构造。

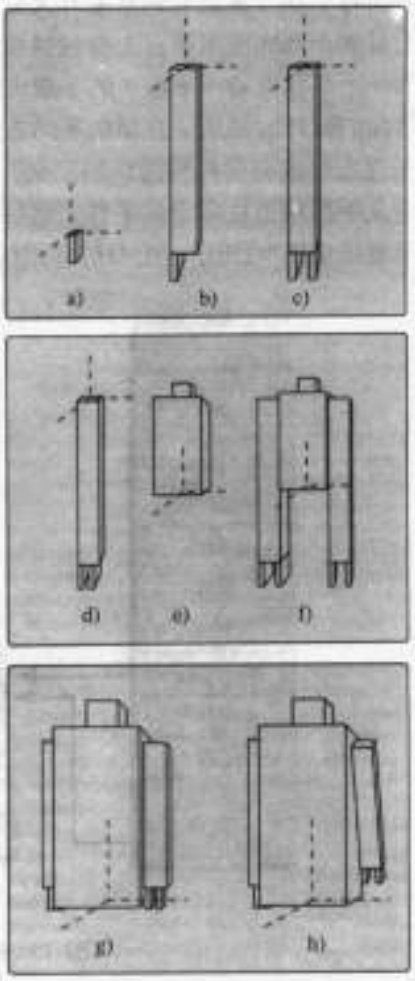


图7-16 构造一个机器人的上身。a)手指，b)固定手臂＋手，c)完整的手臂，d)具有平移的手指的完整的手臂，e)躯体和头，f)具有不合尺寸手臂的上身，g)正确的手臂；h)左手臂旋转

⊖ 谨慎的读者可能会产生疑问：在此模型中，既然它并未真正连在机械手臂上，手指实际上是如何滑动的。事实上，我们的机器人模型的任何部分都未通过所谓的“关节”与其他部分连接。关节的建模和确定它们的操作模式的约束不在此书的讨论范围。

此外，我们在设计构件时不必关心它们将被怎样调用，因为可以用一系列的变换来得到预期的大小、方向和位置。在实际应用当中，使构件在其自身的MCS中标准化是很有用的，这样它可以很方便进行缩放和绕主轴的旋转变换。

还有两点需要补充说明一下。首先，一个程序员不必用单纯的自顶向下的方法来设计，或是用单纯的自底向上的方法来实现；与编程技术中使用的“桩模块和驱动模块”类似，可以在结构层次中使用虚子结构。一个虚子结构可以是一个空的结构；事实上,在SPHIGS中允许一个结构调用一个还未生成的子结构，在这种情况下，SPHIGS会自动生成这个子结构并将它初始化为空。在某些情况下，需要使虚子结构是一个与复杂实体大小近似的简单实体(比如，一个平行六面体)，复杂实体要稍后才能确定。这种方法使得在所有部件都得以确定前，对复杂的结构层次进行自顶向下的调试成为可能。其次，我们还没有对结构层次和模板过程层次做出比较。到目前为止，我们可以说，当我们需要使用一个部件的多个拷贝，并对其中每个拷贝的外观和布局（而不是它们的内部）进行控制（在一定程度上）时，采用层次结构还是很有效的。我们将在7.15节中对模板过程层次和结构层次之间的权衡与折衷给予讨论。

310 ~ 314

7.5.4 交互式造型程序

交互式三维造型程序通过采用前面所述的自底向上的装配过程，使得物体的层次构造更为容易。大部分的这种程序都提供了一个由许多图标组成的面板，这些图标就代表了程序中的基本部件集。如果制图程序是专用的，那么部件集也是专用的；否则，它们就是诸如折线、多边形、立方体、平行六面体、圆柱体、球体之类的通用元素。用户可以选择一个图标以得到对应部件的一个实例，然后对该实例定义一些变换。这些定义是通过输入设备来做的，如鼠标、控制盘等等，并能让用户对实例的大小、方向和位置进行实验，直到满意为止。由于要对三维景物的二维投影中的空间关系进行判断是很难的，更为完善的交互技术是应用三维网格，通常还带有围绕交点的重力场、数值比例、各式各样的滑块以及关于顶点位置的数值反馈信息等等（见8.2.6节）。一些构造程序还允许用户通过组合基本图形部件生成高层部件，并加入部件面板中，以用来组建更高层的对象。

7.6 显示遍历中的矩阵合成

到目前为止，我们已经介绍了程序员是怎样通过自顶向下的设计和自底向上的实现来构造一个模型的。不管模型是怎样构造的，SPHIGS对DAG可采用自顶向下、深度优先搜索把模型显示出来该DAG的根是已经提交的结构。在遍历的过程中，SPHIGS对各层变换和调用中指定的所有几何结构都要进行处理。让我们来看一个自顶向下遍历的例子：根部结构A调用结构B，B又调用结构C。这就是说，C中的图元从C自身的MCS变换到B的MCS，以完成对B的构造；同样，通过把B中的图元（包括调用C得到的图元）变换到A的MCS中，又完成了对A的构造。结果是C中的图元变换了两次，先是从MCS_C到MCS_B，再从MCS_B到MCS_A。

采用5.8节中介绍的表示法，以$M_{B\leftarrow C}$表示结构B时的局部矩阵，在调用C时将MCS_C中的顶点映射到MCS_B中的相应的变换位置。因此，我们把一个顶点从MCS_C映射到MCS_B记为$V^{(B)} = M_{B\leftarrow C} \cdot V^{(C)}$（这里$V^{(H)}$指代表一个其坐标在坐标系$H$中表示的顶点的向量）；类似地，有$V^{(A)} = M_{A\leftarrow B} \cdot V^{(B)}$。因此，为了模拟自底向上的构造过程，遍历程序必须连续应用一系列的变换，以把顶点从C映射到B，再从B映射到A：

$$V^{(A)} = M_{A\leftarrow B} \cdot V^{(B)} = M_{A\leftarrow B} \cdot (M_{B\leftarrow C} \cdot V^{(C)}) \tag{7-1}$$

根据矩阵的结合律，$V^{(A)} = (M_{A\leftarrow B} \cdot M_{B\leftarrow C}) \cdot V^{(C)}$。因此，遍历程序只需把两个局部矩阵组合起

315 来，并将结果矩阵应用于C中的每个顶点。

采用树形表示法，树根位于第1层，后续的子树分别位于第2, 3, 4, …层。根据归纳法可知，对于位于第 j 层(j>4)的任意结构，我们可以通过下式，把处于该结构本身的MCS中的顶点$V^{(j)}$变换为处于根部坐标系统中的顶点$V^{(1)}$：

$$V^{(1)} = (M_{1\leftarrow 2} \cdot M_{2\leftarrow 3} \cdot \cdots M_{(j-1)\leftarrow j}) \cdot V^{(j)} \tag{7-2}$$

由于SPHIGS允许图元在自己的局部MCS中通过局部矩阵做变换，将局部矩阵作用于图元的坐标值就得到了顶点$V^{(j)}$：

$$V^{(j)} = M^{(j)} \cdot V^{(\text{prim})} \tag{7-3}$$

我们用$M^{(j)}$来表示这个局部矩阵，当遍历到这个结构时，$M^{(j)}$用来对该结构的图元变换到它自己的第j层MCS中。若这个结构还调用从属子结构，局部矩阵的作用又有了变化；它将用来把被调用的子结构从第$j+1$层的MCS变换到第j层中去，我们用$M_{j\leftarrow(j+1)}$来表示。这并不意味着矩阵的值发生了变化，而是它的作用发生了变化。

使用结合律合并式(7-2)和式(7-3)，得

$$V^{(1)} = (M_{1\leftarrow 2} \cdot M_{2\leftarrow 3} \cdot \cdots \cdot M_{(j-1)\leftarrow j} \cdot M^{(j)}) \cdot V^{(\text{prim})} \tag{7-4}$$

因此，要把第j层的图元变换到根部的MCS(即世界坐标空间)中去，我们所要做的是要自顶向下，把从根部到图元本身所在的结构之间的所有结构的局部矩阵组合起来。这些局部矩阵的合成（见式(7-4)括号中的项)称为*合成造型变换矩阵*（CMTM）。当遍历到第j层结构中的图元时，CMTM是j个矩阵的合成。只有最后一个矩阵（$M^{(j)}$）才能被结构改变，因为一个结构只能修改它本身的局部矩阵。[⊖] 因此，当一个结构被激活时，CMTM中的前 $j-1$个矩阵是固定的。当遍历到第j层结构时，前面$j-1$个矩阵的合成称为一个*全局矩阵*（GM）（见式(7-2)括号中的项）。在遍历一个结构期间，保留这个GM对SPHIGS来说是很方便的；当一个setLocalTransformation元素改变了局部矩阵（LM）时，SPHIGS通过把这个新的局部矩阵右连接到GM中，就可以很容易地计算出新的CMTM。

现在我们来总结一下遍历算法，稍后在7.12.1节中将会进行详细描述。SPHIGS做一个深度优先遍历，调用任何结构前先保存CMTM、GM和LM；然后将子结构的GM和CMTM初始化为继承得到的CMTM，LM初始化为单位矩阵。CMTM用于做顶点变换，并随LM的改变而改变。最后，当遍历过程返回时，恢复父结构的CMTM、GM和LM，继续执行。由于对矩阵做了保
316 存和恢复，父结构对子结构会产生影响，反之则不会。

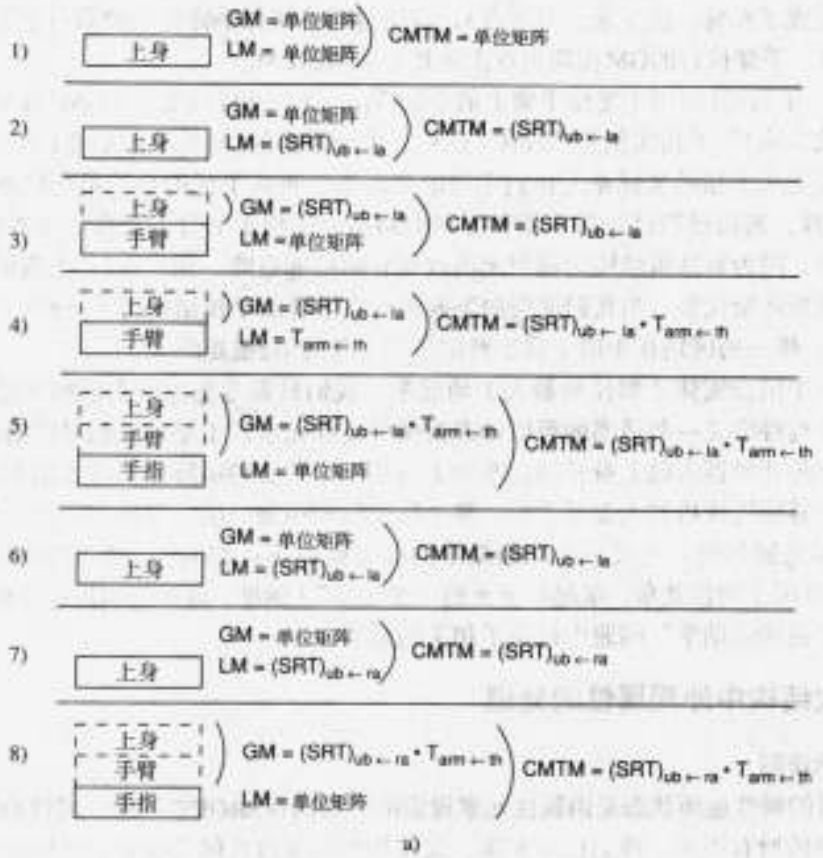
让我们看一下SPHIGS遍历如图7-15所示的上身–手臂–手指三层层次结构时的情况。我们提交UPPER_BODY结构作为根。图7-17a展示了遍历状态的快照序列；每个快照与图7-17b的结构网络图中标有相应序号的点相对应。

遍历状态通过如图7-17a所示的堆栈维护，这个堆栈是往下增长的，当前的活动结构位于实线矩形框中，其祖先位于虚线框中。与当前活动结构相对应的三个状态矩阵示于堆栈的右侧。弧线描出了变换的作用域：GM所对应的弧线包括了对GM有贡献的祖先结构，CMTM对应的弧线则指明它是GM与LM的乘积。再提醒一下，对于每一组变换，第一个变换矩阵采用REPLACE模式，而其余的都采用PRECONCATENATED模式。这样就保证了结构中的第一个旋转变换只作用于头部，而不会影响其他（如左臂），因为这个旋转变换矩阵将被作用于左臂

⊖ 我们给出这个全局矩阵作为一个不能使结构改变的导出实体。在真正的PHIGS中，结构在执行时可以改变GM，但其能力在某种意义下有某种局部性，因为它无法影响其祖先的局部矩阵。

的第一个缩放变换矩阵所替代。

在图7-17b中标出的点1，表示遍历过程刚执行到根部的第一个元素。由于根节点没有祖先，



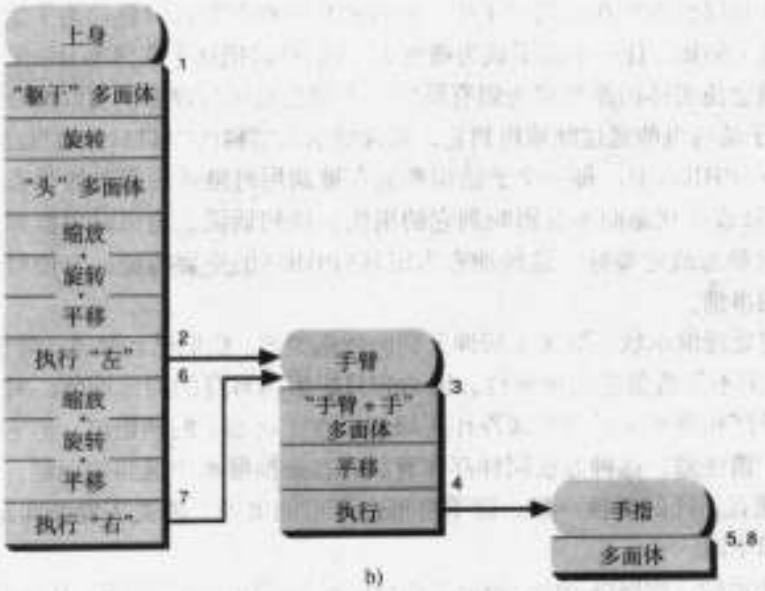


图7-17 三层层次结构的遍历。a) 遍历状态栈一个时刻的状态，b) 带注释的结构网络

它的GM是单位矩阵；LM也是单位矩阵，对于任何结构刚开始执行时均是如此。对于点2，LM被置为三个变换（缩放、旋转、平移，简写为SRT）的合成。因此，CMTM也相应的变为单位矩阵GM与合成矩阵SRT的乘积，以用来将手臂子结构变换到上身的MCS中去，于是通过$(SRT)_{ub\leftarrow la}$生成了左臂。接下来，对于点3，遍历过程将执行手臂结构的第一个元素。在期望第2层实例化时，手臂执行的GM在调用点正是其父结构的LM。

在点4，手臂的LM用于定位手臂上的手指($T_{arm\leftarrow th}$)，CMTM更新为GM与LM的乘积。第2层的CMTM成为第3层手指实例化的GM（点5）。由于手指的LM是单位矩阵，于是其CMTM变换的结果就是先从手指的坐标系变换到手臂的坐标系，再从手臂的坐标系变换到上身的坐标系。在点6的位置，遍历过程已经从手指和手臂的调用返回到了上身。上身结构的矩阵又变回到跟调用前一样，因为对从属结构的调用无法改变它的局部矩阵。到了点7，上身的LM又被新的右手臂的合成矩阵所代替。当我们逐层向下遍历到右手臂的手指结构时（点8），CMTM几乎跟点5处的一样；惟一的区别在于用于将手臂定位到上身的第2层矩阵。

要使一个组合实体（如使机器人）动起来，我们只需考虑怎样去影响父结构中的子结构，并针对每个构件定义一些适当的可以动态改变的变换元素。于是，我们可以通过改变整体结构中的旋转矩阵让机器人的上身转动，改变上身结构中的旋转矩阵让机器人抬起手臂，改变手臂结构中的平移矩阵使机器人张开手指。整个层次结构中每一层的变换都是独立进行的，但产生的净效果却是累积的。定义一个动画最难的部分就在于：如何从一个想要得到的结果（比如“机器人走到屋子的西北角，拿起桌子上的一个木块”）倒推，以得到满足这个要求的变换序列。第21章在“逆向运动学”问题中讨论了相关的细节。

7.7 层次结构中外观属性的处理

7.7.1 继承法则

遍历时的属性遍历状态是由属性元素设定的，如同在SRGP中一样，属性遍历状态被应用于遍历时遇到的所有图元。我们已经知道，父结构如何通过几何变换对子结构施加影响。什么样的规则适用于外观属性呢？在街道例子中，所有的房屋都有默认颜色。为了给某一个实体结构一种特殊的颜色（例如，让一个房子成为褐色），我们可以把这个实体本身一开始就定义成该颜色，但是这样做会使实体的颜色成为固有属性，不能在遍历时改变。更好的办法是“把颜色作为参数传递”，子结构也能通过继承得到它，就像继承父结构的CMTM作为自己的GM一样。

事实上，在SPHIGS中，每一个子结构都能在被调用时继承遍历时的状态（只要该状态存在），并能随意修改该状态而不会影响到它的祖先。换句话说，遍历时属性和变换是动态绑定的，这比定义时静态给定要好。这种动态绑定是SPHIGS的重要特征，它使得自定义子结构实例成为很容易的事情。

子结构如何处理继承状态取决于所涉及到的数据类型。我们已经知道，对于几何变换来说，子结构继承GM但不能覆盖它的继承性，因为它只能影响自身的局部矩阵。对于属性来说，情形更为简单：子结构继承父结构的属性作为局部属性（状态）的初始值，但它能随后对其局部状态进行修改。请注意，这种方法同样存在着我们在变换继承中遇到的问题；正如机器人两支手臂的手指不能有不同的变换一样，以下情形也不可能出现：两支手臂的固定部分颜色相同，而手指的颜色却不同。

317 ~ 319

在图7-18a所示的结构网络图中，街道结构对房屋子结构设置了颜色。其结果图像见图7-18b，相应的代码在图7-19中列出。

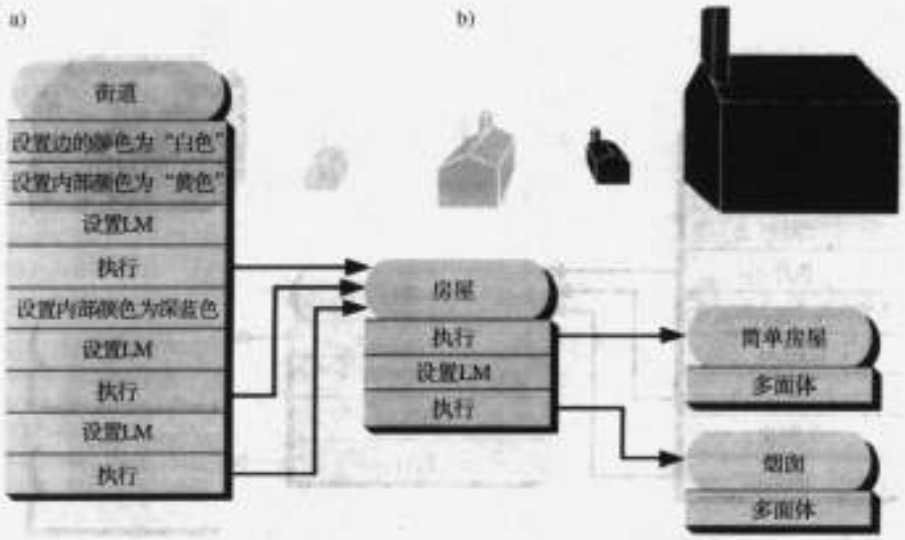


图7-18 在用彩色房屋构建街道模型中使用属性继承。a)结构网络；b)结果图像。(内部颜色由图案模拟。)

子结构中属性可以重置以覆盖继承所得的值。下面的代码段定义了一个修改后的房屋结构，它的烟囱总是红色的。

```
SPH_openStructure (HOUSE_STRUCT);
    SPH_executeStructure (SIMPLE_HOUSE_STRUCT);
    SPH_setInteriorColor (COLOR_RED);
    set up transformation;
    SPH_executeStructure (CHIMNEY_STRUCT);
SPH_closeStructure ();
```

```
SPH_openStructure (STREET_STRUCT);
    SPH_setEdgeColor (COLOR_WHITE);

    SPH_setInteriorColor (COLOR_YELLOW);
    起动变换;
    SPH_executeStructure (HOUSE_STRUCT);

    SPH_setInteriorColor (COLOR_NAVY);
    起动变换;
    SPH_executeStructure (HOUSE_STRUCT);

    起动变换;
    SPH_executeStructure (HOUSE_STRUCT);
SPH_closeStructure ();
```

图7-19 生成图7-18的代码

让我们用这个新的房屋结构与图7-19中的代码生成的街道结构连接起来。图7-20显示了结构网络以及结果图像。遍历过程从STREET_STRUCT开始；棱边的颜色属性和内部的颜色属性都被设为默认值。棱边设置为白色，这个值在遍历过程中自始至终保持不变。第一个setInteriorColor函数将HOUSE_STRUCT的第一个实例置为黄色，这样该实例将黄色传递给SIMPLE_HOUSE_STRUCT，SIMPLE_HOUSE_STRUCT的多面体显示为黄色。当遍历过程从SIMPLE_HOUSE_STRUCT返回到HOUSE_STRUCT时，内部颜色属性立即被下一个元素变为红色。于是第一个房子的烟囱显示为红面白边。当然，这些操作都不会影响到STREET_STRUCT的属性组；当遍历过程从HOUSE_STRUCT返回时，STREET_STRUCT的内部颜色属性又恢复为黄色。接着，该内部颜色属性又设为深蓝色，从而为画后两个蓝色房子做好准备。

7.7.2 SPHIGS的属性及文字不受变换影响

在PHIGS的真正实现中，文字会像其他的图元那样随着变换而变化。因此，对于一辆卡车侧面上的文字，它的透视效果会随着透视缩小效应旋转或显示，就好像字母是由一系列单独的折线和填

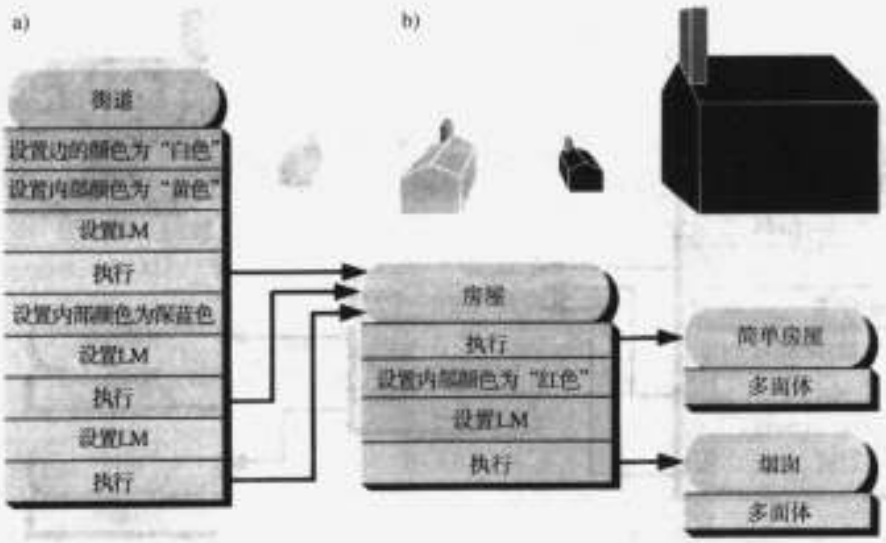


图7-20 改变继承属性的从属结构。a)结构网络，b)结果视图

充区组成的。同样，虚线中的短划线也要受几何变换和透视缩小效应的影响。然而，由于性能上的考虑，SPHIGS中的属性（包括文字）是非几何的。正如在SRGP中，屏幕上的文字尺寸只取决于字体，甚至连字符串都无法旋转——屏幕上的字符串永远都是竖直的。因此，旋转和缩放变换只影响文字显示的起点，不能影响它的尺寸和方向（图7-21）。SPHIGS中的文字图元主要用于标注。

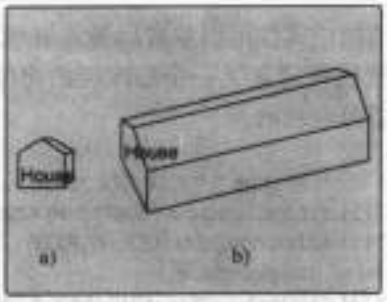


图7-21 SPHIGS中文字的非几何性。a) 变换前，b) 变换后

7.8 屏幕的更新和绘制模式

SPHIGS经常更新屏幕图像以适应当前CSS和视图表的状态。以下动作能使屏幕图像无效：

- 视图表表项的改变。
- 结构被关闭（经过打开和编辑后）。
- 结构被删除。
- 结构被提交或不被提交。

一旦调用SPHIGS完成以上动作中的任意一个，它都必须重新生成屏幕图像以显示所有提交的网络的当前状态。至于SPHIGS如何生成图像则与程序所选择的绘制模式有关。这些模式表示了在再生质量与速度之间做出选择：质量越高，则绘制图像所需的时间就越长。设置绘制模式由下面的过程进行：

320
↓
322

procedure SPH_setRenderingMode (*mode* : WIREFRAME / FLAT / LIT_FLAT / GOURAUD);

这里我们列出了SPHIGS中的四种绘制模式；我们将在第14~16章对它们进行更加充分的讨论。

1. 线框绘制模式

WIREFRAME（线框）模式是速度最快但也是最不逼真的显示形式。这时显示出来的只是由物体的边构成的线条图。此时，视图空间内所有物体的所有边的可见部分都被显示出来，且

不做消隐。所有的图元按照时间的顺序（即遍历程序在数据的提交结构网络中遇到它们的顺序）一一画出；这个顺序与7.3.4节中提到的由视图索引确定的显示优先级有关。

在这种模式下，各边的属性将会以其被设计的模式影响到屏幕的外观；事实上，当一个边标志被设为EDGE_INVISIBLE时，整个的填充区域和多面体都将是不可见的。

2. 明暗处理的绘制模式

在其他三种绘制模式下，通过用填充多边形绘制，SPHIGS以更加逼真的方式显示填充区域和多面体。阴影区域的出现严重地增加了绘图过程的复杂度，因为空间顺序变得很重要——物体被遮挡的部分（由于它们被邻近的物体所遮蔽）不应被画出。确定可见面的方法（即消隐）将在第15章中讨论。

对于这三种绘制模式，SPHIGS对各个面可见部分的内部像素生成明暗效果；绘制的质量因模式的不同而不同。对FLAT模式，即本章的插图中经常采用的，一个多面体的各个面都用当前的内部颜色画出，不考虑场景中光源的影响。如同线框方式一样，边的可见部分被显示出来（如果边的属性标为EDGE_VISIBLE）。如果内部颜色被置为背景色，则只有边被显示出来——FLAT方式的这种用法将会得到图7-9a和图7-14c中那样的图形，即消隐后的线条图。

另外两种高质量的绘制模式生成由光源照射得到的图像[⊖]；光照和明暗处理模型将在第16章中讨论。这些图像的明暗度是非均匀的，每个像素的颜色基于但并不等于内部颜色属性的取值。在LIT_FLAT方式下，同一平面上的各个像素具有相同的颜色，该颜色取决于光线与平面之间的夹角。由于每个小平面都有一个的统一颜色，整个图形看起来棱角分明，相邻平面在公共边上的反差很明显。而GOURAUD明暗处理则消除了这些棱角，使图形显得更为平滑。

在FLAT方式下，边标志属性应被设为EDGE_VISIBLE，因为没有可见边，观察者只能确定出物体的轮廓线边界。但在最后两种高质量模式中，边的可见性通常是关闭的，因为明暗处理能帮助用户确定出物体的形状。

323

7.9 用于动态效果的结构网络编辑

对于任何数据库，我们都希望不仅能创建和查询它用于显示，还能以一种便利的方式修改它，SPHIGS中的结构数据库也是如此。一个应用程序通过本节所介绍的过程来对结构进行编辑；如果应用程序本身含有自己的模型，那么它就必须保证二者在修改时的前后次序。运动动力学需要对模型变换和观察变换进行修改；更新动力学的变化要求对结构进行变动乃至替换。当变动相对比较小时，程序员可以选择对结构的内部元素进行修改；然而当变动很大时，通常情况下就只有删除并重建整个结构了。

在本节的剩余部分，我们将提供编辑内部结构的方法；查阅SPHIGS的参考手册能得到有关编辑操作的信息，以及有关过程的详细描述。

7.9.1 利用索引和标记访问元素

SPHIGS和PHIGS的基本编辑功能类似于采用行号来定位的老式的行程序编辑器。结构中的元素从1到N建立索引；一旦有元素被加入或删除，同一结构内索引号排在后面的所有元素便相应地把自己的索引号加1或减1。惟一的当前元素是其索引被存储在元素指针状态变量中的元素。当一个结构随着SPH_openStructure调用而打开时，元素指针置为N（指向最后一个元素）或置为0（结构为空）。该指针在有新元素插入到当前元素之后时加1，在前元素被删除时减1。该指针也能由编程人员通过绝对或相对的定位命令来明确地设定：

⊖ PHIGS+提供的多种控制绘制的功能，包括多光源的位置和颜色的定义，决定物体光照性质的材质属性等，见第14~16章。

void SPH_setElementPointer (**int** *index*);
void SPH_offsetElementPointer (**int** *delta*);

由于在同一父结构中，一个元素的索引会随着它前面元素的添加或删除而改变，利用元素索引号来定位元素指针很容易出错。因此，SPHIGS允许应用程序在一个结构中放置界标元素，称为标记。一个标记元素在生成时都给定了一个整型标识符：

void SPH_label (**int** *id*);

应用程序可以通过下面的函数来移动元素指针：

void SPH_moveElementPointerToLabel (**int** *id*);

然后该指针向前移动搜索给定的标记。如果在找到这个标记之前到达结构的末尾，搜索不能成
324 功终止；因此，程序将建议你在搜索标记前把指针移到结构的头部（索引为0）。

7.9.2 内部结构的编辑操作

最常见的编辑操作是在一个结构中插入新元素。一旦元素生成过程被调用，新元素就会立刻放置在当前元素的后面，同时元素指针加1以指向这个新元素[⊖]。

另外一种插入方式是要把给定结构中的所有元素拷贝到另一个打开的结构中去(放置在当前元素之后)：

void SPH_copyStructure (**int** *structureID*);

删除元素的过程如下：

void SPH_deleteElement (**void**);
void SPH_deleteElementsInRange (**int** *firstIndex*, **int** *secondIndex*);
void SPH_deleteElementsBetweenLabels (**int** *firstLabel*, **int** *secondLabel*);

在以上各种情况下，删除完成后，元素指针将指向被删除元素的前一个元素，所有剩下的元素都将重新编号。第一个过程删除当前的元素。第二个过程删除两个指定元素之间（包括这两个元素在内）的元素。第三个过程有点类似，但并不删除两个标记元素本身。

请注意，这些编辑工具都影响整个的元素或元素集；这里并没有提供对一个元素内部的数据域进行选择性编辑的方法。因此，举例来说，当程序员需要改变多面体内部的一个顶点时，就必须重新定义整个多面体。

一个编辑的例子

让我们来看一下对一个简单街道示例进行的修改。现在我们的街道只包括第一个房子和村舍，前者固定，后者可以移动。我们在村舍前建立一个标记，这样我们稍后可以通过修改变换来移动村舍。

为了移动村舍，我们在此重新打开街道结构，将指针移到标记，再定位到变换元素，替换变换元素，然后关闭结构。关闭结构后屏幕将自动更新，于是村舍在新位置上显示出来。有关代码见图7-22a，操作的顺序见图7-22b。

7.9.3 改进编辑方法的一些实例块

前面的编辑例子建议我们在需要编辑的元素前放置标记，但是要创建如此多的标记显然是一件麻烦事。下面介绍几种避免这个麻烦的办法。首先是用两个标记把要修改的一组元素包括起来，
325 然后利用标记来删除或覆盖整个元素组。另一种常用的方法是把元素分组处理为固定的格式，每个组引入一个标记。要编辑组内的任一元素，只需把元素指针移到该组的标记，然后通过偏移量将指针从该标记定位到组的内部。由于组的格式固定，偏移量可以很容易地通过一个整数确定。

⊖ 我们在编辑的示例中使用“插入”模式，其实，SPHIGS还支持“替换”编辑模式，这种模式用新图元覆盖现有的元素，细节见参考手册。

```
SPH_openStructure (STREET_STRUCT);
   /* 当一个结构被打开时，元素指针初始化其最表端。 */
   /* 我们必须移动指针到开始处，于是我们可以搜索标记 */
   SPH_setElementPointer (0);
   SPH_moveElementPointerToLabel (COTTAGE_TRANSLATION_LABEL);
   SPH_offsetElementPointer (1);      /* 现在指针指向变换元素 */
   SPH_deleteElement ();              /* 在此我们通过删除/插入组合来替代 */
   SPH_setLocalTransformation (newCottageTranslationMatrix, PRECONCATENATE);
SPH_closeStructure ();
```

a)



b)

图7-22 编辑操作。a) 执行编辑的代码，b) 编辑过程中的结构序列。黑色三角形表征元素指针的位置。（为了图示的目的已简写了调用的语法。）

该方法的一个特例就是，通过在结构执行（structure-execution）元素前附加一个属性设置元素的通用列表，来设计实例化子结构的一个标准方法。这种元素序列的一个典型格式被称为实例块（instance block），如图7-23所示：首先用一个惟一的标记来标识这整个块，然后是一个内部颜色设置，接着是三个基本变换，最后是符号结构的调用。

我们可以创建一些符号常量来提供偏移量：

```
const int INTERIOR_COLOR_OFFSET = 1;
const int SCALE_OFFSET = 2;
const int ROTATION_OFFSET = 3;
const int TRANSLATION_OFFSET = 4;
```

实例块的固定格式的使用保证了对于任何实例都能用同样的方式来修改某个特定的属性。为了改变一个特定实例的旋转变换，我们采用如下代码：

```
SPH_openStructure (ID of structure to be edited);
    SPH_setElementPointer (0);
    SPH_moveElementPointerToLabel (the desired instance-block label);
    SPH_offsetElementPointer (ROTATION_OFFSET);
    SPH_deleteElement ();
    SPH_setLocalTransformation (newMatrix, mode);
SPH_closeStructure ();
```

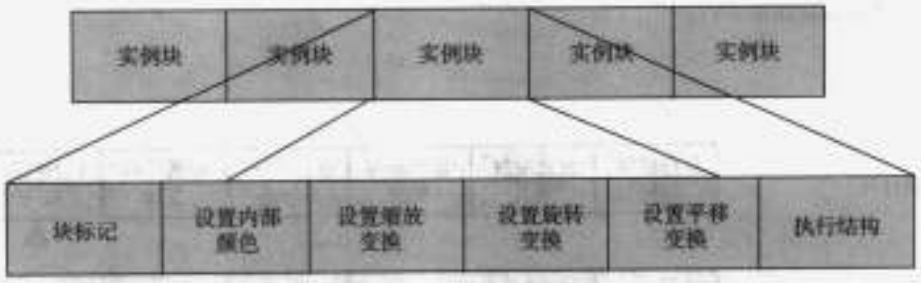


图7-23 实例块格式的例子

实例块的另一个好的特性是引入每个块的标记很容易定义：如果应用程序中维护着一个内部数据库来识别物体的所有实例，那么很自然地，程序内部可以通过给每一个标记设置惟一的值来识别这些实例。

7.9.4 如何控制屏幕图像的自动再生

326
-
327

SPHIGS经常更新屏幕图像以反映结构存储数据库及其视图表的当前状态。但是有时我们不希望频繁地重新生成屏幕图像，一方面是为了提高效率，另一方面是为了避免用户看到编辑过程中一些无关的或者比较模糊的连续变化的图像。

其实我们可以看到，SPHIGS在编辑结构的时候是将重新生成图像的功能挂起，这样不管发生了多少变化，图像只有当该结构被关闭的时候才会重新生成。为了提高效率，还可以采用“批处理”更新方法。因为“批处理”中图元的任何一个删除或变换操作都有可能破坏屏幕图像，从而需要进行图像的修复处理甚至重新遍历一个或多个视图中的已提交的网络。但不管怎么样，让 SPHIGS 在再生屏幕之前一次性计算多次操作的累积的效果显然会快于依次计算每次操作之后的图像并刷新。

连续性改变不同结构也有和上面类似的结论，例如，连续调用*deleteStructure* 删除结构及其子结构时。为了避免这个问题，应用程序可以在这些改变之前将屏幕自动再生功能挂起，并在操作之后再解除挂起。挂起和解除挂起的函数为：

```
void SPH_setImplicitRegenerationMode (ALLOWED / SUPPRESSED value);
```

当屏幕再生功能被挂起时，应用程序仍然可以调用

```
void SPH_regenerateScreen (void);
```

显式地要求屏幕再生。

7.10 交互

SRGP 和 SPHIGS 的交互模块是基于 PHIGS 规范的，因此它们设置设备的模式和属性以及获取量度的手段是一样的。SPHIGS 键盘基本上与 SRGP 相同，不同点在于在 NPC 空间上 SRGP 返回的原点不含有 z 坐标值。SPHIGS的定位设备带有一个通常为常量的 z 坐标值的附加域。另外，SPHIGS 增加了两个交互功能。第一个是关联拾取，即增加了对用户拾取的物体标

识的定位功能。另一个是支持菜单的选择设备，这在相关的手册中有说明。10.1 节将对PHIGS的交互设备进行一个整体的回顾。

7.10.1 定位器

SPHIGS 的定位器返回 NPC 坐标系下的光标位置（其中$z_{NPC}=0$）⊖，以及包含该光标且具有最高优先权的视图的索引号。

```
typedef struct {
    point position;      /* [x, y, 0]NPC屏幕位置 */
    int viewIndex;       /* 其视口包含光标的视图的索引号 */
    int buttonOfMostRecentTransition;
    enum {UP, DOWN} buttonChord[MAX_BUTTON_COUNT];
} locatorMeasure;
```

328

当两个视口有重迭，且光标含于其交集中，在第二个域中返回在视图表中具有最大索引号的视口。这样，就可以利用这些视图索引建立起视图的输入和输出优先级。视图索引域非常有用的原因很多。考虑一个允许用户交互地指定视口的大小的应用程序。比如可以移动或改变Windows的管理窗口。响应一个改变窗口的提示，用户只须拾取视口内的任何位置。应用程序可以根据索引号确定哪个视图被拾取，而不必通过检测一个点是否在一个矩形域内，对视口边界做逐个判断。对于涉及只用于输出的视图的应用程序，则可以根据索引号判断是否需要调用一些相应的过程。

7.10.2 关联拾取

因为 SPHIGS 的程序员是以已建模的实体为单位来考虑问题的，而非组成图像的像素；所以应用程序如果能够确定用户拾取的图像的实体标识将是非常有用的。因此本节将讨论定位器的最基本用途，即为关联拾取过程提供输入的 NPC 点。我们已经知道，在 SRGP 中，关联拾取在平面世界中确定命中问题是非常简单的。我们认为其图像与定位的位置足够近的图元被用户选中。当物体是相互重叠时，会有多次命中，一般选择最后画的物体，因为它被认为在最“上面”。这样在二维关联拾取程序中是以时间的逆序来检测图元的，最先被检测到的物体就是被拾取的物体。但是如后面所提到的原因，在三维中拾取对象时，这种重叠的层次结构将变得相当复杂。好在 SPHIGS支持这种应用。

1. 层次结构中的拾取

我们先想一下拾取的层次结构引入的复杂性。第一问题是用关联拾取来确认被拾取的物体时应当返回哪些信息？这个时候一个结构标识（ID）是不够的，因为一个结构标识无法区分一种结构的多个实例。只有全路径，即沿从根部到被拾取图元的完整祖先的描述，才能够提供惟一的标识。

第二个问题是当某个特定图元被拾取时，在拾取的层次结构中，用户所选中的应当在第几层？例如，当用户将光标放在机器人的手指处时，用户的本意是去拾取手指、手臂、上身还是整个机器人？有时，用户想拾取的是整体；有时只是想去拾取其中的一部分；任何一个层次结构都是有可能的。有些应用程序通过提供反馈机制允许用户依次沿着层次结构从图元浏览到根部，目的是确切地指定到所需要的层（参见习题 7.13）。

2. 比较的准则

当我们确实需要在三维中比较与已定义的物体何等接近时，由于定位器只能提供二维的

⊖ 在PHIGS中，定位器返回三维世界坐标系中的点。许多PHIGS实现中并不能返回有意义的z值，只有支持控制盘和多实时视图的高性能工作站才能提供舒适的获取三维点的人际交互界面（参见第8章）。

NPC 坐标值，所以与定位点相对应的图元的 z 坐标值是无法被利用的。也就是说，SPHIGS 进行比较时只能根据其屏幕图像的坐标值，而不能利用图元在世界坐标系中的坐标值。但不管怎
329 样，如果一个物体被命中，则它必定是关联拾取的候选图元。在线框模型中，SPHIGS拾取的是在遍历过程中遇到的第一个候选物。因为线框模型不包含明显的深度信息，所以用户不可能期望深度相关的关联拾取。（这种策略的效果是优化了关联拾取。）在真实感的绘制模型中，SPHIGS 拾取的候选物是最接近视点的击点（即NPC点位于与用户点击方向垂直的图元曲面上），即最前面的点。这将在7.12.2节进一步讨论。

3. 关联拾取工具

为了完成关联拾取，应用程序调用 SPHIGS 的关联拾取工具[⊖]，它的参数是一个 NPC 点和一个视图索引号，而这些参数通常是上一次与定位器交互返回的值。

```
void SPH_pickCorrelate (
    point position, int viewIndex, pickInformation *pickInfo);
```

返回的信息通过拾取路径标识被拾取的图元及其祖先，拾取路径在图7-24中以C语言数据类型的格式描述的。

```
typedef struct {
    int structureID;
    int elementIndex;
   /* 枚举类型：折线、多边形、执行结构等 */
    elementTypeCode elementType;
    int pickID;
} pickPathItem;

typedef pickPathItem pickPath[MAX_HIERARCHY_LEVEL];

typedef struct {
    int pickLevel;
    pickPath path;
} pickInformation;
```

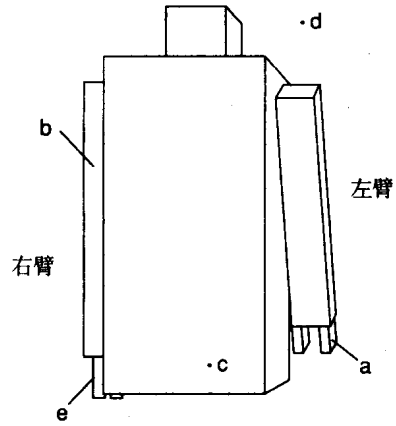
图7-24 拾取路径存储类型

当没有图元接近光标位置时，返回的 *pickLevel* 的值是 0 并且 *path* 域的值是没有定义的。当 *pickLevel* 大于 0 时，*pickLevel* 的值是从根部到被拾取图元的路径的长度，即图元在网络中的深度。而且此时*path*数组的项从 [1] 到 [*pickLevel*] 分别返回路径中从根部到被拾取图元的结构元素的标识。最深层（即项[*pickLevel*])标识的元素就是被拾取的图元；其他层（从项[*pickLevel*-1] 到 [1] ）标识的元素都是一些结构执行。*path*中的每个标识对应一条记录，该记
330 录给出了包含该元素的结构的结构ID，该元素在其所在的结构中的索引、代表该元素类型的代码以及（下面将要讨论的）该元素的拾取ID。

图7-25用图7-15所示的机器人上身的结构网络，并说明在该结构的显示图像中几个拾取返回的拾取信息。

拾取路径如何惟一标识结构（该结构在层次中可能调用任意多次）的每一个实例？例如，我们如何识别是拾取机器人的左手指还是拾取右手指？如图7-25中的点 a 和 点 e所示，这两个手指的拾取路径只是在其根部不同。

⊖ 完全的PHIGS有拾取的逻辑输入设备，它返回和SPH_pickCorrelate过程一样的测量值。



参见图7-15所示的结构网络

(a) level = 3
path[1] : struct UPPER _BODY, element 7
path[2] : struct ARM, element 3
path[3] : struct THUMB, element 1

(b) level = 2
path[1] : struct UPPER_BODY, element 11
path[2] : struct ARM, element 1

(c) level = 1
path[1] : struct UPPER_BODY, element 1

(d) level = 0

(e) level = 3
path[1] : struct UPPER_BODY, element 11
path[2] : struct ARM, element 3
path[3] : struct THUMB, element 1

图7-25 关联拾取的例子

相对结构ID而言，拾取标识符实际上是一种解决拾取相关的更好方案。虽然元素的索引号能够用于标识每个独立的元素，但是在结构被编辑时它常常会发生变化。所以采用拾取ID更为简单，因为当编辑其他元素时拾取ID不会受到影响。拾取ID的默认值是0，并且被包含在结构中。可以通过调用下面的过程来创建拾取ID元素：

void SPH_setPickIdentifier (**int** *id*);

这些拾取ID在显示遍历时被忽略，另外也没有继承的概念。当SPHIGS 开始遍历任何结构时，不管一个结构是子结构还是根，拾取ID的初始值都是0。由于这两方面的原因，拾取 ID 不同于结构的属性。在一个结构内的多个图元可以分别拥有惟一的 ID，也可以共享一个ID，这样就很好地解决了同一个结构内任意的关联拾取，这也正是应用程序所需要的。虽然标记与拾取 ID 采用不同的机制，前者用于编辑，后者用于关联拾取，但它们在应用的过程中常常是相关联的。尤其在结构使用7.9.2节提到的实例块技术组织时，拾取 ID 元素同时也是该实例块的一个组成部分，并且拾取 ID 本身的值通常被设成与块标记相同的整数值。 331

7.11 其他输出特性

7.11.1 属性包

标准的 PHIGS 提供了间接设置属性值的机制。应用程序初始化的过程中可以将属性值存储在它的属性包（attribute bundle）中。每种图元都有自己的属性包，并由PHIGS 软件包为许多属性包提供存储空间，每个属性包在 PHIGS 软件包中都由惟一的整型ID标识。例如，我们可以将“偏爱”的折线段属性集存储在属性包 1 中。随后在进行结构的编辑时，我们必须通过插入元素的方法为折线图元提供说明，使得在遍历的执行过程中折线段的属性是从属性包 1 中获取（而不是从显式指定的遍历属性状态中获取）。

属性包通常被用做“速记”，以简化定义属性的任务。考虑大量无关的图元也必须以相同的这些属性出现的应用程序。因为既然这些图元是与这些属性无关的，所以继承机制没有作用。其实，如果没有属性包，应用程序不得不在结构网络的任何地方冗余地指定所需的属性集。

有时 PHIGS 包的实现对属性包进行初始化，是为了提供与各种工作站相关的预先选定的属性集，使得工作站的优点能够被充分利用。应用程序员可以选择被提示的绑定的初始值，也

可以通过属性包编辑命令修改这些值。动态修改对象外观的简单机制是在不改变结构网络的前提下改变属性包表中包的定义。

7.11.2 高亮度与不可见性的名字集

SPHIGS 支持两种传统的反馈机制，在应用程序中它们常常与 SPHIGS 的拾取功能相关联。这两反馈机制分别是加亮物体和使物体不可见。前者的典型应用是在用户拾取物体时提供反馈，后者是使屏幕只显示必要的信息。在默认的情况下，所有的图元都是可见的和不加亮的。一个图元集可以被冠以一个整型的*名字*，用以在锁紧可见性或加亮状态时标识该图元。

因为一组无关的图元可以共用一个名字，并且一个图元可以拥有多个名字，因为这种命名特点，与结构调用引进的结构层次不同，一个复杂的物体有多种组织方式。例如，一个办公室模型可以表示为一系列的楼层子结构，也可以表示为一些系统的集合体，如供水网络、电线等等。只要简单地给所有的管道图元赋予同一个名字 (PLUMBING)，这样即使每个管道分散在实际结构层次中，我们也能以一个整体的形式找到它们。

如果想让供水系统不可见，我们只要在全局的*不可见过滤器*中加上PLUMBING 这个名字就
332 行了，这时屏幕就会立即更新，所有的管道的图像都不见了。如果除了电力系统之外所有的子系统的名字都设置了不可见过滤器，则屏幕上就会只显示电力系统。高亮度过滤器的工作原理是类似的。这两种过滤器的初始状态都是空的，只有当显式添加或去掉名字，它们才会发生变化[⊖]。应当注意，过滤器一旦发生变化（像改变视图），就会触发屏幕图像的重新生成。这些操作对 CSS 绘制视图的改变量与传统数据库程序的查询时显示数据的不同“视图”是一样的。

这种将名字与图元动态绑定在一起的方法非常类似于将属性赋值给图元的方法。类似于部分显示遍历状态，SPHIGS 维护由零个或多个名字构成的*遍历名字集*。根继承空的名字集。与一般属性一样，当被调用时，子节点继承父节点的名字集。这样一个部件对象的多个实例就可以共同拥有同一个名字，也可以分别命名。在 SPHIGS 的参考手册中阐述了如何在遍历的执行过程中往名字集中添加或从名字集中删除名字的结构元素。

7.11.3 图像交换与元文件

虽然 PHIGS 和其他标准的图形软件包通过系统无关或设备无关提高可移植性，但是由于性能的原因，在一个特定环境中实现的软件包常常具有不可移植的高度优化。例如，CSS 的内部表示可能会在结构或元素中含有针对特定机器的信息。为了给 PHIGS 的不同实现提供一个交换中介，图形标准化委员会已经定义了一种归档文件格式。这部分的标准定义是 CSS 内容的机器无关及环境无关，但是不包含观察信息。PHIGS 的归档文件是在给定时间的数据结构的一个可移植的快照，这样就使得 PHIGS 实现几何模型的共享。

PHIGS 实现同时也支持书写元文件，元文件可以包含应用程序当前在显示表面上所显示的内容的快照。当这些元文件与 ANSI 和 ISO CGM (计算机图形元文件) 标准[ARNO88] 相一致时，这些元文件中所包含图像就可以传送给台式机或交互式图形艺术增强工作站这样的应用环境。CGM 文件也是一种与机器及环境无关的 CSS 形式，但与归档文件不同的是CGM 文件中包含图像的视图信息。

CGM 的典型创建方式是由 PHIGS 输出设备驱动程序遍历 CSS 从而产生 CGM “虚拟设备”的代码，就好像普通设备驱动程序给一个真实的显示系统产生显示代码一样。其他系统通过输入设备驱动程序将元文件读入CSS中，并将元文件格式转化成该系统所用的特定的文件格式。因为

⊖ PHIGS的可检测滤波器允许应用程序指定不可拾取的图元。另外，PHIGS滤波器功能强，可支持包含和排除两种滤波方式。

元文件是三维场景的二维视图，所以任何通过 CGM 获得的图像信息的应用程序都将有二维视图：最初三维的几何信息将会丢失。如果想用标准格式交换三维模型就必须采用归档文件。 333

其他类型的元文件可能会用于调试或备份。跟踪元文件是包含了以时间为序记录了图形软件包过程（及其作为参数发送给它们的）所有调用的列表的历史抄录文件。因此，为了重建 CSS 就必须从头到尾运行一遍。另外还有一些类型的文件记录用户的操作：运行带有这种文件格式的应用程序重新产生 PHIGS 的调用的最初顺序以及相同的 CSS/图像。现在没有这种文件的标准，而且也没有计划去创建该标准。另外，CGM 和 PHIGS归档文件包含历史记录。

7.12 实现问题

SPHIGS 的许多内部功能包括视图表和 CSS 的维护(编辑)和使用(遍历)。既然维护是传统数据结构的问题，而不是图形学专有的问题，这里就不加以讨论。本节将主要讨论结构显示和实现关联拾取的机制。

更新屏幕图像必然会调用显示遍历过程。当允许隐式地重新生成屏幕图像时，下面的这些操作会引起遍历：关闭结构，提交及去掉提交，改变观察变换，改变绘制模式，以及改变过滤器。为了生成视图的内容，提交给该视图的所有结构都会被遍历到。

为了显示提交的结构网络，SPHIGS 通过递归下降的方法遍历其构件结构元素，并根据元素的类型对该元素执行相应的操作。这种将模型映射为屏幕图像（硬拷贝）的显示过程在 PHIGS 中指的是显示遍历，但更一般的是指绘制；相应软硬件中的实现指绘制流水线。

关联拾取遍历非常类似于显示遍历。将遍历过程中遇到的图元与定位器的位置相比较，从而找到候选物。在使用线框绘制的情况下，遍历一旦遇到第一个候选物就会停下来。在其他情况下，就会执行全面的遍历，最终选中的是其 z 坐标与视点最接近的候选物。

7.12.1 绘制

图7-26阐述了实现显示遍历的绘制流水线的概念。第一个步骤是 CSS 本身的实际深度优先遍历。（相反，如果所用的是瞬时模式的图形软件包，则应用程序可以遍历应用模型或过程性地产生图元和属性。）遍历过程遇到的每个图元都会被传递给流水线的剩余部分：首先，（第5章所讲的）模型变换被用来将图元从模型坐标系映射到世界坐标系；然后，观察操作被用来变换和裁剪图元使它适合于规范视见体；接着再映射到（第6章所讲的）NPC并行管中。既然这个过程与显示设备无关，并且采用浮点坐标来处理顶点几何结构，那么遍历后面的流水线的部分通常指的是几何处理子系统。 334

流水线的后端取得了经过变换、裁剪的图元并产生了像素，我们将这种像素处理称为光栅化。在线框模式下，这个过程显然很直观：（通过缩放和平移，忽略z）NPC 坐标很容易映射成整数设备坐标；然后调用基本的光栅图形软件包中的画线函数进行实际的扫描转换。然而采用经明暗处理的绘制却相当复杂，它由如下的三个子过程组成：可见面判定（从视点上看，判定图元的哪个部分是可见的），扫描转换（判定图元图像所覆盖的像素），以及明暗处理（判定这些像素的颜色）。这三个子过程的实际执行顺序由绘制模式和实现方法决定。从第14章到第16章将详细描述光栅化子过程。

1. 遍历

既然绘制流水线除了第一个步骤外的所有步骤都在其他各章讨论，这里只需讨论遍历这一阶段。SPHIGS 可以这样简单地实现屏幕图像的再生：先删除图像，再重新遍历所有提交的根部（视图中含有其列表）。优化再生图像使得尽可能少地遍历 CSS 相当困难，因为一些微小操

作的作用也可能是巨大的。例如，当编辑结构时，很难断定屏幕的多少部分需要再生：有可能该结构根本就不被调用，从而根本不影响屏幕；但它也有可能是会被常常调用到的部件，几乎所有的视图都会用到它。甚至在实现的过程已经能够判断出只有一个视图被一个操作所影响时，刷新该视图都有可能破坏具有高优先级的有重叠的视口内的物体的图像。有效地恢复被破坏的部分一般说来是一个十分艰巨的任务，需要相当多的标记。



图7-26 SPHIGS的绘制流水线

当在高性能的工作站上实现时，图像能够以一秒一个片断的速度遍历和重新生成，有选择地重新生成屏幕图像所需的标记空间和系统时间的开销很可能是不值得的，所以通常需要全面
335 重新生成。不管是全面再生，还是有选择地再生方式，本节后面所讨论的双缓冲技术可以用来防止在显示遍历时出现视觉上的不连续性。

对结构的遍历既可以用硬件实现，也可以用软件实现。这里描述了一种过程语言实现的方法，它的伪代码如图7-27所示。这个过程继承了遍历状态，在活动期间它将该状态当作它本身的局部状态来维护。属性（包括名字集）和三个变换矩阵(GM、LM、及其产品、CMTM)组成了遍历状态。属性和CMTM从上到下一层层传下来，从父节点活动传到子节点活动。（根结构的过程的活动继承了同样的GM和以默认值填充的属性记录。）如图7-27所示，这个过程访问结构中的每个元素，并根据不同的元素类型分别实行相应的操作。

```
void TraverseStructureForDisplay (structID, attributeState, GM)
{
    LM = 单位矩阵
    CMTM = GM;

    for（结构structID中的每个元素）        {
        switch（元素类型）  {
            case 属性或名字集合修改
                更新   attributeState;
                break;
            case LM 设置:
                替换或更新        LM;
                通过将LM拼在GM的后面更新CMTM;
                break;
            case 图元:
                将图元传递到绘制流水线的其余部分;
                break;
            case 执行结构:              /* 一个递归调用 */
                TraverseStructureForDisplay
                    被扩展的结构ID, attributeState, CMTM）;
                break;
```

图7-27 显示遍历的伪码

```
            default:                    /* 忽略标记和拾取ID */
                break;
        }
    }
}  /* TraverseStructureForDisplay */
```

图7-27（续）

2. 通过范围检测的优化方法

前面所讲述的遍历过程无条件地遍历网络的内容。在遍历过程中，所有的结构调用都会被执行，并且不会跳过 DAG 的任一部分。但是通常并不是所有的网络对象都是可见的，因为进行遍历的时候，有效的模型变换和观察变换有可能会导致网络中的大部分位于视见体外。

在遍历的过程中从属结构中的什么信息是可以简单地拒绝的呢？例如当访问到元素“结构执行 S”时，则迅速在 NPC 空间上计算实例 S_i 的边界。然后，再问这样的问题：“S_i 是否完全在 NPC 的视口外面？”。如果它确实是完全在外面，则这个结构执行就可以被跳过，并且不要再进入结构 S 的内部。因为结构 S 有可能就是一个相当复杂的子网的根，不对它进行遍历则会 336
节省大量的时间。实际上，这种方法使我们可以通过比较结构根部的 NPC 边界和视口来判断是否需要简单地拒绝整个网络。

为了实现这种优化，我们需要一个简单的方法来计算任意复杂物体的边界和一个有效的方法将这些边界与 NPC 视口进行比较。NPC 范围可以满足这两个要求，NPC 范围定义为沿着主轴，完全包围物体 NPC 形式的最小包围盒。如果我们可以证明一个实例的范围与视口没有次交集，则我们可以推出该实例是完全不可见的。这种范围是非常理想的，因为视口本身是一个与主轴平行的框，并且计算与主轴平行的框的交集的代价是很小的（见习题7.5）。通过使用NPC范围获得的优化不仅是简单地拒绝，而且可以简单地接受——发现某个实例完全在视见体内，则不必对它进行裁剪。这种子结构的简单接受/简单拒绝测试的范围技术最早被用在BUGS 系统的硬件部分[VAND74]，另外在[CLAR76]中也有描述。

因为实例并不是显式存储在 CSS 中的，所以我们必须根据 CSS 中所存储的MC范围计算出 NPC范围。为了在遍历过程进行范围检测，我们必须先将 MC 范围转化为 NPC 的，再计算实例的 NPC 范围。因为直接转换得到的NPC 范围不一定会刚好是竖直的，所以由转换后所得的 MC 范围来计算 NPC 范围，使得计算得到 NPC 范围可以直接用来与视口进行比较。

为了计算结构 S 的范围，我们必须计算其图元及其子节点的范围的并集。我们可以这样计算多面体或填充区域的范围：先遍历 MC 顶点列表，再经过局部矩阵变换，最后计算x, y, z的最大与最小值。这六个值定义了范围：包围盒的两个角点坐标在（x_{min}, y_{min}, z_{min}）和（x_{max}, y_{max}, z_{max}）。

另外还有一个问题是如何维护范围信息。应当在什么时候计算结构的MC 范围？只要结构被编辑就进行计算还不够。结构的 MC 范围不仅受结构本身内容的影响，而且还会受到其子孙节点内容的影响。因此，进行完每一个结构编辑操作，就必须重新计算一定数目的结构范围。而且重新计算需要对 CSS 中任意大的子集进行遍历。这种范围计算过程是可以优化的，我们可以把计算放在显示遍历的时候而不是在编辑之后。这种技术的优点在于只对那些可见的结构(提交网络的一部分)进行范围的更新，并且保证了响应任意大的批量编辑操作的时候不会重复地进行结构范围的计算。[SKLA90]更为详细地讨论了这种优化方法。

3. 动画和双缓冲技术

SPHIGS 的软件实现也适合用来快速构造动画原型，但不太适合于高质量的实时动画，因为

这种绘制的时间代价太高，并且当对“帧”间进行大量的编辑时范围检测技术并不能很好地工作。
337 动画原型通常是在线框模式下进行绘制，关掉自动重新生成屏幕图像的功能，并采用一个循环算法：应用通过编辑结构数据库来描绘下一场景，显式地重新生成图像；然后编辑来描绘下一个场景，如此不断继续下去。如果场景不是十分复杂，在线框模式下，对于不太多的图元，只用软件就可以实现接近实时的动画。如果有硬件的支持，对于高质量的明暗处理模式也能做到实时。

这种简单实现的副作用是：组成动画的“帧”之间，观察者可以看到屏幕正在被删除，并且或多或少可以看到物体是如何随着遍历的过程被重新画到屏幕上的。SPHIGS 实现可以通过双缓冲来减少这种视觉效果：它通过一个屏外画布或位图来保存下一帧，然后再将这一帧复制到屏幕上。如果这种技术被用在一个具有非常快的copyPixel操作的系统中，则帧与帧之间的切换是觉查不到的；但是场景中的物体的运动仍然会有点不连续。实际上，*动画率*（即每秒钟的帧数）会随着copyPixel开销的增加而减小，但很多情况下，所花代价与这种视觉效果的减少相比，仍是值得的。在4.4.1节中阐述的硬件双缓冲技术会有更好的效果，因为它去掉了copyPixel的时间。

7.12.2 关联拾取

在关联拾取的过程中，SPHIGS 遍历已提交给视口的那些网络，而那些指定的点就位于这些视口内。另外，这个过程的遍历几乎与显示时的遍历相同，既要维护模型变换矩阵，又要执行大量的绘制流水线。而且在显示过程中被忽略的拾取 ID 也将作为递归遍历过程中局部遍历状态的一部分加以维护。(属性组是没有必要维护的，因为在命中图元检测过程中 SPHIGS 是不考虑诸如线的粗度等属性的。)

让我们先分析一下在线框绘制条件下如何关联拾取，这时一旦找到第一个命中图元遍历就会停止。因为遍历是递归的，所以找到第一个命中图元的时刻非常容易确定拾取路径。遍历过程的每次活动只针对拾取路径的一层信息。在进行命中图元检测的时候，先将每个图元转换到 NPC 坐标系下，再同定位器的位置进行比较。(命中图元检测将在本节后面进行阐述。)当这个过程的活动发现了一个命中图元，就返回了，递归过程也就结束了。在返回之前，每个活动都在一个全局的拾取信息数组中保存相应的拾取信息（参见习题7.9）。

在经明暗处理的绘制的模式下，在遍历的过程中遇到图元的先后顺序并不意味着它们与图像射到屏幕的同一部分的其他图元的前后关系。因此，这时SPHIGS关联拾取的算法就不能简单地只选择首次检测到的命中图元。它必须遍历所有已提交的结构网络，维护一个候选命中图元的列表。当遍历完成时，必须比较这些候选物，找出命击中点与视点最近的候选物体——z 坐标值最大。为了计算候选命中点的 z 坐标值，我们将定位器的 x 和 y 坐标加到每个图元的方程中去：对于边采用直线方程，对于面片采用平面方程。(参见习题 7.7 和 7.8)。在 15.4 节将
338 阐述另一种采用硬件的可见面判定算法。显然有许多方法可以优化关联拾取的遍历过程，这包括在显示遍历中所采用的范围检测方法。

1. *命中图元检测的分析*

命中图元检测要用到这两种基本方法：分析和裁剪。为了分析命中图元检测，引入代数方程来判断图元的 NPC 坐标与二维的 NPC 定位器位置是否足够接近。向二维转换，可以通过忽略正投影的 z 坐标，或者采用6.5.4 节中的透视变换来创建一个正视见体。一些分析技术的例子如下：

- 在线框模式下，函数 PtNearLineSegment 用来计算在 NPC 坐标系下从光标位置到面片或填充区域的每条边的距离，以及到折线的每条直线段的距离。同一个函数可以用在经明暗处理的绘制模式下折线段图元。而直线方程被用于计算（参见习题 7.10）。
- 在经明暗处理的绘制模式下，函数 PtInPolygon 用来检测命中图元填充区域或面片的情

况。有一个常用的判断 NPC 光标位置是否在多边形内的算法，它基于 2.1.3 节介绍的奇偶校验规则，从光标所在位置发出一条射线然后计算它与多边形相交的次数。该算法遍历边表，检测交点，并处理特殊情况（例如，交点与顶点重合，边与射线共线）。3.7 节介绍的多边形扫描转换算法处理的问题非常类似，稍微修改就可以当做函数 PtInPolygon 使用（参见习题 7.12）。该算法处理更为一般的情况：凹的和自相交的多边形。如果能够保证多边形不自交或者水平射线只与两条边相交或者多边形是凸的，则优化的计算几何算法也是可以用的[PREP85]。

- 对于非几何的文本文字的命中图元检测非常简单，只要将定位器的位置与文本的屏幕矩形的范围相比较。
- 支持诸如椭圆、曲线、曲面等图元的软件包则需要更为复杂的像第11章和第19章所提及的排序的关联拾取算法。而且如 15 章所阐述的，遇到的问题与光线跟踪中遇到的非常类似。

2. 通过裁剪的命中图元检测

有些硬件裁剪设备和优化的裁剪软件工具可返回状态信息，允许应用程序判断图元的任何一个部分图像是否位于二维整数裁剪矩形内部，而实际上并不把该图元画出来。SPHIGS 的实现程序可以用这种裁剪方法来测试候选物：裁剪矩形设成拾取窗口（包围光标位置的小正方形）然后进行遍历。每个图元（转换成整数设备坐标）都传递给裁剪函数，该函数返回命中图元检测的布尔值（参见习题7.11）。反过来，如果裁剪函数不返回这样的状态信息，我们可以将每个图元画到屏外位图上，该位图采用拾取窗口裁剪矩形。如果像素确定没有发生变化，则该图元即为命中图元的候选物。 339

7.13 层次模型的优化显示

7.13.1 省略

我们可以将一个建筑物构建成由各个部分组成的层次结构。例如，它由很多楼层组成，每一楼层由办公室组成，如此等等。层次结构的内部节点不包含任何一个图元，图元只出现在诸如砖头、平板以及由多面体组成水泥板之类的层次上。虽然这种表示方法可能在建筑上非常有用，但在显示上就不那么有用，例如我们有时只想看看低精度的、去掉模糊的无关的细节之后的简单图像(这也使得绘制更快)。术语省略（elision）指的是由显示遍历程序所做的不再进入子结构的决定。

1. 修剪

7.12.1节介绍了如何通过在视口上检测 NPC 范围，判断子结构是否全部在视见体之外（即整体被裁剪），从而使得显示遍历过程避开执行从属子结构。这种类型的省略，优化显示遍历的典型特征，被称为修剪（pruning）。

2. 剔除

另外，遍历过程也可以检测从属的 NPC 范围的大小，如果在变换和投影之后，子结构的范围非常小，物体的图像被压缩成少数的几个像素，则该子结构可以省略。这种省略称为剔除（culling）。在支持剔除的系统中，应用程序常常指定一个最小的范围，比这个范围小的子结构会被剔除。

当一个子结构被修剪，则它根本就不会被画，但这不是最好的剔除方法。物体被剔除往往不是因为在当前视点下它不可见，而是因为它的图像太小以致于它的细节无法辨别。许多实现程序将它画成一种抽象的形式，而不是不画它，最常见的画法是平行的流水线表示它的 WC 范围，或只是简单的一个矩形（NPC 范围的二维投影）。

3. 层次细节省略

修剪和剔除都是优化技术，避免了不必要的遍历（修剪）或产生无用的图像(剔除)。省略

还可以让用户决定在 CSS 视图上显示多少细节。例如，用户在看整体的建筑模型时可以指定一个低的细节水平，这样建筑物中的每层仅显示为简单的平行管，也可以提高细节水平，从而看见办公室的墙。

MIDAS 微处理器体系结构的仿真器是最早通过显示处理器自动遍历子对象表示的系统，它依赖于子对象在屏幕上的投影大小。这种逻辑上的缩放功能使得在用户动态地缩小 CPU 体系结构框图中处理器部分时有更多的细节显示出来。另外，增加缩放因子，用户甚至可以看到数字表示的地址、数据以及在指令周期的仿真中控制字节沿着系统总线从源地址运动到目的地址。

MIDAS 中的省略是通过 BUGS 向量图形系统[VAND74]的条件执行功能实现的：它的层次显
340 示列表中包括一个条件子结构执行元素，它是用来检测子结构的屏幕范围。PHIGS + 的1988年的规范中以条件执行元素的形式描述了类似的特征，允许 CSS 内部元素显式地执行修剪和剔除。

7.13.2 参考结构

有些 PHIGS 和 PHIGS + 的标准实现允许非标准形式的结构执行，称为*参考结构*。它绕开了代价昂贵的状态保存和恢复的ExecuteStructure（结构执行）机制。有人认为一种更好的透明的提高效率的方法是优化ExecuteStructure操作的实现（使得操作只保存在任意给定时间时所需的状态）。然而这种观点却有些争议。实际更简单的方法是增加一个ReferStructure操作，使得程序员能够将它用于所调用的子结构并不具有任何属性设置元素的情况。

ReferStructure的另一个应用是允许子节点影响其父节点的属性。这在一组对象并不具有共同的父节点但确实需要继承相同的属性组时相当有用，包括潜在的任意数目的模型变换。在这种情况下，我们可以创建一个包含变换和外观属性设置的结构 *A*，然后让每个对象结构指向结构 *A*。以后通过编辑结构 *A*，可以间接地影响到所有指向 *A* 的结构。如果只有外观属性需要受到影响，PHIGS 属性包提供了另外一种标准机制来改变种类繁多的结构的外观。

7.14 PHIGS中层次模型的局限性

虽然本章的重点是几何造型中的层次结构，但有一点很重要，即明白层次结构只是许多种数据表示形式中的一种。本节将讨论一般层次结构的局限性，尤其是 PHIGS 中层次模型的局限性。下一节将介绍几种可以替代层次结构的模型。

7.14.1 简单层次结构的局限性

正如 1.7.2节中所提到的那样，有些应用程序并没有给数据提供真正的数据结构（如离散的绘图数据）或者最多只是对数据进行（部分）排序（例如以代数表示的函数）。许多其他的应用程序更自然地采用网络方式表示，即一般（有向）图（可能带有层次子网）。这其中包括电路和电子线路图、传输和通信网络以及化学工艺图。说明这种简单层次结构在一定的模型下的不足的另一个例子是Rubik 的魔方，这种魔方是一些构件的集合。在任何一种变换后，它的网络和层次（即层、行和列）都会发生根本性的变化。

对于其他类型的模型，这种简单的层次结构也是不足的。例如，绘图仪上的笔架是由水平和竖直的手臂移动的，因此也隶属于这些手臂。简而言之，不管应用程序模型是显示出纯层次结构、不具有层次的网络、具有交叉的网络中的层次结构，还是多重的层次结构，SPHIGS 都
341 可以用来显示该模型；但是我们可能并不希望或不能在最大限度地应用结构的层次。

7.14.2 SPHIGS“参数传递”的局限性

结构的黑盒特性对于模块化是有好处的，但正如下面机器人模型的例子那样，它具有局限性。例如，如何构造具有两只相同手臂的机器人，并使机器人用它的右手臂从桌子上拿起一个

圆柱，然后再带着圆柱离去？拿起的操作可以这样实现：在手臂结构中添加一个圆柱结构的调用，使得机器人或其手臂在移动时就可以带走圆柱。但是如果机器人只有一只手臂结构，但被其上身结构调用两次，那么结果就会导致左右两只分别带走一个圆柱！这样，我们只好建立起两个独立的手臂结构。它们分别拥有一个惟一结构 ID，并且每只手臂分别调用一次。让我们进一步扩展这个例子。如果我们希望有一只手臂是与被控制的手臂无关的，那么我们就必须创建两个完全不同的手臂。显然，在这种情况下，结构层次不像我们原来想的那么有用。

层次结构与一般结构实例的区别在于不同层次水平的变换设置。层次结构不支持结构的实例的原因是结构层次既不具有过程层次的参数传递机制，也不具有控制流构造。更准确地说，它是一种具有初步的在结构及其最有限的结构的条件执行（在SPHIGS + 中）之间进行交互的数据组织方式。我们已经看到父节点的变换被所有的子节点继承下来，并且不能有选择地影响特定的子节点。

相反，在过程的层次结构中，“根”的过程传递参数给根过程调用的过程，被调用的过程或者直接使用这些参数，或者进一步将这些参数传给底层的过程。这样，“根”过程就可以通过中间过程有选择地将参数传到任意深度的过程。而且通过参数传递，一个过程不仅能够控制底层过程操作的数据，甚至还能控制底层过程操作的方式。为了改变操作，底层的过程通过构造控制流来检测参数，并有选择地启用或禁用代码段。由于层次结构缺乏参数传递与控制流机制，在引言中对结构层次与过程层次的类比是相当肤浅的。

通过增强PHIGS 中的属性绑定模型，我们可以给层次结构中任意深度的选定的对象实例指定属性。具有这种机制的系统是 SCEFO [STRA88]，它允许程序员指定在遍历到达某个以路径名表示的状态时将要用到的属性，它的路径名类似于 PHIGS 拾取设备所返回的拾取路径。有了这种功能，可以通过利用两个手指实例具有不同的路径名这个事实有可能分别控制机器人手指的颜色和位置，而不必分别创建两个相同的虚拟的手臂框架。

PHIGS 的另一个参数传递机制的局限性在于它处理的变换和外观属性的继承性规则十分简单。它很难支持比几何变换更为复杂的操作，因此需要一个更为一般的模型来支持实体图元的集合运算（第12章）和诸如弯曲、锥化和扭曲等的变形操作(第20章)。 342

7.15 层次建模的其他形式

我们已经推断结构层次只是编码层次的一种方法——而并非总是最好的方法。本节将讨论结构层次的其他形式。

7.15.1 过程层次

从纯数据层次到纯过程层次的范围中，结构层次几乎总在数据那一端，因为它缺乏控制流。相反，模板过程（即定义模板对象的过程，它由图元或对子模板过程的调用组成）可以使用参数与任意的控制流。有两种不同的应用模板的方法。第一种是它们可以在诸如 SPHIGS 等保留模式图形软件包下使用。这里它们被用做结束创建结构层次的手段。第二种是它们可以用来给瞬时模式图形软件包指定图元和属性。这里它们不意味着结束结构层次，而意味着它们本身的结束；也就是说，它们是显示中所用的层次的惟一表示方法。在这种情况下显示遍历会受到过程遍历的影响——实际上只有当 CPU 提供合理的重新遍历速率时。（一般说来，为了达到平滑的动画效果每秒至少需要15帧；如果每秒30帧就会看起来相当地好。）过程本身实现继承性，并维护变换矩阵和属性状态。它所采用的技术类似于7.12节所讨论的遍历的实现中所用的技术。7.4节提到的纽曼显示过程是最早将过程层次用于动画的例子。

关联拾取在动态的过程遍历中有些棘手，因为它需要重新遍历过程层次。如果第一个候选

物作为选中的物体（即如果绘制模式是线框模式），那么要一检测到命中图元就停止遍历并从过程活动栈的任意层次上返回显然很难。如果不是采用线框模式，则必须具备一种与绘制流水线进行交互的方法，以便获得候选物及其相对光标位置的 z 值。每个过程也是相当复杂，因为它必须同时用在显示遍历与相关图形的遍历。

我们可以通过使用模板过程组合过程层次和结构层次来创建结构。例如，可以通过模板过程来创建机器人的每一个部分，每个过程根据被选中的是瞬时模式还是保留结构模式分别创建显示命令和 SPHIGS 结构。机器人的上身过程可以将参数传递给手臂和手指过程，从而对它们的变换分别进行初始化，以及允许它们分别独立运行。在保留模式下，这样机器人的模板过程的每个调用就创建了一个惟一的 SPHIGS 网络，该网络的所有子结构都有惟一的名字。例如，创建两个手臂结构，使它们拥有自己的分离的手指调用。在多重网络中仍然可以共享叶子节点，例如手指，因为调用程序可以以独立变换对它们进行实例化。为了创建惟一的结构 ID，我们
343 可以给每个根节点分别赋一个惟一的整数区间，并用这个区间内的整数给它的组成部分编号。

如果只用过程层次又会有怎样的局限性呢？首先，除非编程环境支持动态代码生成，否则在运行的时候很难编辑或创建过程。创建新数据结构则相当容易。因此，过程层次典型应用的情况是模板对象的集合已经预定义好了，并且只需动态地修改属性。当然在所有的 PHIGS 机器上都允许编辑由模板过程创建的结构。

另外如果过程层次被用来创建瞬时模式图形，则在经常进行重新遍历的过程层次所包含的 CPU 对应用处理的可用性就会大为降低。在多处理器系统中我们可以卸下一个 CPU，并换上一个专门用于过程遍历的 CPU。相反，在保留模式下，结构网络遍历可以通过专用的硬件协处理器或独立的 CPU完成。

第三，甚至进行一个很小的编辑，瞬时模式的过程层次都必须进行全体的重新遍历，同时必须将所有的显示图元重新传送给图形软件包。对于跨网络或通信线路连接的显示系统，这种需求就会导致巨大的通信量。对于微小的编辑，如果在显示的外围设备上保存和管理一个结构数据库，使得可以进行增量式修改和快速遍历，则将会大为提高 CPU 以及通信系统的效率。

7.15.2 数据层次

不同于过程层次，数据层次非常适于动态创建。与模板过程层次相同的是，它也可以用于瞬时模式和保留模式图形软件包的连接中。如果采用瞬时模式，则 CPU 必须重新遍历应用模型，并且要足够快地驱动软件包，从而提供动态的更新率。创建和编辑对象可以通过改变应用模型，并且重新遍历从而更新屏幕。应用程序必须通过重新遍历自己进行关联拾取。但是类似于瞬时模式下的过程层次，如果显示子系统是跨越通信网络的，那么对每次更新都重新传输图形命令会比只给外围设备传送结构数据库的刷新命令慢得多。

类似于结构层次技术，数据层次也缺乏过程层次方法的灵活性，因为它也没有控制流机制。这些必须通过数据结构中的标志包括进来。具有运行时代码的创建和绑定的面向对象环境提供了一个有吸引力的数据层次和过程层次的一般性组合。对象-子对象层次符号与类-实例层次符号之间存在一种自然匹配。随着处理器性能的改善，面向对象的环境在动态计算机图形学中会成为占主导地位的范型。

7.15.3 利用数据库系统

既然通用数据库比专用数据库系统更为强大，我们就应该在计算机图形学中采用标准的数据库系统[WELL76, GARR80]。但是，这样的数据库是用来处理存在辅助存储器中的大量数据，
344 并按人的时间尺度来提供响应时间。它们被设计用来处理用户-输入查询或者甚至是具有时间

测量的批处理事务。这些时间最好以毫秒为单位，而实时的图形学需要以微秒为单位来访问元素。如果数据库已经对快速遍历进行了优化并有内部图形数据类型及其操作符，那么最好将这个数据库放在内存中。它至少应当能够为检索数据调用过程，传递从数据库的域中提取出来的参数来传递数据。

虽然有些图形系统采用关系数据库，但受关系模型影响的结构的局限性实际上与慢速的标准关系数据库是一样的，它们都限制系统去研究环境。由于面向对象的环境对组合数据和过程非常有用，这样的环境如果与面向对象的数据库相关联，则有望去掉关系数据库的这种限制。然而，面向对象的数据库的低性能使得在近期内不适合于实时图形学。

7.16 小结

本章介绍了几何模型，重点是表示零件装配的层次模型。虽然许多数据与对象类型并不是层次的，但至少许多人造的对象或多或少的具有这种特性。PHIGS 以及我们的改进版本 SPHIGS 是用来提供几何对象，特别是以多边形和多面体的层次模型保存的几何对象的有效的自然的表示方法。因为这些软件包保存对象的内部数据库，程序员很容易就可以对数据库做出修改，并且软件包自动地产生更新后的视图。这样应用程序就可以建立和编辑数据库，特别是响应用户的输入，而软件包则负责产生数据库的特定的视图。这些视图应用了大量的绘制技术，对图像质量与速度进行了权衡与折衷。软件包也提供了定位器和选择输入设备以及关联拾取，使得对象可以在层次结构的任何一个层次上进行拾取。高亮度过滤器和可见性过滤器被用来启用或禁用针对物体外观的其他形式的控制。

由于结构特性及其查找和编辑手段的限制，这样一种专用系统最适合于运动动力学和光更新动力学，尤其是如果结构数据库能够在优化为 PHIGS 外围设备的显示终端上维护。如果许多结构数据库在连续图像之间必须更新，或者如果应用程序数据库能够快速地被遍历且计算机和显示子系统之间不存在瓶颈，那么在瞬时模式下使用图形软件包且不保留信息会更加有效。

结构层次位于纯数据层次和纯过程层次之间。它具有数据层次的特点——拥有动态编辑的优点。同时通过属性遍历状态机制，它还允许将参数传递给（几何和外观属性）子结构的简单形式。但是，由于缺乏一般控制流的构造，这种参数传递机制是有限制的，而且结构无法有选择地在子结构不同实例中设置不同的属性。相反，模板过程可以被用来建立(层次)结构的多个副本，它们在结构上是相同的而在子结构的几何属性或外观属性上是不同的。另外，它们可以 345
用来驱动瞬时模式软件包。

SPHIGS 是面向由多边形与多面体构成的几何模型，特别是具有层次特性的几何模型。第11章和第12章介绍的几何模型有更加复杂的图元以及图元的组合。在进入更加先进的造型方法之前，我们在后面几章中先介绍一下交互工具、技术和用户界面。

习题

7.1 a. 通过给图7-16中的机器人添加一个基座，使得它的上身可以在这个基座上转动来完成该机器人模型，并建立起它在房间内运动的简单动画。

b. 建立一个产生动画的 SPHIGS 应用程序。在该动画里，有一个只有一只手臂的机器人，它走到一张放着一个物体的桌子那里，拿起物体，并带着该物体离开。（采用一只手臂的机器人的理由参见 7.14节。）

7.2 增强机器人的动画来提供用户的交互功能。允许用户从桌子上的许多物体中挑选（用定位器）一个让机器人拿走的物体，并让机器人带走。

7.3 重新设计两只手臂的机器人模型，使得每只手臂上手指单独控制，从而使得两只手臂能够分别拾取物体。

7.4 增强机器人的动画，使得动画的三视图同步地显示，这其中包括俯视图和“机器人的视野”视图（即机器人在走动所“看到”的视图）。

7.5 设计在图7-27的递归显示遍历过程中添加修剪省略。假设结构的 MC 范围存储在该结构的记录中。注意，将 MC 的范围包围盒转换成 NPC 的范围包围盒，从而满足范围包围盒必须与主轴对齐的要求。

7.6 更新递归显示遍历程序，使得它维护为每个结构保存的 MC 的范围信息。假定在编辑完结构S之后，S结构记录中的extentObsolete布尔域被设置。同时也假定给定任何一个图元，函数返回的该图元的NPC 范围总是有效的。

7.7 给定线段的 NPC 的端点坐标和定位器测量值，设计分析计算候选线段的命中点的算法。

7.8 设计分析计算候选填充区域的命中点的算法。

7.9 采用伪代码，设计只支持线框模式的递归关联拾取的遍历过程。

7.10 实现函数 PtNearLineSegment 并分析其在关联拾取中的作用。为了成为候选物体，线段图像必须在定位器位置的 P 像素宽度范围内。

7.11 应用裁剪方法实现函数 PtNearLineSegment。优化修改（图3-45中的）Liang-Barsky 裁剪算法，因为被裁剪掉的线段是不需要的——只有一个布尔值将会返回。

7.12 完成用在关联拾取中的函数 PtInPolygon。处理当射线通过顶点或者射线与边重合时的特殊情况。参见 [PREP85] 和 [FORR85] 中所讨论的本问题的微妙之处。

7.13 设计拾取用户界面，使得用户可以指定想要的层次结构的层。针对机器人模型通过写一个使用户可以高亮机器人的各个部分（从独立的部分到整个子系统）的应用程序来完成并检测你设计的界面。

346

第8章 输入设备、交互技术与交互任务

本章是关于设计与实现图形人机界面的三章中的第一章。由于现在硬件和软件的价格已经低得足以给办公室和家庭提供高效的计算能力，因此高质量的用户界面已成为向各类用户提供计算能力的“最后一个有待开发地带”。正如近来软件工程由软件结构发展为软件活动（软件过程）一样，用户界面工程这一新的领域也正在形成用户界面原理和设计方法学。

在对计算机进行正式研究的过程中，人们近来才对人机界面的质量产生兴趣。一直到20世纪80年代早期，人们研究重点还是如何优化两种珍贵的硬件资源：计算机时间与内存。当时，程序的效率是最高目标。今天随着硬件价格的急剧下降，以及面向图形的个人计算环境功能日益增强（如第1章中所讨论），我们已经能够承担优化用户效率而不是计算机效率的费用。因此，尽管本章提出的许多观点需额外的CPU周期和内存空间，但在用户生产率和满意度方面的潜在回报会大大超过这些资源所需的合理附加费用。

用户界面的质量常常决定了一个用户是喜爱还是轻视一个系统，系统的设计者受到赞扬还是指责，系统在市场上是成功还是失败。实际上，在空中交通控制以及核电厂监控这些要求严格的应用领域，差的用户界面甚至会导致大规模灾难性事件。

桌面隐喻用户界面（具有窗口、图标和下拉菜单，而且它们都充分利用了光栅图形）由于易于学习并且几乎不需任何打字技巧而备受欢迎。这些系统的大多数用户都不是计算机编程人员，与许多编程人员喜好大相径庭，他们不喜欢老式的、不易学习的、面向键盘的命令语言界面。一个交互式图形应用的设计者必须对用户的需求十分敏感，即用户需要容易学习且功能强大的界面。

另一方面，将来参加工作的许多用户已经在家里或学校通过使用计算机而具有了一定的计算机文化背景，他们对计算机的熟悉程度将会提高。因而对于工作场所的系统开发人员可以假定用户已经知道了一般的计算机概念；而一些教育软件和游戏软件的开发人员，仍将继续为不熟悉计算机的用户进行软件设计。

本章讨论用户界面的基本要素：输入设备、交互技术以及交互任务。输入设备已经在第2章和第4章中介绍过，这里将详细阐述各种输入设备的用法。交互技术是指使用输入设备向计算机中输入信息的方法，它是设计精巧用户界面的基本部件，而交互任务对采用交互技术输入的信息按基本类型进行分类。

第9章讨论在将多个基本部件融合成一个完整的用户界面设计时所涉及到的问题。重点在于自顶向下的设计方法；首先明确设计目标，然后通过逐步细化过程进行开发。该章讨论了所见即所得(WYSIWYG)、命令语言以及直接操纵等各种对话风格的优缺点，叙述了影响用户界面的窗口管理器术语，并以正反两方面的各种例子，描述并列举了设计原则以及界面设计的注意事项。第8章和第9章中的很多论点在以下文献进行了更加深入的讨论，请参阅Baecker和Buxton[BAEC87]，Hutchins、Hollan和Norman[HUTC86]，Mayhew[MAYH90]，Norman[NORM88]，Rubenstein和Hersh[RUBE84]，Shneiderman[SHNE87]所著的教科书，Salvendy的参考书[SALV87]以及Foley、Wallace和Chan的综述[FOLE84]。

第8章和第9章中的许多例子都源自苹果计算机公司的Macintosh系列微机的用户界面。虽然Macintosh的用户界面并不很完美，但它与以往普遍采用的界面相比有很大的改进。

Macintosh于20世纪80年代早期开发，其前身主要是Xerox公司Palo Alto研究中心(PARC)在20世纪70年代中期开展的探索性工作，后来被一些系统（如Microsoft Windows、Presentation Manager、NeXT公司的NeXT Step、the Commodore Amiga,、Digital Research的GEM以及其他很多系统）加以模仿并在一定程度上加以扩展。（本书的很多内容在Macintosh系列微机上使用Microsoft Word编写，许多图是在Macintosh上使用Freehand画成。）

第10章介绍用户界面软件。图形用户界面应设计成易于学习及快速使用，但实现起来就是另外一回事了。掌握适当的软件工具非常关键。该章先讨论SRGP和SPHIGS的输入处理能力，然后讨论更普遍的、功能更强的输入处理能力。窗口管理器是很多高质量的用户界面的关键性组成部分，该章叙述了窗口管理器的内部结构和实现策略。该章最后介绍了用户界面管理系统的主要概
348 念。用户界面管理系统（UIMS）为界面的设计者和实现者提供了一种手段，以便快速地进行开发、试验、修改他们的界面构想，从而降低用户界面开发过程的测试与细化主要步骤的成本。

本章讨论的重点是输入设备，即用户向计算机系统输入信息时使用的硬件部分。在第4章中我们已经讨论过很多这类设备，本章介绍其他设备，并讨论选用某种设备而不用另一种的原因。8.1.6节描述面向三维交互的输入设备。我们继续使用定位设备、键盘设备、选择设备、定值设备和拾取设备等逻辑设备类型，这些类型是由SRGP、SPHIGS和其他与设备无关的图形软件包中所采用的。

*交互任务*是用户送到一个信息单元的输入项。四种基本的交互任务分别是*定位*、*文本输入*、*选择*和*定量*。定位交互任务中的信息输入单位当然是位置。与之类似，文本输入任务产生一个文本字符串，选择任务产生一个对象标识，定量任务产生一个数值。许多不同的*交互技术*可用于一项给定的交互任务，例如，一项选择任务可以通过使用鼠标从菜单中选择菜单项来实现，也可以使用键盘输入选择名、按功能键或者使用语音识别设备来实现。类似地，单个设备也可以用于多项不同任务，如鼠标经常用来定位和选择。

交互任务不同于前面几章讨论的逻辑输入设备。交互任务由“用户需要完成什么”来定义，而逻辑输入设备是按任务如何由应用程序和图形软件包来实现进行分类。交互任务是以用户为中心的，而逻辑输入设备是一个针对编程人员和图形软件包的概念。

如果与自然语言做一个类比，那么使用输入设备进行的单个动作类似于字母表中组成单词的单个字母。构成一个交互技术的输入设备动作序列类似于构成一个单词的字母序列。单词是有含义的单位；如同几个交互技术可以用来实现同一项交互任务一样，几个同义词也可表达同一种含义。用户输入的所有信息单位的含义可以归结到四种基本交互任务之一，这正如单词可以划分为动词、名词或形容词等类型一样。一个交互式对话由交互任务序列组成，这正如一个句子是由单词序列组成一样。

8.1 交互硬件

我们在这一节介绍4.6节中没有提及的一些交互设备，详细阐述这些设备是如何工作的，并且对各种设备的优缺点加以讨论。本节按4.6节的逻辑设备分类进行讨论，它可作为4.6节内容的更详细延伸。

各种交互设备的优缺点可以分三级讨论：设备级、任务级和对话级（即几种交互任务序列）。*设备级*以硬件本身的特点为中心，不涉及由软件控制的设备使用方面。例如，在设备级，我们会注意到一种形状的鼠标手握起来可能比另一种形状的更加舒服，数据输入板比游戏杆占用更
349 多的空间。

在任务级，我们可以针对同一项任务使用不同设备，对交互技术进行比较。从而可能得出以下结论：有经验的用户通过功能键或键盘输入命令比通过菜单选择更快；或者用户使用鼠标来拾取可见的对象比使用游戏杆或者光标控制键更快。

在对话级，我们考虑的不仅是一个个独立的交互任务，而且考虑这些任务组成的序列。手在设备之间移动时需要花费时间：虽然用鼠标完成定位任务通常比用光标控制键更快，但是如果用户的双手已经位于键盘之上，并且需要继续在光标定位以后按顺序用键盘完成下一个任务，这时用光标控制键定位可能比鼠标更快。对话级的内容将在第9章中讨论，在那里，我们还将论述如何由本章介绍的各种基本部件构造完整的用户界面。在考虑各种设备时，始终在头脑中记住这三个级别可以使我们避免很多混淆。

本节所讨论的设备级，重点关注的是设备的足迹（footprint，指其占用的工作区域）、操作员疲劳度以及设备分辨率。设备的其他一些重要特性（如费用、可靠性、可维护性等）随着技术的发展而变化太快，这里就不讨论了。设备与计算机连接的最普遍方法是串行异步RS-232终端接口，它通常能使接口设计相当简单，这些细节本节将略去不讲。

8.1.1 定位设备

按三种相互独立的特性来给定位设备分类非常有用：绝对设备或相对设备、直接设备或间接设备、离散设备或连续设备。

绝对设备（如数据输入板、触摸板）有一个参考区域或原点，并报告相对于原点的目标位置。相对设备（如鼠标、跟踪球和控制速度的游戏杆）没有绝对原点，只报告相对于原先位置的目标变化情况。相对设备可用于描述位置的大幅度变化；用户可以沿桌面移动鼠标，向上举起将它放回到最初的开始位置，并再次移动。数据输入板也可编程用做相对设备：指示笔从“远”状态移到“近”状态（即靠近输入板）以后读入第一个(x, y)坐标，用其后读入的所有坐标减去该坐标，仅产生x和y坐标的改变量，这些改变量可加到原先的(x, y)位置。这一过程一直延续到指示笔回复到“远”状态。

相对设备不宜用于数字化绘图，而绝对设备可以。相对设备的优点在于应用程序可以将位于屏幕上任何位置的光标重定位。

用户可以使用直接设备（如光笔、触摸屏）用手指或其他代用品直接指点屏幕，而使用间接设备（如输入板、鼠标或游戏杆）用户则是用不接触屏幕的设备在屏幕上移动光标。对于后一种情况必须学会眼-手协调的新形式；不过，家庭和游艺厅中计算机游戏的激增正在创造着一种环境，在此环境中，很多不常用计算机的用户已经学会了此类技巧。但是，直接指点会引起手臂疲劳，尤其是对不常用计算机的用户。

350

连续设备是平滑的手动可以产生平滑的光标移动的设备。输入板、游戏杆和鼠标都是连续设备，而光标控制键则是离散设备。连续设备最典型的特点是允许比离散设备更自然、更容易、更快速的光标移动，大多数连续设备比光标控制键更容易在任意方向上移动。

使用连续设备进行光标定位的速度受控制-显示比率的影响，控制-显示比率一般称为C/D率[CHAP72]，它是手的移动（控制）与光标移动（显示）之间的比。该比率大有益于精确定位，但这种快速移动过程很枯燥；比率小有益于加快光标移动速度但不利于精确定位。所幸的是，对于一个相对定位设备而言，该比率不必固定不变，而是可以随着控制-移动速度的变化而灵活改变。快速移动表示用户正在做粗略的手动，所以使用小的比率；而移动速度降低时，C/D率则相应增大。C/D率的改变使得用户可以用鼠标在一个15英寸的屏幕上精确地给光标定位而甚至不必移动一下自己的手腕。对于间接离散设备（光标控制键）而言，也存在相似的技

术：每个单位时间光标移动的距离随着按键时间的延长而增加。

在没有支撑点的情况下，如果胳膊伸向屏幕，那么使用直接设备进行精确定位是相当困难的。请试一下用这种姿势在黑板上写你的名字，然后与你正常书写的名字比较。如果将屏幕的放置角度调整到接近水平的位置，可以减轻这个问题。另一方面，间接设备允许手后掌放在支撑点上，这样可以更有效地使用手指的精细运动控制能力。不过，在绘图时并不是所有连续间接设备都同样令人满意的。试一试分别用游戏杆、鼠标、输入板上的指示笔写你的名字。用指示笔写得最快，结果也最好。

其他有趣的定位设备包括Versatron的脚踏鼠标[VERS84]，这种鼠标放在地板上不动，用户将拇趾球放在设备上，脚后跟放在地板上，用从左向右和从前往后的运动来控制脚踏鼠标。试验性的鼹鼠标(mole)是一个有支点的带集成开关的搁脚板[PEAR86；PEAR88]，与脚踏鼠标一样，它们都不需要用手操作。Personics的头鼠标[PERS85]用一种固定在头部、由三个麦克风组成的装置来测量与声源的距离，将头部的小旋转运动转化为光标移动。视线跟踪仪能判定眼睛所指的位置，从而能使光标移动或者选取所指的目标[BOLT80；BOLT84；WARE87]。这些设备与传统设备相比经常是精确度不高而且相当昂贵，因此一般仅用于不用手操作的应用程序。8.1.6节讨论的三维定位设备也可以用于二维定位。

8.1.2 键盘设备

众所周知的QWERTY键盘已经伴随我们很多年了，富有讽刺性是这种键盘最初的设计目的是为了减慢打字员的打字速度，这样打字机的印字锤就不会那么容易卡住了。研究表明更新
351 型的Dvořák键盘[DVOR43]比QWERTY键盘更快一些[GREE87]，这种键盘把元音和其他高频字符放在手指最容易击打到的位置，但没有被广泛接受。很多非专业打字用户有时使用按字母顺序组织的键盘，但越来越多的人正在使用QWERTY键盘，多项实验表明QWERTY键盘胜过按顺序组织的键盘[HIRS70；MICH71]。

和弦键盘(chord keyboard)有五个键，类似于钢琴键，可以用一只手操作，同时按下一个或多个键“演奏和弦”。用这五个键可以演奏31种不同的和弦。学会使用和弦键盘（以及其他类似的速记员风格的键盘）要比学习QWERTY键盘花的时间更长，但是熟悉的用户可以相当快地击键输入，同时空出一只手来完成其他任务。不过，增加的训练时间意味着这种键盘不适宜代替标准的字母数字型键盘的使用。

其他面向键盘的考虑事项不涉及硬件而是涉及软件设计，如让用户不必同时按下Ctrl键或Shift键而输入频繁使用的标点符号或校正字符，将危险的操作（如删除）分配给远离常用键的那些键等。

8.1.3 定值设备

某些定值设备是有限的，就像收音机上的音量控制旋钮只能转到防止进一步旋转而设置的截止位置。有限的定值设备输入一绝对的值，而一个连续转动的电位器可以向任一方向旋转无限次。设定一个初值，无限的电位器可用于返回一绝对的值，否则，返回值视为相对值。提供某些反馈信息能使用户判定当前确定的是什么样的相对值或绝对值。在定位设备部分讨论的C/D率在使用滑动式电位器和旋转式电位器输入值时也可以使用。

8.1.4 选择设备

功能键是一种常见的选择设备。功能键的位置影响着它们的易用性：固定在阴极射线管(CRT)斜面上的键比键盘上的键或分立装置上的键用起来更困难。脚踏开关可以在用户的双手没有空闲、而又必须经常关闭开关的场合使用，例如，与8.1.1节描述的头鼠标配合使用，脚踏

开关可以方便地提供与单钮鼠标等价的功能。

8.1.5 其他设备

在这一部分内容中，我们讨论一些较为少见、尚处于试验阶段的二维交互设备。语音识别器非常有用，因为它们将用户的双手解放了出来去发挥别的作用。语音识别器将模式识别方法应用于我们说话时产生的波形。典型地，波形被分成一定数量的不同频率的波段，每个波段中波形振幅随时间变化的情况形成模式匹配的基础。可是，模式匹配过程中会产生一些错误，因 352
此尤为重要的是使用识别器的应用程序要提供方便的校正能力。

各种语音识别器在以下方面有所不同：是否识别特定说话人的必须经训练的说话波形；能否识别连续语音，还是个别单词或短语。与说话人无关的识别器只有非常有限的词汇量，一般只包含数字和50~100个单词。一些离数单词识别器在适当训练之后可以识别多达几千个不同单词的词汇量，但如果用户患了感冒，就必须重新训练识别器。识别器的用户必须在每个词之后停顿一些时间以提示系统这个词结束了，停顿时间一般为100~200毫秒，如果单词集合较大还可能更长。也可以用商品化的硬件来实现有限词汇量的连续语音识别这种较为困难的任务，只是费用更高。词汇量越大，越需要更多的人工智能技术利用一连串句子的上下文和语义来消除歧义性。有几个词汇量在20 000个单词以上的系统，能识别诸如“Write Mrs. Wright a letter right now!”之类的句子。

语音合成器产生波形，以一定的真实程度近似人说出来的单词[KAPL85]。最简单的合成器使用音素（phoneme），它是形成单词的基本的声音单元。用这种方法生成的声音不自然，生硬呆板。比较成熟的基于音素的系统增加了抑扬顿挫感。其他系统实际上是回放数字化的语音模式，它们听起来很逼真，但是需要数千字节的存储器来存储数字化的语音。

包括Macintosh和NeXT系列机等多种个人计算机，都有标准的能产生声音和音乐的语音合成器，那么计算机产生语音反馈就相当普遍了。语音最适用于增强而不是代替视觉反馈效果，偶尔用到时效果最好。例如，在培训时可以向学生演示一些过程的图形动画，同时伴有语音描述正在观看的东西。请参阅[SIMP85；SIMP87]进一步了解语音的识别和生成，包括在人机界面中有效应用这类功能的一些指导原则。

声音生成器可以用来生成音乐的音调和其他效果，这样能引起对特定场景的注意，尤其是当用户不在看显示器时。例如，除了屏幕上显示的信息外，可以用两种不同的音调发出“打印机缺纸”或“内存将满”这样的报警信号。将一条已受约束的直线重新定向与另一条直线平行，也可能产生报警的嘟嘟声。

数据输入板已经以多种方式延伸使用。多年以前，Herot和Negroponte使用一种试验性的压力敏感的指示笔[HERO76]：当压力高、绘画速度慢时表示用户正在仔细画线，这种情况下完全按画出的效果记录曲线；压力低、速度快表示正在快速画线，这种情况下仅记录连接两个端点的直线。近期市场上可以买到的输入板[GTCO82]不仅对指示笔的压力敏感而且也对方向敏感。由输入板产生的5个自由度可在多个方面进行创造性应用。例如，Bleser、Sibert及McGee实现的GWPaint系统可以模仿艺术家所用的各种工具，如斜体笔，这些工具对压力和方向都敏感[BLES88a]。图8-1展示了该系统产生的艺术创造力。 353

Green[GREE85]将光学原理应用于开发一种类似输入板的设备，这种设备给予艺术家比压力敏感和倾斜度敏感的输入板更大的自由度。用户可以使用刷子、手或者其他任何便于使用的工具在输入板上作画，位于输入板下方的电视摄像机录下刷子在输入板的任意位置的形状和动作，产生的视频信号加载到光栅显示器的刷新缓冲区。根据所用的电视摄像机，可以达到500或1000个单位的分辨率。

由Buxton及其同事开发的一种处于试验阶段的触摸输入板，可以同时感受多个手指的位置，还能感受每个接触点覆盖的区域[LEE85a]。该设备主要是一种触摸板，但是作为一个输入板在工作平面上使用，而不是作为一个触摸板固定在屏幕上方。该设备用途相当广泛[BUXT85]。不同大小的手指压力与接触点覆盖区域相互关联，用于发送用户命令：轻微的压力引起光标出现并且跟踪手指运动；增大压力与在鼠标或圆盘上按下按钮相似，用于开始一个反馈，如拖动对象；减小压力则停止拖动。



图8-1 数字2，一幅Jasper Johns风格的画面，由Teresa Bleser使用对压力和倾斜度都很敏感的GTCO输入板以GWPaint程序所画。（经乔治·华盛顿大学的T.Bleser允许使用。）

另一种得到不只是位置信息的方法是，使用几根可测量应力的金属支撑杆将一个触摸板悬挂在显示器前方[HERO78，MINS84]，施加于触摸板上的压力转换成可测量的应力，也可以通过将金属杆调整到适当方向测量出施压和扭动的方向。测量值可以用来使显示的物体旋转、缩放等。

输入设备价格的下降以及计算机适用范围的增大，导致了各种新颖交互设备的不断引入。20世纪60年代Douglas Engelbart发明了鼠标，在近20年以后才开始流行[PERR89]。下一代鼠标会是什么样子目前还不清楚，但是我们希望它的酝酿时间更短。

8.1.6 三维交互设备

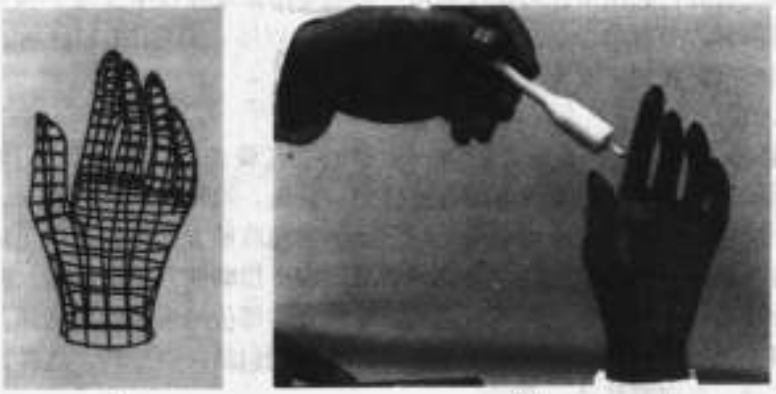
一些二维交互设备很容易扩展到三维交互应用中，游戏杆可以有一个手柄作为第三个维数（图4-38），跟踪球可以做成除了能感受到绕两个水平轴的旋转，还可以感受到绕垂直轴的旋转。但是在这两种情况下，利用设备进行手动与在三维空间中做相应运动，这两者之间并没有什么直接联系。

空间球（参见彩图I-14）是一个装有应力仪的坚硬球体，用户向任意方向推或拉该球体，可提供三维平移和定向。这种情况下，至少运动的方向与用户移动硬球的目的相对应，尽管手实际上不动。

另一方面，有很多设备可以记录三维的手部运动。由Michael Noll开发的尚处于试验阶段的Noll盒允许一个12英寸立方体中的圆球运动，并通过与电位器相连的滑杆传感。Polhemus 3SPACE三维位置和方向传感器采用三根发送天线与三根接收天线之间的电磁耦合。发送器三个天线的线圈相互垂直，形成一个笛卡儿坐标系，并依次产生脉冲。接收器有三根类似安装的接收天线，每次发送线圈发脉冲时，在每个接收线圈中产生电流。电流强度大小不仅依赖于接收设备和发送设备之间的距离，也依赖于发送设备线圈与接收设备线圈之间的相对方向。由三个连续脉冲产生的九个电流值组合，用于计算接收设备的三维位置和方向。图8-2展示了该设备的一项常用用途：数字化三维物体。

数据手套（DataGlove）记录了手的位置、方向和手指的移动。如图8-3所示，它是带有小而轻的传感器的一个数据手套。每个传感器是一条不长的光纤电缆，其一端有一发光二极管，另一端有一个光敏晶体管。在对弯曲敏感的区域电缆的表面是不平滑的。当光纤折曲时，某些

LED的光线丢失，因而光敏晶体管就收到很少光线。另外,手套上的Polhemus位置和方向传感器将记录手的移动。戴了数据手套后，用户能抓取目标，移动和旋转目标，再释放目标。因此数据手套提供了十分自然的三维交互[ZIMM87]。彩图I-15说明这一概念。 355



a) b)

图8-2 a) 线框显示结果，b) 用于数字化三维物体的Polhemus三维位置传感器。(3Space数字化仪，由Colchester公司的Polhemus提供。)

人们做了相当多的努力来创造一种常称之为人工现实或虚拟现实的环境，这是完全由计算机生成的环境，具有现实的外观、行为和交互技术[FOLE87]。用户配戴一个安装在头部的立体显示器来显示左、右眼视图，头部的Polhemus传感器使头部位置与方向的变化引起立体显示器显示内容的变化，数据手套允许三维交互，麦克风用于发布语音命令。彩图I-16显示了各种设备的组合情况。



图8-3 VPL数据手套，展示用于传感手指运动的光纤电缆和Polhemus位置与方向传感器。(摘自J.Foley的《高级计算界面》，SCIENTIFIC AMERICAN公司1987年出版，版权所有。)

其他几种技术可用于记录三维位置。4.6.1节讨论的声音输入板技术可以扩展到三维空间中，创造一种声音指示笔，其中一种方法使用了三个互相垂直的麦克风。握在手中的笔每秒钟振动20~40次，声音到达每个麦克风的时间决定了指示笔所在的三个圆柱体的半径，因此指示笔的位置可以通过三个圆柱体的相交位置来计算出。一种类似的方法使用三个或四个标准麦克风，这里，指示笔的位置由球体相交位置计算出，球体以麦克风为中心，球的半径由声音到达每个麦克风所需时间决定。

所有这些系统工作在相当小的体积内——8到27立方英尺，光传感器的移动自由度甚至更大[BURT74；FUCH77a]。发光二极管放在用户身上（要么在一点上，如手指尖，要么遍布全身测量身体运动情况），光传感器安装在用户工作的半明半暗小房间的角落高处，依次增强发光二极管的亮度。传感器可以确定发光二极管所处的平面，这样发光二极管的位置就在三个平面

相交之处。（通常还使用第四个传感器，以防其中一个传感器看不见发光二极管。）指尖及其他点上的小反射器可以代替发光二极管，传感器拾取反射光而不拾取发光二极管发出的光。

Krueger[KRUE83]开发的一种传感器可以记录二维环境中手和手指的运动。用一个电视摄像机录下手的运动情况，利用增强对比度和边缘检测等图像处理技术发现手和手指的轮廓。不同的手指位置被解释为各种命令，用户可以抓取和操纵物体，如彩图I-17所示，这项技术可以通过使用多个照相机扩展到三维环境中。

8.1.7 设备级人的因素

从人的因素观点来看，并不是所有同一类型的交互设备都完全等价（详细内容请参见[BUXT86]）。例如，鼠标在一些重要方面有所区别，首先，物理形状有所不同，从半球形到加长的、低轮廓矩形都有。按钮位置也不同，位于鼠标侧面或前方的按钮可以使鼠标在按下按钮时移动一点，位于鼠标顶部的按钮就没有这种效果。用手指朝前方抓取鼠标，通过手腕和手指运动可以将鼠标移动一小段距离。但是，感受鼠标位置的部位经常是朝向尾部，几乎不可能进行细微控制。事实上，鼠标在指尖下的微小左向移动可能包括一点旋转，这样，位置传感器所在的鼠标尾部实际上会向右移动一点！

356
~
357

各种键盘在设计参数方面存在很大差异，如键帽的形状，键与键之间的距离、按键所需压力、键的凹陷高度、键跳、听觉反馈效果、键完全下陷时的触感，以及重要的键（如返回键和回车键）的位置与尺寸。参数选得不正确会降低工作效率、增加出错率。回车键做得太小会招致错误，就如同将硬件的重新启动键设置得离其余键太近一样。以上及其他设计参数在[KLEM71；GREE87]中有所讨论，而且已经是最新国际标准讨论的主题。

短游戏杆的柄尖移动距离短，促使采用低C/D率。如果用较长的游戏杆手柄，用户就无法将手掌放在工作台面上，这样就没有稳定的平台做细微调整，因此会影响精确性和速度。

设备的这些不同之处意味着对于一个用户界面设计者来说，仅仅指定一个专门的设备级别还不够，必须定义具体的设备特性。不幸的是，并不是每个用户界面设计者都有条件自由选择设备，通常这种选择是已经确定的。于是设计者只能希望设备设计得比较好，并尽量在软件方面弥补硬件的不足之处。

8.2 基本交互任务

作为一个基本的交互任务，交互系统的用户输入一个在应用环境中具有一定意义的信息单元。这样一个单元应多大？例如，将一个定位设备移动一小段距离，输入的是一个信息单元吗？如果新位置是针对一些应用目的，诸如重定位对象或指定曲线的结束端点，则输入的是一个信息单元。但如果这个重定位动作仅仅是用户将光标移动到菜单项顶部这一连串重定位动作中的一步，则菜单选择才是一个信息单元。

基本交互任务（BIT）是不可分割的，也就是说，如果将基本交互任务分解成更小的信息单元，那些更小的单元本身对于应用将毫无意义。本节讨论BIT，下一节讨论组合交互任务（CIT）。CIT是此处讨论的基本交互任务的集合，如果将BIT视为原子，那么CIT就是分子。

用于交互式图形学基本交互任务的完整集合包括定位、选择、文本输入以及定量。本节描述各种基本交互任务，并讨论针对每种任务的一些交互技术。不过，交互技术非常之多，不可能一一列出，况且我们也无法预料新技术的发展状况，只能尽可能地讨论各种技术的优缺点。请记住，一项专门的交互技术可能在一些场合很好用，但在另一些场合则不好用。

8.2.1 定位交互任务

定位任务包括给应用程序指定一个(x, y)或(x, y, z)坐标位置，实现这一任务常用的交互技术

要么是将屏幕光标移动到希望的位置然后按一下按钮，要么就是在实际的或模拟键盘上键入希望的位置的坐标。定位设备可以是直接设备或间接设备、连续设备或离散设备，也可以是绝对设备或相对设备。此外，可以在键盘上显式地键入移动光标命令，如向上、向左等等，或者可以将相同的命令发到语音识别装置。而且，各种技术可以结合使用——控制光标的鼠标可以用来近似定位，箭头键可以用来将光标移动一个屏幕单位进行精确定位。 358

有很多常用的术语在任一种交互技术中都要涉及，我们首先来讨论这些术语，并详细介绍具体的定位技术。

1. 坐标系

定位任务中一个重要术语就是坐标系，反馈信息就是定义于其中的。如果将一个定位设备向右移动去拖动一个对象，该对象应该移向哪个方向呢?至少有三种可能：对象在屏幕坐标系中沿着x坐标增大的方向移动，在世界坐标中沿着x坐标增大的方向移动，或者在对象自己的坐标系中沿着x坐标增大的方向移动。

第一种可能性是正确的选择，即沿着屏幕坐标系中x坐标增大的方向移动。对于后两种情况，增大的x方向一般不一定沿着屏幕坐标系的x轴方向。例如，如果观察变换包括一次180°的旋转，则世界坐标系的x轴会转到与屏幕坐标系的x轴相反的方向。这样的话，向右移动定位设备会导致对象向左移动。请试一下将鼠标旋转180°，用这类反馈信息定位！这样的系统无疑会严重破坏刺激-响应兼容性（S-R兼容性）的人的因素的原则，即系统对用户操作做出的响应必须是在相同方向上，而且响应的数量应该与操作成正比。如果相对于屏幕将数据输入板旋转一定的角度，也会产生相似的问题。

2. 分辨率

定位任务所需的分辨率可以从几百个单位变化到几百万个单位。显然，从键盘输入一对(x, y)值能提供无限高的分辨率：键入的数字串长度完全依照需要而定。光标移动技术能获得多高的分辨率呢?输入板、鼠标等的分辨率典型地能达到500到2000个可分辨的显示设备单位。通过进行窗口-视口的变换来放大客观世界的一部分，可以使一个单位的屏幕分辨率对应于任意小的单位世界坐标系分辨率。

触摸板提出了其他一些关于分辨率的有趣问题。有些触摸板精确到1000个单位，但是用户的手指大约0.5英寸宽，如何让手指达到这样高的精确度呢？用手指触摸到的起始点作为最终位置是不合适的，用户需通过移动或摇晃手指在触摸板上来回拖动光标。由于手指使所指示的精确位置变得模糊不清，光标臂（大小）应做得比正常状态略长，或者光标会偏离实际的接触点。通过实验发现，拖动偏移光标虽然慢一些，但比用起始接触点更加精确[POTT88]。通常情况下不推荐使用触摸板完成频繁的高分辨率定位任务。 359

3. 网格

许多定位任务中的一个重要的视觉辅助手段是在工作区域上附加一个网格（精度可能很低），帮助排列位置或对象。使图元的端点落在网格上也很有用，就好像每个网格点都被一个重力区域所包围。网格化能以一种清晰的外观帮助用户生成图形，为增强网格效果，应用程序只需将定位坐标四舍五入到最近的网格交叉点（有些情况下，仅当坐标点已经靠近网格交叉点时这样做）。网格通常应用于世界坐标系。显然网格经常是规则分布且横跨整个显示器的，但不规则网格、不同区域的不同网格以及旋转网格在创建图形和插图时都很有用[BIER86a；FEIN82a]。

4. 反馈

有两类定位任务：空间定位任务和语言定位任务。在空间定位任务中，用户知道目标位置

在什么地方，目标位置在空间上与附近的元素有联系，例如在两个矩形之间画一条直线或者将第三个对象放在两个对象中间。在语言定位任务中，用户知道位置(x, y)坐标的数值。对于空间定位任务，用户需要反映屏幕上实际位置的反馈信息；对于后一种任务，需要反馈位置的坐标。如果提供了错误的反馈形式，用户必须在心里将一种形式转换成另一种形式，两种反馈形式都可以通过既显示光标又显示其坐标值来解决，如图8-4所示。

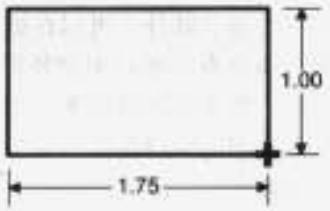


图8-4 关于构造对象尺寸的数字反馈信息。随着光标（+）移动改变对象的高度和宽度，这样用户可以将对象调整到想要的大小

5. 方向选择

一些定位设备不允许在任意方向上都能运动，例如，有些游戏杆和控制开关对离开主轴的运动比对在主轴上的运动施以更大的阻力。如果定位任务本身通常限制为水平和垂直上的运动，这种约束是有用的。

6. 学习时间

用间接的方法学习眼手的协调，与学习驾驶汽车的过程实质是一样的。一般需考虑学习时间，但它只是一个较为次要的因素。Card及其同事[CARD78]对鼠标和游戏杆进行过研究发现，训练改善了错误率与速度，甚至新手的成绩都相当好。例如，经过广泛训练后用鼠标进行选择的时间（包括移动光标至目标，按下按钮）从2.2秒下降到1.7秒。不过，一些用户发现间接协调非常困难，经过具体训练后就好多了。

一种特定的定位任务是连续定位，它用一连串位置定义一条曲线。可用一系列相连的非常
360 短的直线段来逼近定位器走过的路径，如图8-5所示。为了保持光滑的外观，曲率半径小的地方可以用更多条直线段逼近，或者沿光标轨迹显示一个个单独的点，或者沿各个点画出更高阶的曲线（见第11章）。

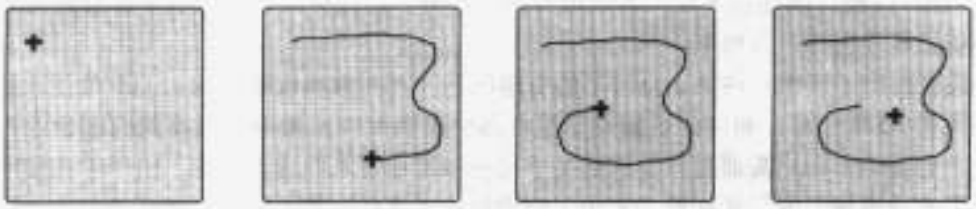


图8-5 连续绘图

用指示笔进行精确的连续定位比用鼠标更容易，因为指示笔可以用手指肌肉精确控制，而鼠标主要由手腕肌肉控制。出于同样原因，很难用鼠标将绘图数字化，它也缺少一个绝对参考区域和十字准线。另一方面，鼠标只需要很小的桌面空间并且比输入板便宜。

8.2.2 选择交互任务——大小可变的选项集合

选择任务就是从一个选项集合中选取一个元素，典型的选项集合包括命令、属性值、对象类和对象实例。例如，一个典型画图程序中的线型菜单是属性值的集合，而这种程序中的对象类型（直线、圆、矩形、文本等等）菜单是对象类的集合。有些交互技术可用于对这四种选项集合中的任一种进行选择，而其他交互技术则通用性较小。例如，不论选项集合是什么类型，

点击一个集合元素的可视化表示，就可以选取该元素。但另一方面，虽然功能键非常适用于从命令集合、对象类集合或属性集合中选取元素，但难以在绘图时将一个个独立的功能键分配给每一个对象实例，因为该选项集合的大小可变，而且经常很大（超过可用功能键的数目），同时由于用户创建并删除对象，使对象集合变更非常快。

我们使用术语（相对）固定大小的选项集合和大小可变的选项集合来区分选项集合。前者包括命令、属性、对象类选项集合，后者包括对象实例选项集合。“相对”是指这些选项集合都可能随着新的命令、属性、对象类（如绘图系统中的各种符号）的定义而发生变化，但是集合大小变化并不频繁，而且通常不会变化太大。另一方面，大小可变的选项集合可能变化得相当大，并且变化频繁。 361

本节讨论专门适用于变化可能性很大的大小可变的选项集合的各种技术，包括通过名字选择与通过定位选择技术。下一节讨论专门适用于（相对）固定大小的选项集合的各种技术，除了复杂应用中大的（但相对固定大小的）命令集合外，这些选项集合一般比较小。将要讨论的技术包括键入或说出名字、缩写词及其他表示集合元素的代码；按下与集合元素相关的功能键等（类似于在键盘上键入一个字符）；在菜单中点击集合元素的可视化表示（文本表示或图形表示）；循环遍历集合直到想要的元素被显示出来；用连续定位设备做特殊运动。

1. 通过名字选择对象

用户可以键入选项的名字。这种想法很简单，但是，如果用户不知道对象的名字怎么办？或者像经常容易发生的那样，同时显示几百个对象怎么办？又或者用户没有理由知道对象名字怎么办?尽管如此，这项技术对多种情况都有用。首先，如果用户极有可能知道各种对象的名字，就像一个舰队的指挥官知道各艘舰船的名字一样，那么用名字引用对象就很明智，并且比定位更快，尤其是当用户可能需要翻滚显示器屏幕查看想要的对象时更明显。其次，如果显示器太拥挤杂乱以致于难以通过定位来选取，以及不能进行缩放（可能由于此图形硬件不支持缩放功能并且软件缩放太慢），这时就只能通过名字选取对象。如果只是因为显示器太拥挤，那么用命令将对象名切换为可用或不可用就非常有用。

如果按照元素的含义给选项集合中的各元素命名，则采用键盘输入可允许我们使用通配符或禁忌字符，而做出多种选择。通过名字选择最适合于经验丰富、经常进行输入的用户，而不适合偶而进行输入的用户。

如果要从键盘键入名字，则每次击键之后立即显示一条有用的反馈信息，将选项集合中与目前键入的字符序列相匹配的名字列表（或者部分列表，如果全部列表太长的话）显示出来。如果用户已经回想起前几个字符，这样显示出来名字列表有助于用户记得刚才是怎样拼写名字的。一旦键入一个惟一匹配的名字，就会自动在列表中高亮度显示正确的名字，或者自动完成名字尚未键入的部分。这种技术称为自动完成。新用户有时难以适应该技术，因而需要慎重使用。另一种用于输入名字的策略是拼写校正（有时称Do What I Mean 或 DWIM），如果键入的名字与系统中已知的名字不匹配，其他与键入名相近的名字会作为可选名字呈现给用户。判断相似性可能与查找单字符错误一样简单，也可能有多个字符错误或者缺少字符。

用户可以不用击键而是借助语音识别器说出一个名字、缩略词或者代码。语音输入是从数据中区分命令的一种简单途径：通过语音输入命令，通过键盘或其他方式输入数据。在键盘环境中，语音输入消除了用特殊字符或其他特殊方式区分数据与命令的必要性。

2. 通过定位选择对象

8.2节引言部分提到的任何一种定位技术都可用于选择对象。对需点击的期望对象，先定

位然后指示（一般通过按下按钮）。但是，如果该对象像第7章中的机器人那样具有多层结构又怎么办呢？如果光标停在机器人的手上方，就弄不清楚用户需点击手、胳膊还是整个机器人。
362 可以使用Select_robot（选择机器人）和Select_arm（选择胳膊）之类的命令指明结构的层次。另一方面，如果用户工作所处的层次不经常改变，就可以用一条单独的命令，如Set_selection_level（设置选择层次），来修改结构的层次，这样工作速度更快。

如果系统设计者不知道结构的层次数目并且数目可能非常大（如绘图系统中，符号由图元和其他符号组成），这时就需要采用另外一种方法。至少需要两条用户命令：Up_hierarchy（向上一层）和Down_hierarchy（向下一层）。用户选择某对象时，系统高亮度显示最低层的可见对象。如果正是想要的对象，用户继续操作。如果不是，用户发出第一条命令：Up_hierarchy。高亮度显示的整个第一层对象包含已检查对象。如果这不是用户想要的，他就再一次向上遍历，图像的大部分仍是高亮显示。如果正处于最高一层，可以用Down_hierarchy命令沿相反方向向下选择对象。此外，Return_to_lowest_level（转到最低层）命令在深层次结构中非常有用。可以在另一个窗口的层次图中显示当前的选择位于哪一层。图8-6所示的状态图显示了层次选择的一种方法。另外，使用一条Move_up_hierarchy（移动到前一个层次）命令能够在到达节点后跳回到最初所选的叶节点。

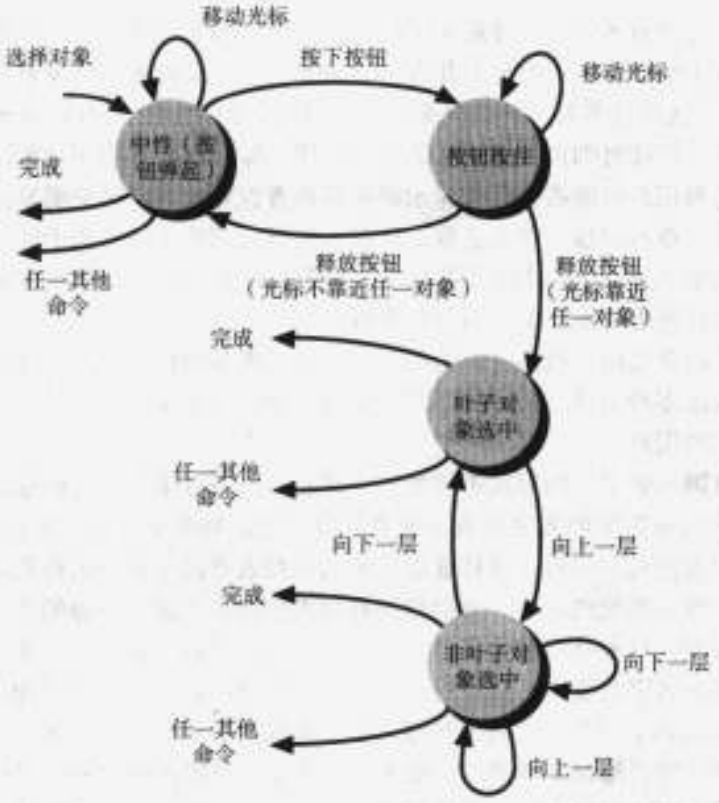


图8-6 用于具有任意层次对象~选择技术的状态图。Up和Down命令用于向上、向下层次移动，"Leaf object selected"状态Down_hierarchy命令不可用。用户通过用光标定位对象、按下按钮然后释放来选择对象

一些文本编辑器使用字符、词、句子、段落四级结构。如在Xerox Star文本编辑器中，用户通过将屏幕光标定位在字符之上并且单击鼠标的选择按钮来选择一个字符。如果不选择字符

而是选择词，则快速连击两次按钮，依此类推通过快速双击选择上一层的对象。

8.2.3 选择交互任务——相对固定大小的选项集合

菜单选择是一种从相对固定大小的选项集合中进行选择的最常用技术，这里讨论几个菜单设计中的关键因素。

1. 菜单排序

菜单中各元素可以按很多种不同的顺序来组织，包括按字母排序、按功能进行逻辑分组、最常用的优先、最重要的优先、最大的优先、最近创建或修改过的优先。这些排序方法也可以结合使用，按功能分组的菜单可以在组内按字母顺序排序，而功能组本身再按使用频度排序。图8-7举例说明了几种可能的组织方式。从一个菜单到另一个菜单保持结构上的一致性非常有用，因此在一个应用程序的所有菜单中采用统一策略很重要。一些研究人员发现，按功能排序最有帮助，而很多菜单的结构也反映了这一点。

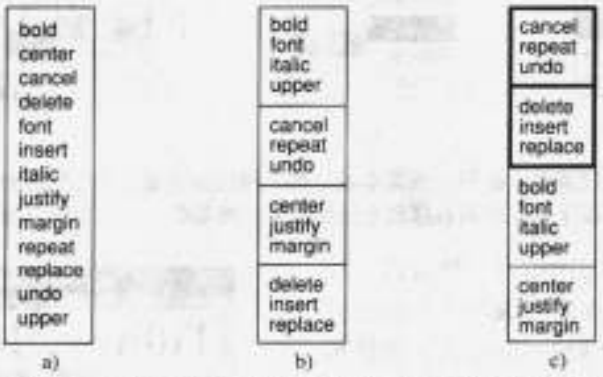


图8-7 三种菜单组织方式。a) 按字母顺序排列的菜单；b) 按功能分组的菜单，每组内部按字母顺序排列，组之间也按字母顺序排列；c) 该菜单将几个不同应用程序中共有的命令列在菜单顶部以保持与其他应用程序的菜单的一致性，这些命令带有粗边。各菜单项是Card菜单排序实验中使用的一些菜单项[CARD82]

2. 单层与多层设计

如果选项集合太大一次不能全部显示，就需要采用这种最基本的菜单设计方法。这样的菜单被细分成按逻辑结构组织的层次，或者将选项的线性序列分页显示或滚动显示。很多窗口管理器使用的滚动条允许以一种简洁的方式显示所有相关的滚动命令和分页命令，还可以提供一种快速定位滚动命令的面向键盘的方式。例如，可以用箭头键滚动窗口，也可将Shift键与箭头键组合使用，在可见窗口内部移动选项，如图8-8所示。极端情况下，窗口大小可以缩小到只有一个菜单项，产生如图8-9所示的只有一个槽的投币机类型的菜单。

363
↓
364



图8-8 滚动窗口内部的菜单。用户通过选择向上箭头键和向下箭头键或者通过拖动滚动条中的方块来控制菜单项的上下滚动

如果使用分层菜单，用户首先要从最高层的选项集合中选择，然后出现第二级选项集合，重复这一过程直到选择分层菜单树的叶节点（即选项集合本身的一个元素）。在进行分层对象选择时，需要提供导航机制，以便如果选择了不正确的子树，

用户可以向上退回一个层次，同时需要提供视觉反馈信息以使用户能意识到所处的层次。

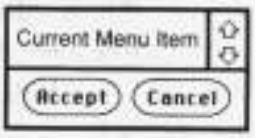


图8-9 一个菜单选择小窗口。一次只显示一个菜单项，滚动箭头用来改变当前菜单项，当选中Accept按钮时选取当前菜单项

有多种方式显示菜单层次。随着菜单选项的一层层展开，由下一级菜单完全取代上一级菜单固然可以，但是不利于用户了解所处的菜单层次。图8-10描述的级联式分层菜单更具吸引力，每个菜单都必须充分显示，使用户可以看见完整的高亮度显示的选择路径，同时必须采用一些方法来说明每个菜单项是一个叶节点还是下一级菜单的名字（图中的向右箭头充当这一角色）。另一种菜单分层方式是只显示在菜单各层中遍历时所选的各级菜单项的名字，以及当前层次中的所有选项。

365



图8-10 一个弹出式分层菜单。a) 第一级菜单响应按下按钮操作显示出光标所处位置，可以上下移动光标选择所需字体；b) 向右移动光标弹出下一级菜单；c) 重复该过程弹出第三级菜单

面板式分层是描述层次的另一种方式，如图8-11所示。这种分层方式比级联式占用的空间更多一些。如果层次结构不是太大，也能显示一个详细描述整个分层情况的层次树。



图8-11 一个分层选择菜单。最左边一栏代表最高一层菜单，该栏中所选项的全部子选项显示在下一栏中，依此类推。如果没有被选项，则右边右栏为空。（NeXT公司授权，©1989 NeXT, Inc.。）

设计分层菜单时通常要考虑菜单的深度与广度。Snowberry等人[SNOW83]通过实验发现，使用选择层次较少而选项范围更广的菜单能改进选择时间和准确度。Landauer和Nachbar[LAND85]及其他研究人员也报告过类似的结果。但是，这些研究结果并不能推广到缺乏自然性及可理解性结构的分层菜单。

366

分层菜单选择几乎总需要一种相关的键盘或功能键加速技术以提高熟练用户（即所谓的"高手"）的选择速度。如果分层树的每个节点都有一个惟一的名字，要做到这一点很容易，用户可以直接输入名字。如果用户忘记了名字，菜单系统还提供备份。如果在层次的每一级内部保持名字惟一性，熟练用户只须键入指向所需节点的完整路径名。

3. 菜单放置

菜单可以显示在显示器屏幕或者第二个辅助屏幕上（图8-12），也可以印刷在输入板或者功能键标签上。屏幕上显示的菜单可以是静态的并且长久可见的，也可以按要求动态出现（如浮动菜单、隐含式菜单、弹出式菜单、下拉式菜单和拉出式菜单）。

彩图I-18所示的印刷在输入板上的静态菜单，在应用目的固定的系统中使用起来很容易，但是，

使用输入板或者辅助屏幕都需要用户的视线离开应用显示器，这样就破坏了视觉连续性。优点是节省显示器空间，这一点常常很珍贵。还有一个优点是可以在一个菜单中容纳大量的命令。



图8-12 一个双显示器工作站。两个显示器中的一个用来显示图的轮廓，另一个显示细节；也可以一个显示图形，另一个显示菜单。（Intergraph公司授权。）

弹出式菜单是在做出相应选择的时候出现在屏幕上的，它要么是响应一个明确的用户操作（典型地按下鼠标或输入板定标器的按钮），要么是由于下一个对话步骤要求进行菜单选择而自 367
动出现。这类菜单一般显示在光标位置，这个位置通常是用户的视觉注意的中心，可以保持视觉连续性。弹出式菜单一个引人注意的特点是认为最近选择过的菜单项比其他项更有可能被再次选到，因此首先高亮度显示最近从选项集合中选择过的菜单项，将光标定位在该菜单项上。这样，如果以使用频率对菜单项进行排序，则首先高亮度显示最常用的命令，应该将最常用命令放在菜单的中间（而不是顶部）以便选择其他菜单项时光标移动距离最小。

弹出式菜单及其他隐含式菜单节省了宝贵的屏幕空间——用户界面设计者最宝贵的资源之一。第2章和第19章讨论的快速RasterOp指令促进了这些菜单的使用。

弹出式菜单经常是上下文敏感的。在一些窗口管理系统中，如果光标是在窗口的标志区（窗口的顶部标题）中，就在菜单中显示与窗口操纵有关的命令；如果光标是在窗口的正文区中，就在菜单中显示与应用程序本身相关的命令（显示哪些命令依赖于光标所指对象的类型）；否则，在菜单中显示创建新窗口的命令。这种上下文敏感性一开始可能会使新手不知所措，但一旦理解了就非常有用。

与弹出式菜单不同，下拉式菜单和拉出式菜单被定位在沿屏幕边缘排列的菜单条中。Apple Macintosh、Microsoft Windows以及Microsoft Presentation Manager都使用下拉式菜单。图8-13所示的Macintosh菜单还利用了快捷键和上下文敏感性。图8-14显示了拉出式菜单，它可作为下拉式菜单的一个替代品。这两种类型的菜单有一个两级层次结构：菜单条是第一级，下拉式菜单和拉出式菜单是第二级。下拉式菜单和拉出式菜单既可以显式激发，也可以隐式激发。在显式激发情况下，一旦光标停在菜单条上，按下鼠标按钮就会出现第二级菜单；然后可以将光标移动到所选项上再释放按钮。在隐式激发情况下，将光标移到标题栏中就会显示菜单，不需要按按钮。选中一项或者将光标移出菜单区域这两种方式可以释放菜单。这些菜单有时也称为延迟菜单，由于它们显示时比较难以捉摸可能会使新用户糊涂。

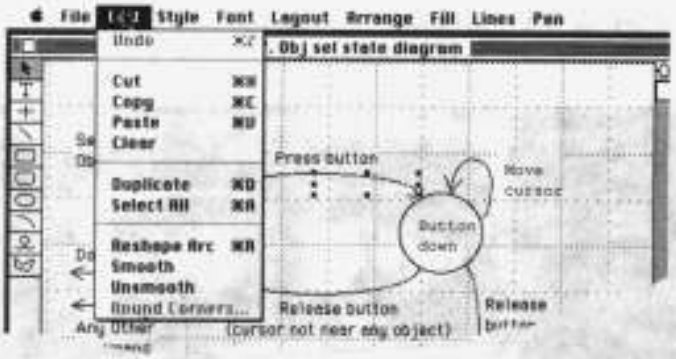


图8-13 一个Macintosh下拉式菜单。最后一个菜单项是灰色而不是黑色，这表示该项当前不可用（当前被选择的对象是一条圆弧，没有圆角）。Undo命令也被置为灰色，这是由于原先执行过的命令不可能被撤消。菜单中的缩略词是供熟练用户使用的快捷键。（Claris公司版权所有，1988，保留所有权利。）

全屏幕菜单用起来可能好也可能不好，这主要依赖于菜单的使用上下文环境。其缺点是应用程序绘图会被弄得模糊不清，从而清除了可能有助于用户做出适当选择的上下文环境。这一缺点也可以克服掉，如使用光栅显示器的查色表在被弄得模糊的应用程序绘图之上以高亮度颜色显示菜单。

4. 可视化表示

菜单表示方式的基本问题是使用选项集合中各元素的文本名字，还是使用图标或是其他的图形化表示，这一问题放在下一章详细讨论。不过，请注意图标表示的菜单比文本化菜单在空间结构的组织上更灵活，因为图标不需要像文本字符串那样长而且细，如图8-15所示，而且容易描述内在的图形概念（主要指图形属性和几何图元）。

5. 当前选择

如果一个系统有这么一个概念，即选项集合的“当前所选元素”，则菜单选择时允许该元素被高亮度显示。有些情况下，由系统提供最初的默认设置并且一直使用到用户修改了默认值。可以用多种方式显示当前所选元素，以汽车收音机上的调节按钮为模式的单选按钮（radio-button）交互技术即是其中一种（图8-16）。另外，一些弹出式菜单假定用户更有可能重选上次选过的菜单项，因此高亮度显示最近选择过的选项并且将光标停在该项上。

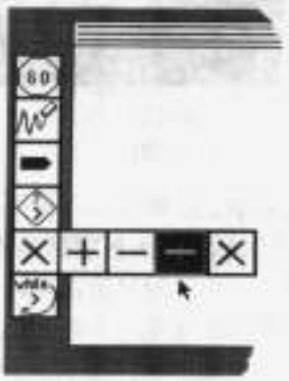


图8-14 一个拉出式菜单，其中最左端的持久显示的元素表示当前的选择，最新选择的菜单项（背景与其他菜单项相反）将成为当前的选择结果。这与大多数菜单的风格相反。大多数菜单中，菜单名被持久显示，并且释放改动过的菜单后不显示当前的选择结果。菜单选自Jovanović的Process Visualization System [JOVA86]。（Branka Jovanović授权。）

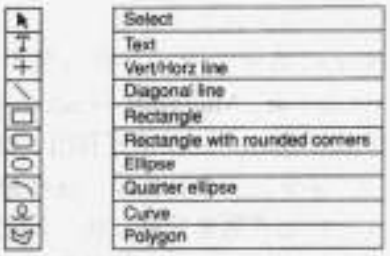


图8-15 同一几何图元的图标化菜单和文本化菜单。图标化菜单比文本化菜单所占空间更少。（Icons©1988 Claris公司，保留所有权利。）

6. 菜单项的大小和形状

定位的精确性和速度受每个菜单项的大小的影响，Fitts法则指出，菜单项越大，选择速度就越快[FITT54；CARD83]，但另一方面，小的菜单项占用的空间更少并且允许在固定大小的区域中显示更多的菜单项，不过，在选择过程中产生的错误更多。因此，在使用小菜单项以节省屏幕空间方面与使用较大菜单项以减少选择时间和错误率方面存在着矛盾。

图8-17显示的弹出式饼形菜单[CALL88]在光标位置出现，用户将鼠标从饼形菜单的中心向需要的选项移动时，目标的宽度变大，从而减少发生错误的可能性。这样，用户在速度与错误的权衡方面能够明确控制，此外，从当前光标位置到每个菜单项的距离相同。



图8-16 用于从相互排斥的选项集合中进行选择的单选按钮技术。（NeXT公司授权，©1989 NeXT, Inc.。）

7. 模式识别

在涉及模式识别的选择技术中，用户使用连续定位设备（如输入板或鼠标）做连续运动。模式识别装置自动将运动序列与一系列已定义的模式相比较，每个模式对应于选项集合中的一个元素。图8-18显示了一个草图模式的集合以及相关的命令，取自Wallace的SELMA排队分析器[IRAN71]，表示删除、大写、移动等校对用的标记特别适用于这种方法[WOLF87]。



图8-17 一个四项饼形菜单

368 ↓ 370

模式识别技术不需要打字技巧并保持触觉的连续性，而且，如果命令与某个对象有关，可以利用光标位置进行选择。很多Macintosh应用中使用的移动命令就是一个很好的例子：光标定位在将要移动的对象上方，按下鼠标按钮选择该对象（反色显示该对象以提供反馈信息）。随着用户移动鼠标（仍然按着按钮），对象也跟着移动，释放鼠标按钮则将对象与鼠标分开。熟练用户利用这项技术可以非常快速地工作，因为该技术不需要工作区域与命令输入设备之间的手部运动。如果与数据输入板和指示笔一起使用，该项技术可以用于至少几十种模式，但是对于用户来说学习大量不同的模式很困难。

最近，Rhyne将透明输入板和液晶显示器组合到一个便携式膝上型计算机的原型中[RHYN87]，各种模式通过透明板输入，并被识别、解释成命令、数字和字母，信息的输入位置也有一定意义，图8-19演示了该设备用于电子表格程序的情况。

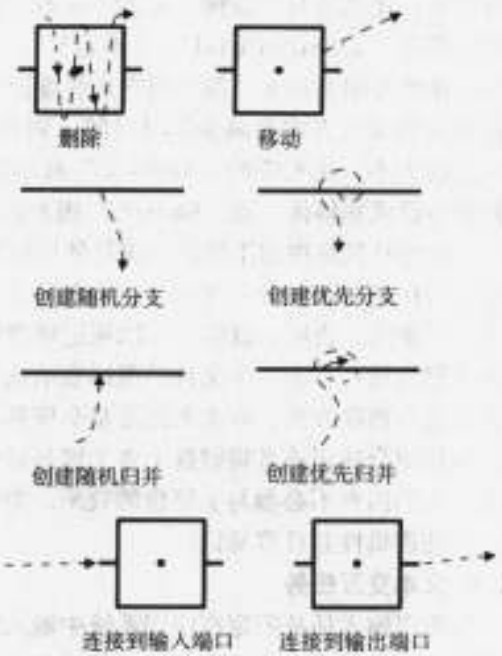


图8-18 用虚线描述的各种运动被识别成命令，摘自Wallace的SELMA排队分析器[IRAN71]



图8-19 一个IBM实验用的显示器和透明数据输入板。正在执行电子表格应用程序，用户刚刚将三个数字用圆圈圈了起来，指示这三个数字的和输入到含有Σ符号的单元格中，还创建了一个新栏的标题。系统对这些图形和字符进行识别，并用适当的命令输入到电子表格应用程序中。（由IBM T.J.Watson研究室授权。）

8. 功能键

可以将选项集合中的元素与功能键联系起来（如使用常规键盘上的单键作为功能键输入），但不幸的是没有那么多的键可用！各个键可用于分层选择模式中，还可以使用和弦修改它们代表的含义。比如说与功能键一起按下键盘上的Shift键和Ctrl键。不过，要学习用于一些命令的外部键组合（如Shift+Ctrl+L）并不容易。一般它们留做想提高效率的常规用户练习。在键盘上放一张练习用的模板，提示用户这些难记的组合键可以加快学习过程。也可以定义一些代表某种含义的组合方式来减少学习时间，例如，Macintosh上的Microsoft Word使用“Shift+>”增大点的大小，用相应的“Shift+<”减小点的大小；“Shift+I”将普通正体字设成斜体，或将斜体字设成非斜体，而“Shift+U”用类似方式处理带下划线的正文。

371 ~ 372

一种弥补鼠标按钮不足的方法是使用时间因素来扩展一个按钮可能包含的意义。例如，通过单击与快速双击区分。如果双击代表的含义与单击所代表的含义在逻辑上有关联，则这种技术效率非常高，否则，就需要采取死记硬背的方式来记住单击与双击分别代表什么含义。这样的例子很常见：单击一个文件的图标表示选中该文件，双击表示打开该文件。在擦除命令上单击表示进入擦除方式，双击则擦除整个屏幕。这项技术也可用于多按钮鼠标上的每个按钮。将鼠标按钮组合使用或者将键盘上各个键与鼠标按钮一起组合使用，可以产生与使用多个按钮逻辑上（人的因素不必参与）等价的效果。为达到最佳效果，用于组合多种模式的组织方案必须有一定的逻辑性并且容易记忆。

8.2.4 文本交互任务

文本串输入任务限定在应用系统中输入的字符串不赋予任何特殊意义，因此，输入一条命令的名字不是一项文本输入任务，相反，输入图形的说明以及向文字处理器中输入文本都是文本输入任务。显然，最常用的文本输入技术就是使用QWERTY键盘。

1. 字符识别

用户使用连续定位设备来输入字符，通常用输入板的指示笔。计算机对这些字符进行识别，

这比识别扫描的字符要容易得多，因为输入板记录了笔划的顺序、方向，有时还记录笔划的速度和压力。可以用一种模式识别算法将这些记录内容与所存储的每个字符模板相匹配。例如，大写字母“A”包含三划：向下两划、水平一划。可以训练识别器识别手写的不同字体：每个字符的各项参数通过用户所画的样本计算得到。自20世纪60年代早期就已经在交互图形学中使用字符识别器[BROW64；TEIT64]，在[NEWM73]中描述了由Ledeen开发的一个Teitelman的识别器的简化改进版本，在[WARD85；BLES86]中描述了一个商用系统。

由于一秒钟很难手写印刷体两个以上字符（不信你试试看！），所以字符识别不适用于大量的文本输入。相同字母手写草体输入比手写印刷体要快，但是没有比较简单的识别算法适用于草体字母：每个人的笔迹中存在的巨大差异以及难以将单词区分为一个个字母是两大难题。

2. 菜单选择

可以用菜单显示一系列的字母、音节以及其他基本元素，用户通过使用选择设备从菜单中选取字母来输入文本，这项技术在几种情况下很好用。首先，如果只输入一个很短的字符串并且用户的双手已经位于定位设备之上，则通过菜单进行选择比手在键盘上来回移动更快。其次，如果字符集合很大，则这种方法完全可以替代键盘。 373

也可将层次菜单选择用于大型字符集合，如用在汉语与日语中，这样的系统用符号的图形特征（如粗水平线、粗垂直线等）来表示层次。一种更常用的策略是通过语音拼写来输入单词，然后将字符串与字典中的匹配。例如，日语使用两套字母系统（片假名和平假名）来拼写上千个假名字符，这些字符拼写从汉语借鉴而来。

3. 文本输入技术的评估

如果要输入大量文字，惟一能很好替代熟练的键盘打字员的工具就是自动扫描仪。图8-20列出了经实验测定的各种技术的击键速度。一个需要逐字找键盘的打字员由于要执行找键盘、将手移动到该键位置、击键等任务而速度很慢；但受过训练的打字员执行击键任务，有时只需要手或手指略略移动一点就到该键位置。图8-20没有列出的语音输入技术虽然慢，但对于那些必须腾出手来做其他事情（如处理日常文本工作）的应用来说很有吸引力。

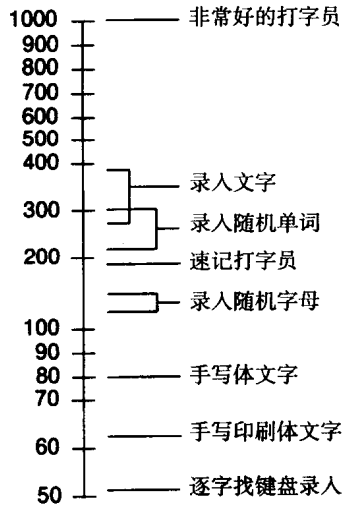


图8-20 使用不同技术输入文本和数字信息所需的数据输入速度，以每分钟击键次数计算。（摘自［VANC72，P.335］与［CARD83，P.61］。）

8.2.5 定量交互任务

定量交互任务用于在某个最小值和最大值之间指定一个数值，典型的交互技术有键入数值、用刻度来设置值、使用上下变化的计数器选择值。与定位任务类似，定量交互任务可以是语言的，也可以是空间的。如果是语言的，则用户知道输入的具体值；如果是空间的，则用户用某一增量来增加或减少一个值，从而得到的可能是最终想要值的近似。前一种情况下，交互技术显然必须包括所选值的数值反馈信息（获取反馈的一种方式是给用户以输入实际值）；后一种情况下，对数值的设置有一个大概印象更加重要，这一般用空间定向反馈技术来实现，如显示一个刻度盘或者标尺，其上显示当前（也可能是以前）的值。 374

输入数值的一种方式是使用电位器。决定使用旋转式还是直线式电位器需要考虑的因素是，改变一个值所产生的视觉反馈信息是旋转式的（如一个转动的时钟臂）还是直线式的（如一个上升的温度计）。虽然旋转式电位器有指针指示，而且旋转式电位器更容易调节，但一个或一组滑动式电位器的当前位置比旋转式的更容易掌握；但是，大多数图形系统制造者只提供旋转式电位器。另一方面，旋转式电位器很容易调节。同时采用直线式电位器和旋转式电位器有助于用户将数值含义与每个设备联系起来，始终采用一致的方向也很重要：顺时钟方向运动与向上运动通常应增加值的大小。

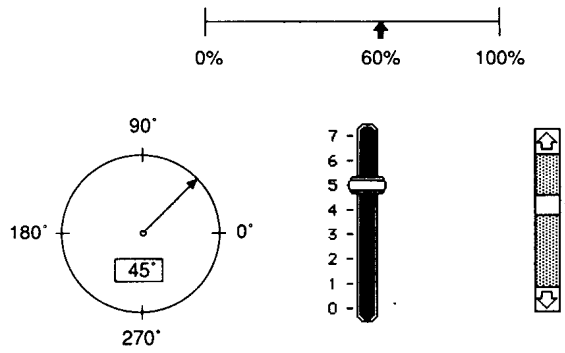


图8-21 用户通过拖动控制指针输入数值的几种刻度盘。通过指针和数字显示的两种方式来提供反馈信息。（垂直滑动尺由苹果计算机公司版权所有。）

使用连续刻度操纵，用户可将光标定位在所示标尺的当前值指示器上，按下选择按钮，沿着标尺将指示器拖到期望数值上，然后释放选择按钮。通常使用指针指示标尺上所选的值，还可以提供数值回应。图8-21显示了这样几种刻度盘以及相关的反馈信息。用这种方式输入数值的范围或精度可以通过使用如下方法进行扩充：将定位设备作为相对设备而非绝对设备，并使用8.1.1节讨论的可变C/D率。这样可以通过重复一连串的击键操作来增加值的大小：移到右边，拿起鼠标，移到左边，放下鼠标，等等。Thornton的数字转轮[THOR79]就采用了这样一种技术。

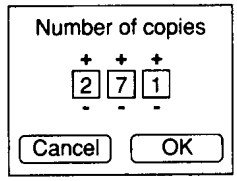


图8-22 一种用于指定数值的上下变化的计数器。用户将光标定位在“+”或“-”上并按住定位设备上的选择按钮，则相应数字增大或减小直到释放按钮为止

如果所需的解决方案比连续刻度操作技术所能提供的要求更高，或者如果屏幕空间非常珍贵，可以使用上下变化的计数器，如图8-22所示。

8.2.6 三维交互任务

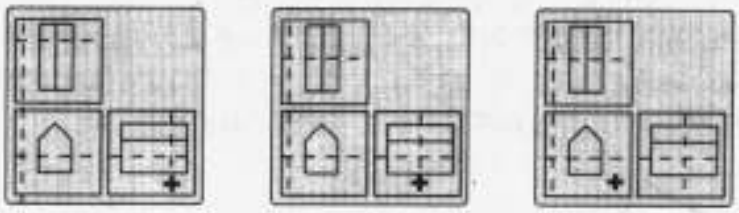
前述用于二维应用系统的四种交互任务中有两种在三维应用中更加复杂：定位任务和选择任务。在这一节，我们还引入另外一种三维交互任务：旋转任务（用于在三维空间中给对象定向）。复杂的一个主要原因是难以理解光标或对象相对于其他显示对象的三维深度关系，这一点与二维交互完全相反。在二维交互过程中用户很容易理解光标是在某对象的上方、旁边还是正指向对象。复杂的一个次要原因是因为通常所用的交互设备（如鼠标和输入板）都只是二维设备，需要用某种方法将这些二维设备的运动情况映射到三维空间中。

与左、右眼视图相对应的立体眼镜有助于理解一般的深度关系，但是作为一种精确定位的方法其精确性很有限。有关将立体眼镜用于人眼的方法在第14章、第18章以及[HODG85]中有所讨论，其他表现深度关系的方法在第14~16章讨论。

本节第一部分讨论定位和选择技术，这两种技术联系很密切。第二部分讨论交互旋转技术。

图8-23表现了一种三维定位的常用方法。在鼠标控制下，二维光标在三个视图之间自由移动，用户可以用按下按钮、拖动、释放按钮的操作序列，选择任一条三维光标的虚线并拖动该虚线。如果按下按钮事件紧挨着两条光标虚线的交点，则两条光标线都被选中，并随着鼠标一

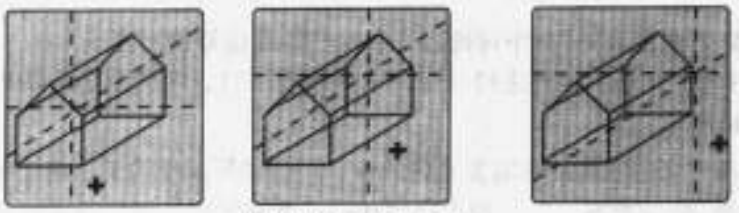
起移动（8.3.2节讨论的重力可以使拾取交点非常容易）。虽然这种方法要求用户必须同时在一种或两种维数的空间中工作，看起来有一定的限制，但是，将三维操纵任务分解成更简单的低维数任务有时候很有好处。使用多个视图有助于选择和定位：相互重叠的对象在一张视图中难以区分，但在另一张视图中可能不会相互重叠。



当二维光标位于三维光标虚线之上时按下按钮　　拖动三维光标，所有视图被相应更新　　释放按钮，二维光标不再控制三维光标

图8-23　对同一场景（一所房屋）使用三个视图的三维定位技术。二维光标（+）用于选择一条三维光标的虚线

另一种三维定位方法由Nielson和Olsen[NIEL86]开发并在图8-24中有所描述，该方法要求所有三个主轴都用非零长度来投影，通过在三个主轴的投影方向上移动鼠标来控制三维十字光标，光标的十字标线与主轴平行。图8-25说明如何将二维定位器的移动映射到三维：二维区域中鼠标的移动影响一个特定的轴。当然，三维运动被限制在一次一个轴。



在z轴方向上移动二维光标（+）使三维光标在z轴方向移动　　水平移动二维光标使三维光标在x轴方向移动

图8-24　三维光标的移动受二维光标移动的方向控制

这两种技术具体说明了二维定位器的运动映射到三维运动的方法，我们其实可以用按钮来控制哪个三维坐标受定位器的二维自由度影响。例如，定位器一般控制x轴和y轴，但按下键盘的Shift键就改为控制x和z（这要求键盘如第4章中讨论的那样不被编码）。另外可以用三个按钮分别将定位器与所选择的坐标轴相关连。还可以不从二维设备映射到三维，而是用真正的三维定位器，如8.1.6节的游戏杆和跟踪球。

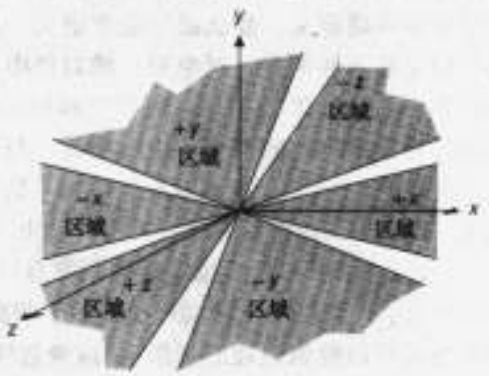


图8-25　鼠标移动的六个区域，这些区域使三维光标沿主轴移动

受约束的三维运动在三维定位中非常有效，网格与重力有时可以弥补深度关系中的非确定性，而且可以辅助进行精确定位。有些物理设备提供另一种约束形式，这些物理设备使得沿主

375
|
377

轴运动比沿其他方向运动更容易，一些跟踪球和游戏杆就有这种特性。也可以用等角应力计和空间球设备来模拟（见8.1.6节）。

不过，针对上下文环境的约束常常比这些一般约束更有用，它可以让用户定义运动应该与某直线或某平面平行或者位于其上，而不是必须与主轴和平面平行。例如，采用Nielson和Olsen开发的一种方法[NIEL86]，所选对象的局部坐标系定义（如图8-26所示）了运动方向。在一种由Bier[BIER86b]开发的更普及的技术中，用户将一个名为skitter的坐标系放在对象的表面，重新定义可能的运动方向（如图8-27所示）。另一种约束特定平面上运动的方式是给予用户对视图平面方向的控制权，并且将平移限制在与视图平面平行的方向。

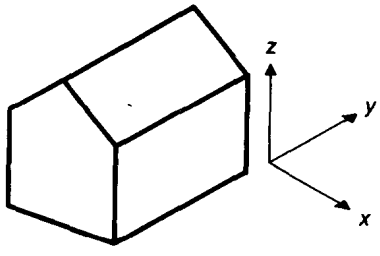


图8-26 屏幕显示房子的局部坐标系，其中指出了对象可以移动的三个方向。为保持刺激-响应兼容性，我们可以用鼠标移动的方向来决定所选择的轴，如图8-25所示

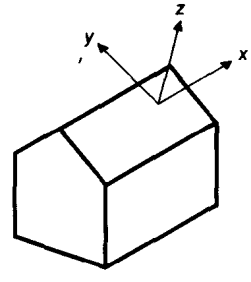


图8-27 交互地调整坐标系，使该坐标系(x, y)平面与屋顶平面相重合，所示坐标系显示出任一可平移对象的三个运动方向

378 一种三维拾取方法已在第7章中讨论过，它由二维定位器确定的一个(x, y)位置，找到有着最大z值的输出图元。另外一种方法可以在显示线框视图时，用三维定位器找到与该定位器的(x, y, z)位置最近的输出图元。

在定位与选择方面，三维旋转需要考虑的事项包括理解深度关系、将二维交互设备映射到三维以及保证刺激-响应兼容性。一种容易实现的三维旋转技术提供滑动标尺或刻度表来控制围绕三个轴的旋转。刺激-响应兼容性要求正常情况下三个轴应当位于屏幕坐标系中：向右是x方向，向上是y方向，向屏幕外（或屏幕内）是z方向[BRIT78]。当然，旋转的中心要么必须以单独步骤明确定义，要么必须隐含定义（典型的是以屏幕坐标原点、对象原点或对象中心为旋转中心）。如图8-28所示，围绕屏幕的x轴和y轴旋转非常简单。随着滑动块的移动，与滑动块相关联的(x, y, z)坐标系被旋转以显示旋转的效果。可以用二维跟踪球代替两个滑动块。

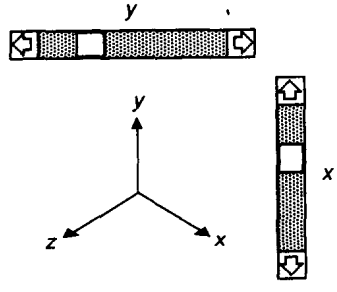


图8-28 用于控制绕屏幕x轴和y轴方向的旋转的两个滑动标尺

通过添加一个用于z轴旋转的刻度盘可以很容易地将二轴旋转方法扩展到三个轴旋转，如图8-29所示（刻度盘比滑动尺更适用于刺激-响应兼容性）。图8-30所示的立方体表面刻度盘的排列方式可以提供更强的刺激-响应兼容性，该立方体清楚地指出每个刻度盘控制的轴。也可以用三维跟踪球代替这些刻度盘。

鼠标的移动可以直接映射到对象的移动，而不需要滑动标尺或刻度盘这些媒介。展现在用户面前的情景就好像是将图8-28中的两个滑动尺附着在被旋转对象的上方，这样水平方向上的

鼠标移动被映射到围绕屏幕坐标y轴方向的旋转，而垂直方向上的鼠标移动被映射到围绕屏幕坐标x轴方向的旋转（图8-31a）。对角线上的移动没有任何影响。滑动标尺并不真正显示出来，但是用户要学会想像它们就是显示在那里。此外，可以告诉用户：一个想像中的二维跟踪球被附着在被旋转对象的上方，这样可以用跟踪球代替鼠标来做垂直、水平或对角线方向上的移动（图8-31b）。这两种方法都提供三维空间中的两轴旋转。

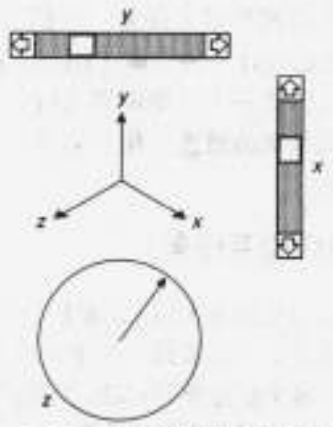


图8-29 两个用于控制绕屏幕上x轴和y轴方向旋转的滑动标尺与一个用于控制围绕z轴方向旋转的刻度盘。该坐标系代表世界坐标系，并且显示了世界坐标系如何与屏幕坐标系相关联的

对于三轴旋转，有三种与现实世界中概念相近的方法非常有趣。在相互重叠的滑动尺方法中[CHEN88]，显示在用户面前的是两个直线型滑动尺，这两个滑动尺与一个旋转型滑动尺相互重叠，如图8-31c所示。直线型滑动尺中的运动控制绕x和y轴方向的旋转，而围绕两个直线型滑动尺的相交部分的旋转运动控制z轴方向的旋转。在由Evans、Tanner和Wein开发的一项技术中[EVAN81]，三个连续的鼠标位置被相互比较，以确定鼠标的移动是直线型的还是旋转型的。直线型水平或垂直方向的运动控制绕x和y轴的旋转，直线型对角线方向的运动既围绕x轴旋转也围绕y轴旋转，并且旋转型运动控制绕z轴的旋转。尽管这是一种相对技术，不需要直接在被旋转对象之上或者一个专门区域中做运动，但可以指导用户利用这些运动来操纵附加在对象之上的三维跟踪球（图8-31d）。在由Chen开发的虚拟球体方法中[CHEN88]，用户以一种独立方式实际操纵这个附着的三维跟踪球，就好像是个真正的球一样。通过按下鼠标按钮，鼠标的移动可控制跟踪球旋转就像是用手指移动一个真的跟踪球一样。对这后两种方法进行比较的一项实验[CHEN88]表明，这两种方法没有性能差异，但用户一般更喜欢Chen的方法。

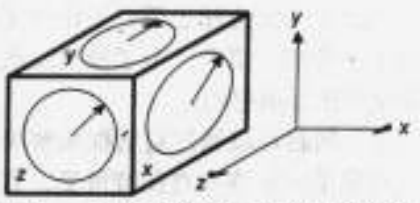
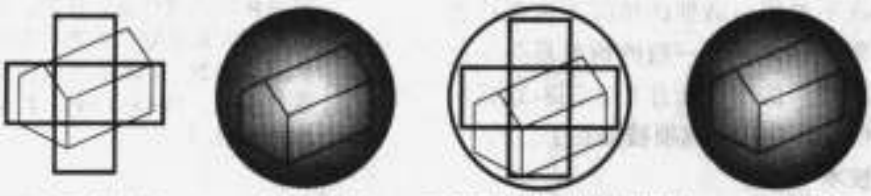


图8-30 三个用于控制绕三个轴旋转的刻度盘，立方体上刻度盘的位置提供很强的刺激-响应兼容性



a)两个滑动标尺 b)二维跟踪球 c)两个滑动尺，旋转刻度盘 d)三维跟踪球

图8-31 三维旋转的四种方法。在每种方法中，用户使用二维设备做运动，分别对应于将实际设备附着在对象之上时所做的运动。可以转动三维跟踪球从而提供z轴方向的旋转，而二维跟踪球仅提供两轴方向的旋转运动

经常有必要将几项三维交互任务结合起来，这样，旋转运动需要一项被旋转对象的选择任务、一项旋转中心的定位任务和一项实际旋转的定向任务。指定一个三维视图可以被看成一项组合的定位（眼睛位于何处）、定向（眼睛如何定向）以及缩放（视图的范围，或有多少投

影面被映射到视口）任务。我们可以将前面讨论过的一些技术结合起来创建这样一项任务，也可以通过设计一种鸟瞰（fly-around）功能来创建，观察者可以利用这种功能驾驶一架想像中的飞机围绕一个三维世界飞行。一般用螺距、卷轴、偏航角来控制，外加速度用来加速减速。利用鸟瞰的概念，用户需要一个总的视图（如二维平面视图）来指示想像中飞机的地面位置和航向。

8.3 复合交互任务

复合交互任务(CIT)是基于上一节所介绍的基本交互任务而建立的，事实上，一个复合交互任务是若干个集成到一个单元的基本交互任务。复合交互任务主要有三种形式：(1) 对话框，用于指定多个信息单元；(2) 构造，用于创建需要两个或更多位置的对象；(3) 操纵，用于对已经存在的几何对象进行重定形。

8.3.1 对话框

我们经常需要从一个选择集合中选取多个元素。例如，文本属性（如斜体、黑体、下划线、空心和全部大写）都不是互斥的，用户可能会同时选择其中的两个或更多个。另外，相关的属性可能组成几个集合，如字体与字型。在单选时有用的一些菜单技术不能满足多选的需要。例如，当做了一次选择之后，下拉式菜单和弹出式菜单通常都会消失，为了做第二次选择，你必须要再次激活它们。

这个问题可以通过对话框来解决。对话框是一种菜单，它将一直保持可见，直到用户显式地关闭它。另外，对话框允许从多个选择集合中进行选择，并且它包含了一些区域，在这些区域中，用户可以输入文本或数值。在对话框中所做的选择可以立即被改正。当所有的信息都输入对话框之后，用户一般通过一个命令显式地关闭它。在对话框中指定的属性或其他值也可以立即被应用，让用户预览字体或线型等改变的效果。对话框中有时包含一个“应用”命令，使得在不关闭对话框的情况下也能让新的属性值生效。然而，更一般的情况是必须关闭对话框才能使新的设置起作用。图8-32显示了一个对话框，其中几个选项被加亮了。

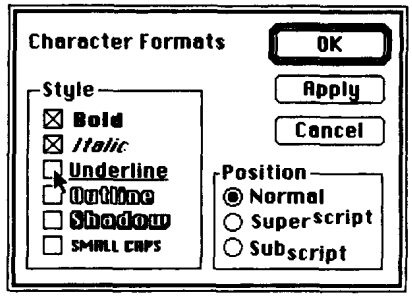


图8-32 一个文本属性对话框，其中几个不同的属性值被选中了。“OK”按钮周围的黑框表示可以用键盘的“Return”键作为它的替代。“Apply”按钮用来应用新的属性值，从而用户可以观察到它们的效果。“Cancel”按钮用来恢复变化前的属性值。注意，文本属性值是同时通过它们的名称和图形化示例来表示的。（屏幕图像由微软公司版权所有，1983~1989。经微软公司许可引用。）

8.3.2 构造技术

构造直线段的一种方法是让用户指定一个端点，然后指定另一个端点；当第二个端点被指定后，在两个端点之间画出直线段。然而，采用这种方法，用户很难在最终的直线段被画出之前尝试不同的线段位置，因为在指定第二个端点之前，直线段并没有实际被画出来。通过这种交互方式，用户为了重新定位端点，每次都必须调用一个命令。

一个较好的方法是第2章讨论过的橡皮筋线技术。当用户按下按键（通常是输入板指示笔的按键或鼠标的按键）时，光标（一般由一个连续定位设备控制）建立了直线段的起始端点。随着光标的移动，直线段的终点也跟着移动。当按键释放时，直线段的端点被确定。图8-33

显示了一个采用“橡皮筋线”技术的绘图顺序。用户的动作序列如图8-34状态图所示。注意，只有当按键被按住的时候，“橡皮筋线”状态才是激活的。也只有在这个状态下，光标的移动才会改变当前的直线段。关于如何将交互技术中的状态转换和与该技术一起使用的设备所提供的转换相匹配的更详细讨论请参阅[BUXT85]。

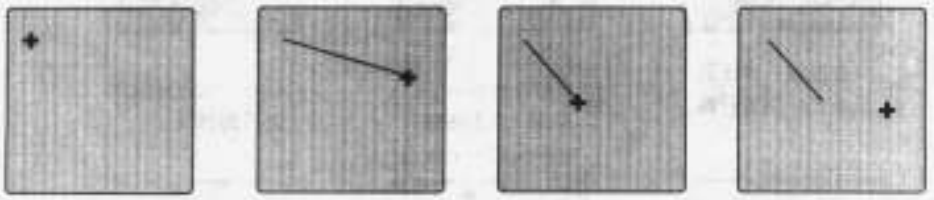


图8-33 橡皮筋线画线



图8-34 用橡皮筋线技术绘图的状态图

从橡皮筋线绘图引申得到了一整套交互技术。橡皮筋矩形技术从一个按键动作开始，这个按键动作确定了矩形的一个角点，然后，它的对角点与光标动态地连接在一起，随着光标移动而移动，直到释放按键为止。这种技术的状态图与橡皮筋线技术的状态图之间的惟一区别是它们的动态反馈不同，一个是矩形，一个是直线段。橡皮筋圆技术创建一个圆，该圆的圆心由初始光标位置确定，该圆经过了当前光标位置，或者该圆在由对角确定的正方形的内部。橡皮筋椭圆技术创建一个长短轴平行于坐标轴的椭圆，该椭圆位于其外接矩形内，而外接矩形由初始光标位置和当前光标位置共同确定。如果外接矩形是正方形，得到的是圆。所有这些技术有共同的用户动作序列：按键、移动定位器并观察反馈、释放按键。

一种用来创建折线（首尾相连的直线段序列）的交互技术是橡皮筋线技术的扩展。用户首先输入了创建折线的命令，然后通过点击按键输入橡皮筋线的顶点。在输入了所有的顶点之后，用户输入结束命令。结束命令通常用如下几种方式完成：不移动光标而双击按键；点击鼠标的另外一个键；或者输入一个新的命令。如果这个新的命令来源于菜单，那么折线的最后一条直线段随着光标移到菜单，然后消失。图8-35显示了创建一条折线所需要的典型的事件序列。图8-36是相应的状态图。

379
-
383

多边形可以通过类似的方法绘出。在某些场合，用户用光标返回到起始顶点处而告诉系统多边形输入结束了。在另外一些场合，用户利用一个功能键显式地通知系统，系统自动完成多边形闭合。图8-37显示了创建多边形的一种方式。

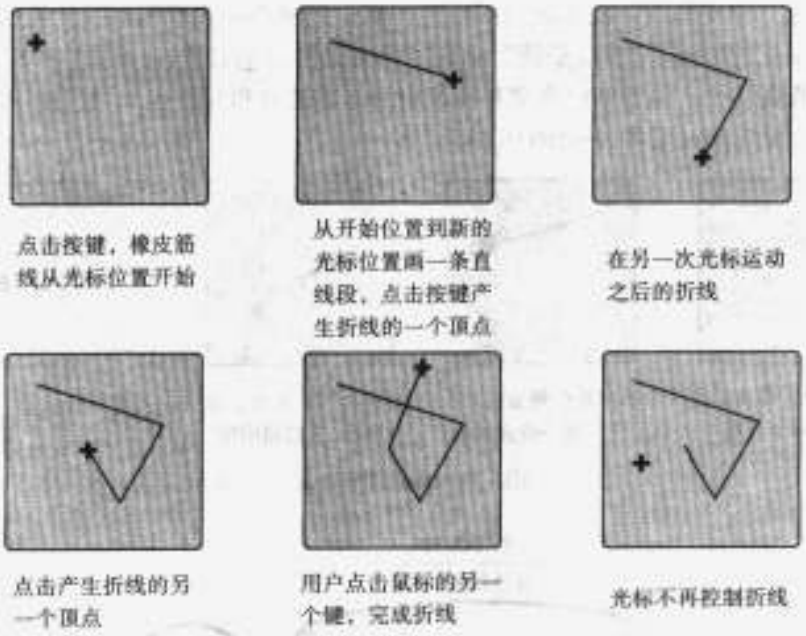


图8-35 用橡皮筋线技术绘图

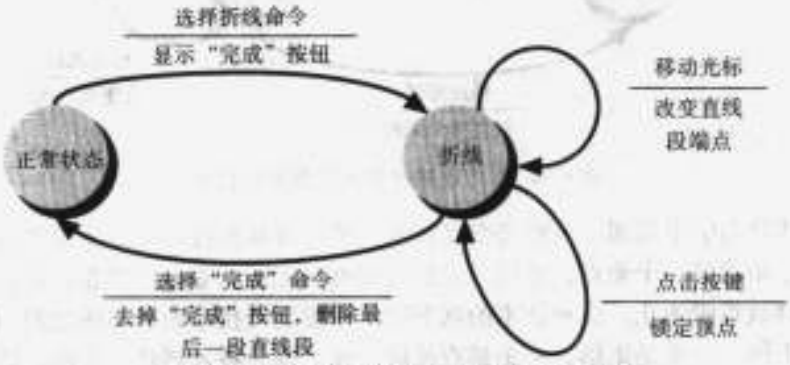


图8-36 用橡皮筋线技术创建折线的状态图

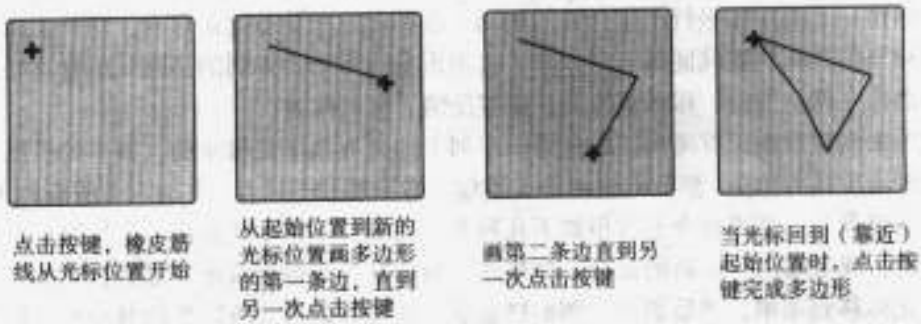
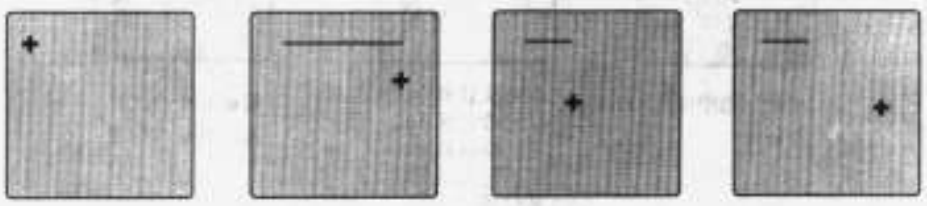


图8-37 用橡皮筋线技术画多边形

在以上任何一种技术中，不同的约束可以施加到当前的光标位置。例如，图8-38显示了使用与图8-33相同的光标位置产生的一列直线段，但是有水平约束。一条竖直线段，或者一条其他方向的直线段，也可以通过同样的方式绘出。全部由水平和竖直线组成的折线，就像印制电

路板、VLSI芯片和一些城市地图中那样，很容易创建；直角可以由用户命令指定，也可以通过光标改变方向自动产生。这种思想可以推广到任意形状，如圆、椭圆或其他曲线。曲线从某个位置开始，然后通过光标的移动来控制曲线的哪些部分被显示。一般地，光标的位置被作为约束函数的输入，约束函数的输出被用来显示对象的适当部分。



按下按键，橡皮筋线从光标位置开始　从起始点到x轴的新光标位置画线　释放按键，橡皮筋线结束，直线段固定　光标不再控制直线段

图8-38　用带水平约束的橡皮筋线绘图

重力是另一种约束。当绘图时，我们经常希望新直线段的端点从已经存在的直线段的端点或其上开始。如果端点是使用网格创建的，那么匹配一个端点是容易的；否则匹配端点将很困难，除非利用费时的放缩功能。这种困难可以用以下方法避免：假想已经存在的直线段周围有一个重力场，当光标进入它的重力场时，它被吸引到直线段之上。图8-39显示了一个带重力场的直线段，重力场在端点处更大一些，使得匹配端点特别容易。

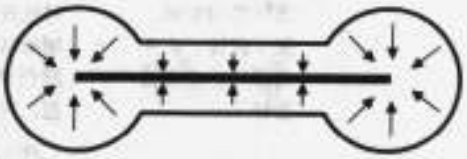
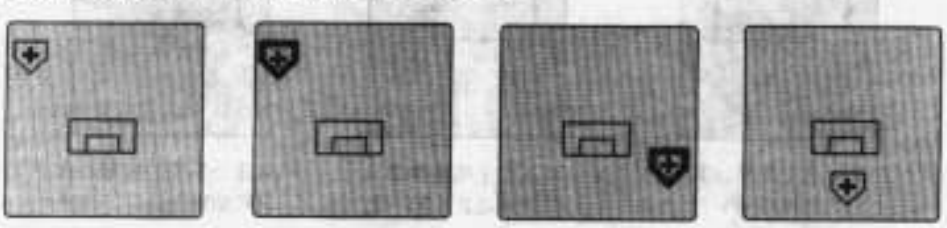


图8-39　直线段被重力场包围，用于帮助拾取直线段上的点；如果光标落在重力场内，它将被吸引到直线段上

8.3.3　动态操纵

仅仅创建直线段、矩形等图形是不够的，在许多场合，用户必须能够修改已经创建的几何实体。

如图8-40所示，在光标的控制下，拖动选定的符号从一个位置移动到另一个位置。通常按下按键动作开始拖动（在某些场合，按下按键也用来选定位于光标下将要被拖动的符号），然后，一个释放按键的动作将符号固定在新的位置，随后的光标移动对符号没有影响。这个按下按键、拖动、释放的序列通常称为*点击-拖动交互*。



将光标定位在被拖动符号之上，按下按键　符号被加亮，确认符号已被选中　几次光标移动　释放按键，符号固定在新的位置

图8-40　拖动一个符号到新的位置

对象的动态旋转也可以通过类似的方法完成，只是我们必须要指定旋转中心或旋转轴。一个方便的策略是让系统显示当前的旋转中心，并且允许用户按照需要修改它。图8-41显示了这

样一个场景。注意，同样的方法能用于缩放，此时用户指定缩放中心而不是旋转中心。

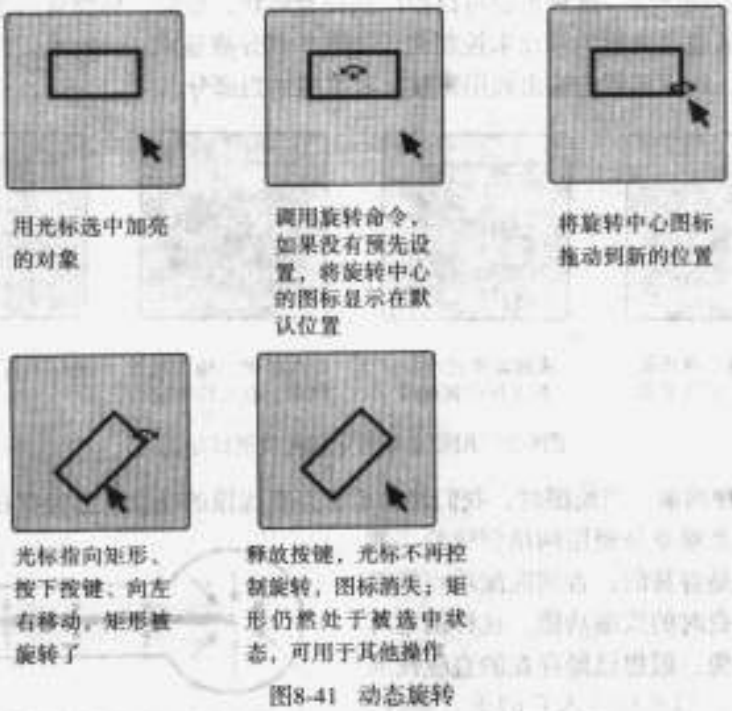


图8-41 动态旋转

手柄的概念有助于提供对对象的缩放，它不需要用户显式地指出缩放中心在何处。图8-42显示了具有8个手柄的对象，手柄被显示成小的方块，它们分布于对象包围盒的角点和边界上。用户选择其中的一个手柄，然后通过拖动就可以缩放对象了。如果手柄位于一个角点上，那么对象的对角点的位置被锁定。如果手柄位于一条边界的中间，那么对象的对边的位置被锁定。当将这种技术集成到一个完整的用户界面中时，只有对象被选中，其上的手柄才会被显示。同时，手柄也是惟一的表示对象被选中的视觉编码，因为其他视觉编码（如粗线段、虚线、亮度改变等）都有可能是图形对象本身的组成部分。（闪烁是另一种可视化表示方法，但它容易分散人的注意力并且有些恼人。）

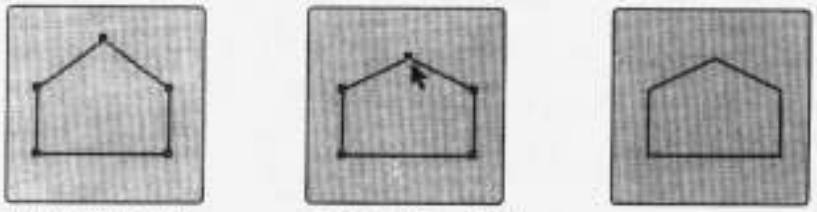


图8-42 用于改变对象形状的手柄

384 - 387

拖动、旋转和缩放影响整个图形对象，如果我们仅仅想改变单个的点，比如多边形的顶点，怎么办呢？可以给顶点命名，然后用户输入顶点的名称和它新的(x, y)坐标。但是，用于移动整个对象的指点–拖动的策略更富有吸引力。在这种情况下，用户指点一个顶点，选中它，拖动它到新的位置。与该顶点相邻的顶点通过橡皮筋线与它保持连接。为了使选中顶点这个任务变得容易，

我们在它周围建立重力场，我们可以在鼠标靠近时让它闪烁，或者我们在每个顶点上显示手柄，如图8-43所示。类似地，用户可以通过选中并拖动的方法移动多边形的一条边，并且保持边的斜率不变。对于平滑曲线和曲面，手柄也可以用来操纵控制形状的顶点，这在第11章将继续讨论。



选中多边形以便修改其顶点，手柄在各个顶点上出现

在顶点处按下——移动——释放按键，导致顶点移动

多边形不再被选中，手柄消失

图8-43 手柄用于改变多边形的顶点

下一章中，我们将讨论了如何将基本交互技术以及复合交互技术集成到完整的人机对话中所涉及的设计问题。

习题

8.1 检查一个你所熟悉的人机界面，列出用过的每一个交互任务，看看它们属于8.2节中4类基本交互任务的哪一类。如果一个交互任务不属于其中任何一类，试着将它进一步分解。

8.2 用鼠标或其他相对定位设备实现具有自适应C/D率的光标跟踪，考察鼠标速率v和C/D比率r之间的不同关系：$r = kv$和$r = kv^2$。同时也要寻找常量k的一个合适值。

8.3 做一个试验，比较下面几对技术的选择速度和精度：

a. 用鼠标和输入板选择屏幕上的一个静态菜单。

b. 用触摸板和光笔选择屏幕上的一个静态菜单。

c. 宽的、浅的菜单与窄的、深的菜单。

d. 只要光标在菜单条上就会出现的下拉式菜单与需要鼠标单击的下拉式菜单。

8.4 扩展图8-6的状态图，使之包含“返回到最低层”命令。该命令把选择层次返回到最低层，因此最先被选中的将再次被选中。 388

8.5 实现一个与任意单词表一起使用的字符输入自动完成技术。用不同的单词集合（如UNIX命令和适当的名称）做测试。考虑如何处理以下情况：匹配不存在；匹配找到之后，由用户键入更正；给用户提示。

8.6 为一列命令或文件系统子文件夹实现一个级联式分层菜单和面板式分层菜单，完成过程中你遇到了什么问题？比较一下这两种技术的选择速度。

8.7 实现一个弹出式菜单，使它在关闭之前允许多重选择。一种关闭的方式是用户将鼠标移出菜单，一种是点击按钮关闭。你更喜欢哪一种关闭方式？解释你的结论。询问5个用过这两种技术的人，看看他们更喜欢哪一种。

8.8 在具有查色表的彩色显示器上实现一个菜单包，使得菜单以亮的但部分透明的颜色显示，而菜单下面用柔和的灰色显示。

8.9 实现本章介绍的任何一种三维交互技术。

8.10 对8.2.6节中介绍的每一种定位技术，确定二维定位器移动被映射到的线和面。

8.11 画出弹出式分层菜单的状态图，画出面板式分层菜单的状态图。 389

第9章 对话设计

我们已经讨论了制作一个交互式图形系统的用户界面所需要的基本部件（building block）——交互设备、技术和任务。现在考虑如何用这些部件（building block）来组合一个有用而友好的界面形式。*用户界面设计*在很大程度上可以说是艺术而不是科学，因此这里所述的一部分是对交互式系统设计的一种看法。如果加以创造性地运用，一些特定的“执行”和“不执行”，有助于把注意力集中在人的*因素*上，也可以称之为交互系统的*工效学*（ergonomics）。

用户界面设计的主要目标是提高学习和使用速度，降低错误率，帮助快速回忆界面的使用方法以及增强对潜在用户和购买者的吸引力。

*学习速度*是指一个新的用户从开始到熟练运用系统所需要的时间。这对一个可能不会被任何一个人所经常使用的系统而言尤为重要：用户一般不太愿意花好几个小时来学习一个一周内仅使用几分钟的系统！

*使用速度*是指一个熟练用户执行系统中某一特定任务所需要的时间。这一点在一个用户要花大量时间来重复使用某一系统时尤为重要。

*错误率*是指用户与系统的每次交互中的错误次数。错误率对学习速度和使用速度都有影响；如果一个系统容易发生操作错误，那么由于用户需要花时间来修正错误，从而延长了学习时间，降低了使用速度。然而，对于某些哪怕一个错误都不能发生的应用系统而言，错误率必须作为一个单独的设计对象来考虑。例如，空中交通控制、核电厂控制以及战略军事指挥控制系统等。这些系统通常以牺牲一些使用速度来换取较低的错误率。

用户能*快速回忆*系统的使用方法是另一个独特的设计目标。因为用户可能较长时间不使用系统，一旦使用，系统应能很快重现于用户脑中。

界面的*吸引力*与系统的市场密切相关。当然，喜欢一个系统或它的特点与系统的易用性不应是一回事。比较两种设计的大量实验表明，一种设计的系统功能很强，但另一种设计的使用速度实际更快。

有时候，系统不太容易做到既易学又易用。而且虽然有一段时间曾是这样，但现在我们已经学会了如何去满足多个设计目标。协调系统的易学性和使用速度的最简单和最常用的方法是为初学者提供一个基本命令的“初学者工具箱”，但它仅仅是所有命令集中的一个子集。这个初学者工具箱在菜单上可以看到，以便于学习。所有初始的和高级的命令，都可以通过键盘和功能键进行操作，从而有利于提高使用速度。一些高级的命令有时也放在菜单中，通常在菜单层次的较低层，供那些尚不知道它们相应键盘命令的用户使用。

需要知道的是，学习速度是一个相对的量。有十个命令的系统要比有一百个命令的系统的学习速度要快，在这种情况下用户对十条命令中每一条的具体功能的理解比一百条的情形要快得多。但如果应用系统的界面需要丰富的功能，十条命令可能需要设计得富有创造性和想像力，但这样会变得不易学习，相反一百条命令可能相当容易地对应到所应用的需要中。

在最后的分析中，即使解决这些目标中的一个也不是一件容易的事。遗憾的是，在用户界面的设计中，规则性的东西太少。设计时的选择是否合适依赖于很多不同的因素，其中包括设计目标、用户特征、使用环境、可用的硬件和软件资源，以及所做的预算。尤其重要的是，必

须屏蔽用户界面的设计者的自我感觉，使得设计的驱动因素是用户的需要，而不是设计者的需要。能做出快速且即席设计的设计者是不存在的。好的设计需要仔细考虑很多问题，并且在真正用户做测试原型时需要有耐心。

9.1 人机对话的形式和内容

人机对话的概念是交互式系统设计的中心，人机对话和人与人之间的对话有许多有益的相似性。毕竟，人类已有了相互通信的有效方式，从这些实践中了解我们所进行的通信方式，对研究人机对话是有意义的。对话不仅包含词语还有手势：事实上，人类在语言出现前就能够用手势、声音和图像（壁画、象形文字）等进行交流。计算机图形突破了与计算机纯语言交互的
392 限制，它允许我们利用图像作为另外一种通信模式。

在人机对话中，许多人与人之间的对话特征应该予以保留，有效通信的人们共享着共同的知识和假设。同样地，人与计算机之间的对话也要有这些共同点。并且，这些假设和知识都应该是针对普通用户的，而不是针对精通计算机用户界面的设计者的。例如，一个研究分子几何结构的生物化学家对原子、键、两面角、残留物等很熟悉，但他不知道也不必知道像链表、画布、事件队列这样的概念。

学习使用用户界面和学习使用外语很相似。回想一下自己学习外语的经历。在努力学习词汇和语法时，句子说得很慢。但随着练习的增加对语法规则更加熟悉，就可以把重点放在词汇量的扩充上，从而更有效地交流你的想法。一个交互系统的新用户也要经历类似的学习过程。事实上，如果一个新的应用概念必须和新的语法规则和词汇一起学习，学习起来就会很困难。因此，设计者的任务就是保持用户界面规则和词汇的简单性，并采用用户已经知道的或容易学习的概念。

人机对话的语言应该是高效的和完备的，并且应该具有自然的序列规则。利用高效的语言，用户可以快速简洁地把命令传给计算机。利用完备的语言能表达所讨论议题的所有相关的想法。定义语言的命令或句法的序列规则应该具有最小数目的简单、易学的案例。简单的序列规则有助于使得训练最少并且允许用户随时把注意力集中在手边的问题上；复杂的法则会使用户思考时分心并使思考过程不连续。

一种用户界面可能是完备的但不一定高效；也就是说，想法的表达可能是困难而费时的。例如，一个系统在逻辑设计时仅需提供简单的部件，或非或者与非，但这样的系统很难使用，因此效率不高。在系统中，最好提供这样的方法，即可以利用少量的基本命令构造复杂的命令。

可扩展性可以用于使系统的语言效率更高，方法是通过现有条目的组合形成新的条目。可扩展性在操作系统中通常通过脚本、编目过程或命令文件方式提供，在程序语言中，通过宏的方式提供，但在图形系统中却很少提到。

在人与人之间的对话中，一个人提出问题或者做陈述，另一个人通常很快做出响应。即使不是马上回答，听者通常也可以通过对方的面部表情或者手势获得信息。这就是反馈，它也是人机对话的重要组成部分。在两类对话中，对话的最终响应或者是利用语言提供，或者是通过手势或面部表情（也就是说，通过图形图像的方式）提供。

说话的人偶尔会犯错误，然后说，“哎呀，我不是这个意思”，而且听者就会忽略最后一句话。在人机对话中，能够撤销错误的操作也是很重要的。

393 在谈话中，讲话者可能在表达想法时需要听者的帮助，或者需要听者进一步地解释。或者说话的人会一时离题转到别的话题上去，中止刚才的话题。在人机对话中，这样的功能也应该可以实现。

在这样的一般框架中，就可以更确切地来定义人机界面的组成部分。它由两种语言构成，其一，用户与计算机的通信；其二，计算机与用户的通信。第一种语言是通过应用到各种交互设备的动作来表达的，也可能通过语音。第二种语言是通过点、线、字符串、填充区域和颜色的组合来形成显示的图像和信息，也可能通过语音合成来表达。

语言主要有两部分组成：语言的含义和语言的形式。语言的含义是指它的内容或者它的信息，而形式是指含义是如何传送的。在人与人的对话中，要表达“我很高兴”的意思，可以通过语音“我很高兴”或者是通过微笑来表达。在人机对话中，要表示“删除temp9”的意思，可以用键盘敲命令“DELETE temp9”或者把表示文件temp9的图标拖到回收站里去。这种界面的形式通常称之为它的“视感”（look and feel）。

在界面设计中，含义有两个要素：概念上的和功能上的。形式也有两个要素：序列和对硬件原语的绑定。人机界面的设计者必须详细说明这四个要素。

概念设计是用户必须掌握的主要应用概念的定义，因此也称之为应用的用户模型。概念设计通常要定义对象、对象的属性、对象间的关系和对象的操作。在一个简单的文本编辑器中，对象就是字符、行和文件，文件的属性就是它的名字，文件是一系列行，而行是一系列字符，对行的操作就是插入、删除、移动和复制。对文件的操作就是创建、删除、重命名、打印和复制。用户界面的概念设计有时候通过用户熟悉的某些东西的类比和隐喻来描述，例如，打字机、卡片盒（Rolodex是美国一家著名的归档卡片盒制造公司，该公司名称已成为“卡片盒”的代名词。——译者注）、绘图表格和仪器、桌面或文件柜。虽然这些类比对初始理解通常是有帮助的，但如果把它们不切实际地引申解释为计算机系统所提供的更高级的功能的话，这些类比就变得有害了[HALA82]。

功能设计指定界面功能的详细设计：对象的每一个操作需要哪些信息，会发生什么错误，对这些错误如何处理，每一个操作的结果是什么。功能设计也称为是语义设计，它定义的是含义，而不是动作序列或产生动作序列的设备。

顺序设计作为界面形式的一部分，定义输入和输出的次序，顺序设计也称为语法设计。对于输入而言，序列由规则构成，通过这些规则含义的最小单元（通过交互技术输入到系统中）构成完整的句子。这里的单元不能再分了，否则含义就不完整了。例如，鼠标的移动和点击这两个动作用来做菜单选择，但它们的单个动作并不能给应用提供有关的信息。 394

对于输出，序列的概念包含空间和时间两个因素。因此，输出序列包含二维和三维的显示布局，以及显示形式的时间变化。与输入序列的情形一样，输出序列的含义单元也不能够再分；例如，对于电路设计者而言，晶体管符号是有含义的，但构成符号的单根线段没有这样的含义，含义通常可以通过符号和图来形象地表达，也可以通过字符串来表达。

硬件绑定设计，也称为词法设计，也是界面形式的一部分，绑定决定了如何从硬件原语形成输入、输出单元的内容。输入原语就是可用的输入设备，输出原语就是图形子程序包提供的形状（如线和字符）以及它们的属性（如颜色和字体）。因此，如第8章所述，对于输入，硬件绑定就是对交互技术的设计或选择。对于输出，硬件绑定设计是由形成图标和其他符号的显示原语及其属性组成的。

为了说明这些思想，让我们考虑一个简单的家具布置规划。概念设计中的对象是房间和不同的家具，对象之间的关系是房间里有家具。对家具对象的操作是创建、删除、移动、旋转和选择；对房间对象的操作是保存和重建。功能设计是详细描述这些关系和操作的具体含义。

顺序设计可以是先选择一个对象，再对这个对象进行操作。硬件绑定构件的输入语言可以是用鼠标选择菜单条中的命令、选择家具对象及提供位置。输出顺序设计定义了屏幕分配，包

括输出被划分成几个不同的区域以及菜单、提示和错误信息的精确显示位置。输出设计的硬件绑定层包括字体、线宽和颜色、填充区的颜色，以及输出原语组合来创建家具符号的方式。

9.2节讨论了用户界面可以采用的一些基本形式；9.3节给出了可应用于所有四种设计级别的一组设计指导准则。在9.4节中，讨论了关于输入序列和绑定的一些问题。9.5节介绍了输出序列和绑定的视觉设计规则。9.6节概括了用户界面的总体设计方法论。

9.2 用户界面风格

人机界面的三种通用风格是：*所见即所得*、*直接操纵*以及*图标*。本节介绍这些既相互关联又相互区别的概念，考虑它们的应用、优缺点以及相互之间的关系。关于其他人机交互风格也有简要讨论：菜单选取、命令语言、自然语言对话以及问答对话。但重点不是这些，因为对图
395 形来说它们不是惟一的。（菜单是最接近的，但它在图形学出现之前就已使用了。然而，图形学允许将图标而不是文本作为菜单元素来使用，并且提供对文本字型、字体及菜单修饰的各种丰富选择。）这些风格之间并不相互排斥，成功的界面经常融合多种风格的元素来达到设计目标；而决不是单独一种所能达到的。

9.2.1 所见即所得

所见即所得（即WYSIWYG，发音为wiz-ee-wig）是交互式图形学的基础。其表示方式基本上与应用程序创建的图像相同，用户用这种表示方式在WYSIWYG界面上进行交互显示。绝大多数交互式图形应用程序具有WYSIWYG构件，但并不全是这样。

许多文本编辑器（绝大多数图形应用程序）具有WYSIWYG界面，以黑体字符形式显示的文本打印时也是黑体字符，用一个非WYSIWYG编辑器，用户可以看到文本中的控制代码。例如，

In this sentence, we show @b(bold), @i(italic), and @ub(underlined bold) text.

就是下列硬拷贝输出：

In this sentence, we show **bold,** *italic*, and **underlined bold** text.

一个数学公式的非WYSIWYG形式可能是这样的：

@f(@i(u)@sub(max) − @i(u)@sub(min),@i(x)@sub(max) − @i(x)@sub(min))

所得到的结果是：

$$\frac{u_{max} - u_{min}}{x_{max} - x_{min}}$$

在这样的非WYSIWYG系统中，用户必须把结果的想像中的形式和控制代码相互转化。在输入的代码处理之前，并不能确保控制代码能生成想像中的形式。

WYSIWYG也有一些缺点。当屏幕与硬拷贝设备之间的空间、亮度或色彩分辨率不一样时，很难在两者之间建立精确匹配。第13章讨论了在精确复制色彩方面可能引起的问题。更重要的是某些应用程序不能用单纯的WYSIWYG界面来实现。首先考虑文本处理这种最常见的WYSIWYG应用程序。许多文本处理器提供了标题类型，用于定义章、节、小节以及其他标题的视觉特征。这样，“标题类型”是一个必须视觉表示的对象属性。但标题类型并不是最后硬拷贝的一部分，因此，由定义来看，也不是显示的一部分。有一些简单的解决方法，比如在显示的左边空白显示标题类型代码，但它们与WYSIWYG思想相违背。就是这个原因，
396 WYSIWYG有时也叫作“所见皆所得（what you see is all you get）”。作为第二个例子，图7-1所示的机器人手臂没有展现出机器人的身体、手臂等部位之间存在的层次关系，它确实没有展

示这些关系。这些例子不是用来表现WYSIWYG的，而是说明它的局限性。

9.2.2 直接操纵

直接操纵用户界面是指用视觉表示形式能对对象、属性或关系进行操作的用户界面。操作由作用在视觉表示物上的动作激发，比较典型的是使用鼠标，也就是说，命令并不是用传统的方式（如菜单选择或键盘）显式激发，而是隐含在动作中，且动作是作用在视觉表示物上的。这种表示物可以是文本（如对象或属性的名字），也可以是更一般的图形图像（如图标）。在本节的后面，我们将讨论适合视觉表示物（文本或图标形式）的环境。

图9-1所示的是Macintosh界面使用直接操纵的一部分。磁盘和文件都表示成图标，从一个磁盘拖动一个文件的图标到另外一个磁盘就是将文件从一个磁盘拷贝到另一个磁盘，拖到垃圾箱图标就删除了该文件，在早期的Xerox Star系统中，拖动一个文件到打印机图标上就会打印文件，发明“直接操纵”这个词的Shneiderman[SHNE83]讨论了这种技术的其他一些例子。



图9-1 Macintosh屏幕。右上角是磁盘图标；它下面的是目录图标，灰色表示目录是打开的。左边是打开目录的内容，用带文字的图标表示文件。围绕在光标周围的图标轮廓所代表的一个文件正被拖进右下角的垃圾箱中。（屏幕图形由苹果公司提供。）

直接操纵有时代表一种最好的用户界面风格，它确实比较强大而且特别容易学习。但Macintosh界面对熟练用户来说却比较慢，因为即使用户知道其他风格更快时，也不得不使用直接操纵。用直接操纵打印文件“Chapter 9”需要寻找并选取文件的可视表示，然后给出打印命令。要发现这个文件图标可能要滚动查找一大堆图标。如果用户知道文件名，那么直接键入“Print Chapter 9”就会更快。同样要删除类型为“txt”的所有文件就需要查找和选取每一个这样的文件，并拖到垃圾箱。用UNIX风格的命令：“rm *.txt”，它使用星号(*)来查找文件名中所有以“.txt”结尾的文件，用这种方式删除文件，要比直接操纵快多了。 397

将直接操纵和命令语言功能结合起来的用户界面，会比单纯依靠直接操纵方式快。注意，直接操纵鼓励使用较长的、更具有描述性的名称，而这样会抵消由使用键入命令方式节省的一些速度。除了一些简单的面向流程图的入门性介绍，或者在一些特殊情况下能通过实例进行演示的构件[MAUL89；MYER86]之外，有些应用程序并不能进行直接操纵[HUTC86]，如编程。

典型的直接操纵界面混合了其他界面风格，它通常用菜单或键盘激发命令。比如，在大多数画图程序中，用户可以通过命令旋转一个物体，而不是简单指着它，用手柄做旋转操作（如8.3.3节）。事实上，通常很难构造这样一个界面，使得它的所有命令项都能通过直接操纵实现。这就强调对于一个用户界面来说，单一的交互风格是不够的；混合几种交互风格通常要比依赖一种好得多。

表格填写型用户界面是另一种直接操纵类型。这里，表格填写过程是这样的：指定一个字段，然后进行输入，或者通过从一个列表（选择集）中选取针对该字段的几种可能值之一。表格填写的受限功能域及与填写实际表格之间的明显对应关系，使直接操纵成为一种自然的选择。

WYSIWYG和直接操纵是两个分立且不同的概念。例如，图形图像的文本表示可以通过直接操纵来修改，而WYSIWYG系统的图形图像可以通过单纯的命令语言界面来修改。特别是当这两种方式同时使用时，就变得像许多成功用户界面所表现的那样，功能很强，容易学习，并且使用快捷。

9.2.3 图标化用户界面

图标是对象、动作、属性或其他概念的图形化表示，用户界面设计者经常选择图标或单词来表示这些概念。注意，使用图标与直接操纵无关：文本可以像图标一样进行直接操纵，而且文本可以表示概念，有时比图标更好。

文本和图标哪个更好?与大多数用户界面设计问题一样，答案是："要看情况"。图标有许
398 多优点。特别精心设计的图标要比文字更快识别出，并且占的屏幕空间比较小。如果精心设计，图标可以与语言无关，相应的界面可以在不同的国家使用。

图标设计至少有三个不同目标，其重要程度依赖于特定的应用程序：

1) 识别——如何快速准确地识别图标的含义。

2) 记忆——如何一经学习就可记住图标的含义。

3) 区分——如何很好地区分图标。

有关几种不同图标设计方面的实验报告可参见[BEWL83]。关于图标设计问题的更进一步的讨论可参见[HEME82；MARC84]。

表示对象的图标相对来说容易设计。图9-2收集了不同程序中使用的图标。如果给每一属性值一个合适的视觉表示，那么就很容易表示对象的属性。对于交互式图形编辑器中使用的属性，如线宽、纹理和字体，这种表示肯定是可以实现的。数值可用标尺和刻度盘图标来表示，如图8-21所示。

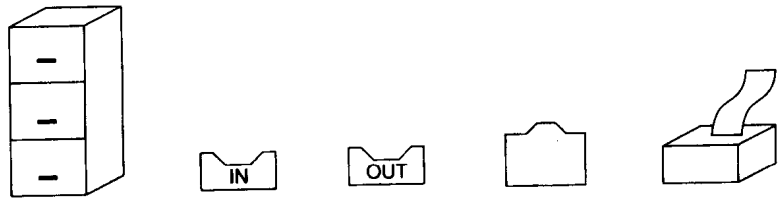


图9-2 用来表示办公对象的常见图标

作用于对象上的动作（也就是命令）也可以用图标来表示。在这方面有好几种设计策略，首先，命令图标可以表示为在现实世界中执行该动作的对象。因此，剪刀可以用来表示剪切，刷子可以用来表示粘贴，而放大镜可以表示缩放。图9-3所示为这些命令图标的集合。这些图标开始比较难懂，因为用户必须首先识别该图标是什么，然后理解所表示的对象做什么。这种理解图标所表示什么对象的两步过程，当然没有仅仅只需要一步那么令人满意。更复杂一点，假设一个对象表示了几种不同的功能。例如，一个刷子图标可以用于粘贴（在指定位置上做粘贴），还可以用于涂颜料（上色）。如果在同一个应用程序中需要粘贴和上色，那么这个刷子图标就有二义性。当然，有时在指定的应用程序中只有一种含义。

命令图标的另一种设计策略是显示命令的前后效果，如图9-4和彩图I-19至彩图I-21所示。如果该对象的表示是紧凑的，那么这样的命令图标就很好用。然而，如果一个命令图标可用于许多不同类型的对象上，那么用图标表示的特定对象就会误导用户认为该命令没有它实际所表示的一般性。

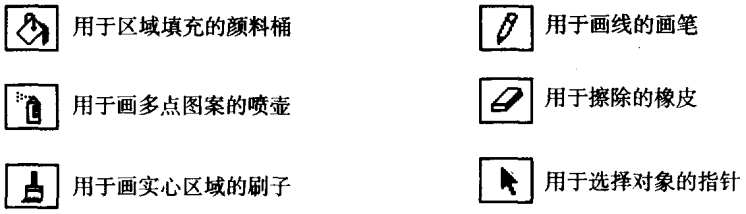


图9-3 用于执行相应命令的表示对象的命令图标（由Claris公司版权所有。）

在每像素两个二进制位的显示器上实现的NeXT用户界面中，图标用于不同的目的，参见彩图I-22。

最后的设计方法就是为操作寻找一个更为抽象的表示，典型例子如图9-5所示。这些表示可以依赖于一些特定文化概念（如八边形的停止标志图）或者可以更具一般性的（如X表示删除）。

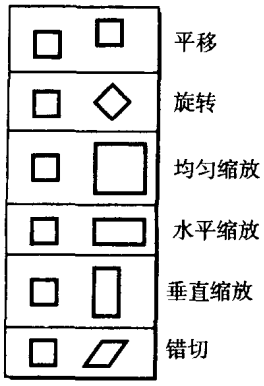


图9-4 通过在应用命令前后显示的正方形来指示几何变换的命令图标

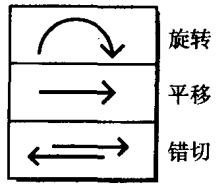


图9-5 用于在图9-4中描述的一些操作的几个抽象命令图标。不是所有的几何操作都能以这种方式表示

任意设计的图标不一定特别容易识别。图9-6显示了用于表示Macintosh程序的大量图标。您准猜不出每个程序是做什么的！然而一旦学会，这些图标所表示的功能看上去就很容易记忆和区分。

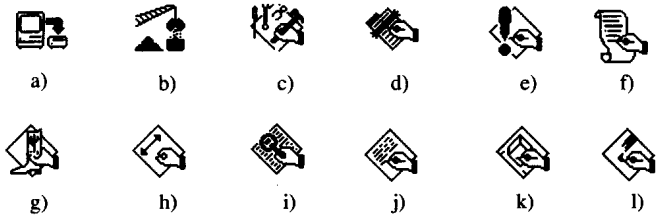


图9-6 表示Macintosh程序的图标。每一个图标表示什么？在绝大多数情况下，图标表示了操作或创建的信息类型。有关答案参见习题9.14。

许多操作系统的图形界面使用图标来区分由不同应用程序使用的文件。由一个应用程序产生的所有文件共享同一个图标。如果目录或磁盘含有许多不同类型的文件，那么由图标形状所允许的区分就很有意义（见图9-7）。然而，如果所有文件是同一类型，那么这种区分就没有什么用（见图9-8）。

图标也可能很少使用。一些用户不喜欢图标，如垃圾箱，认为这些想法太幼稚、“装腔作势”，甚至有失他们的身份。设计者可能同意或不同意这种评价，但用户的意见通常比设计者的意见更重要。不喜欢计算机或程序的用户于是就可能产生一种负面态度，这种情况应认真对待。

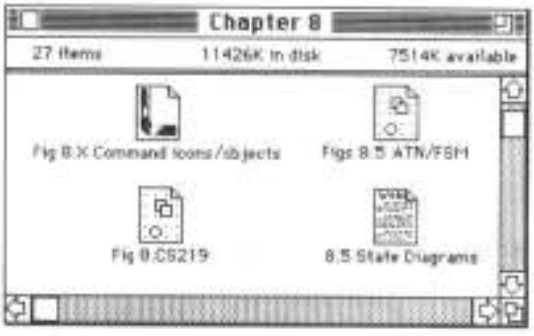


图9-7 用图标和文本表示的磁盘目录的内容。图标有助于区分不同的文件。（屏幕图形苹果公司版权所有。）

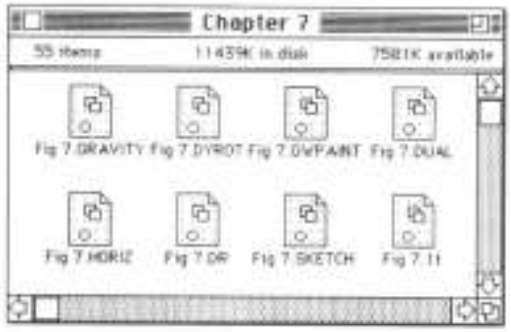


图9-8 用图标和文本表示磁盘目录内容，既然所有的文件都是同一种类型，图标就无助于将一个文件和另一个文件区分出来，而且还占用额外的空间。（计算机屏幕图形苹果公司版权所有。）

9.2.4 其他对话形式

前一节讨论的对话风格也被称为“本质上图形化”，这是因为我们的注意力放在图形化的面向交互上。其他的一些对话形式本质上不是图形形式的，但是却可以用在图形应用程序中。四种这样的形式是菜单、命令语言、自然语言对话、问答对话。我们在前一章已经讨论了关于菜单的许多特定设计问题。在本节中，我们简要讨论一下这些对话形式的更一般的问题。

菜单广泛应用于图形和非图形的应用程序中。但是，对于这两种情况，菜单的最基本的优点是用户可以通过识别记忆来工作，将可视的图片（文本或图标菜单项）和已经熟悉的单词、意思联合在一起。与识别记忆相对的是回忆性记忆，用户为了向计算机输入信息必须从记忆中回忆一个命令或概念。两者相比，菜单减少了用户的记忆负担，因此对初学者特别有吸引力。表格填写形式的菜单可以形象地指定当前的选择，进一步减少了用户的记忆负担，而且需要当前选择时还允许快速输入。另一方面，菜单限制了可供选择的选择集的大小，而其他对话形式却没有这种限制。

命令语言是一种传统的人机交互方式。这种技术可容纳大量的选择集而且比较容易扩展（只要添加另一个命令），而且对擅于击键的熟练用户来说也相当快。学习时间是它的一个主要的缺点，击键技能是第二个制约因素。使用命令语言发生的错误要比使用菜单多，主要是由于击键错误和回忆错误。

自然语言对话常被认为是交互系统的最终目标：如果计算机能理解我们在日常生活中用英语写或说的命令，那么每个人都会使用它们。然而，目前大词汇量的语音识别器必须单独训练以识别特定人的声音；它们也犯错误，必须通过某种方式来纠正。对于长句子，也很麻烦。由于自然语言不一定在应用程序所处理的命令集内，因此也常常含义不清，使用自然语言界面的用户可能会发出一些不能满足的请求，从而导致用户的挫折感，并导致系统的错误执行。

这个问题可以通过限制领域（从而是受限词汇）的自然语言系统来克服，这样，用户熟悉系统的功能，从而不会发出一些不合理的请求。绘图程序和操作系统就是这类系统的例子。

然而，在认为自然语言交互是最终目标的想法中有一个基本缺陷。如果这个想法是正确的，我们就会满足于单纯依赖电话和/或键盘进行彼此之间通信。正因为这个原因，在交互式图形应用程序中自然语言的语音输入很适合和其他对话方式结合起来使用，如允许语音和手的交叉使用来加快交互速度。毕竟，这才是我们在真实世界中的工作方式：我们指着物体并且谈论着它们。这一概念在十年前用于操纵对象的“Put-that-There”[BOLT80；BOLT84]程序系统中就做了引人

注目的演示。在这个系统中，用户通过指着物体说“Put that”，然后指向另一个地方说“There”来移动它。最近在一个VLSI设计程序中，有关用命令的声音输入结合鼠标对物体和位置的选择的研究显示，使用这种交互方式的用户的速度可以比仅使用鼠标和键盘的要快60%[MART89]。

*问答对话*是由计算机开始，用户的响应限于一组已知的答案。用户可以用键盘进行输入，给出任意一个答案。如果已知答案集比较小，问题中可包括所有可能的回答；用户可用菜单选取代替击键作为一个用户回答的方式。在这种限制下，问答对话变成了一系列的菜单选择。这种对话形式常见的一种缺点是不能回退几步来纠正某个回答。这种形式所隐含的另一个更常见的问题是上下文问题：用户只能用以前和当前的问题来辅助解释当前问题。相比之下，使用填表方式对话，用户就可以看到需要输入的所有域，而且能很快回答，例如，地址中的公寓号码是在街道地址域中，还是在单独的公寓号码域中。

表9-1比较了几种用户界面对话形式。有关这些风格优缺点的更为广泛的讨论可参见[SHNE86]。

表9-1 七种用户界面形式的比较

	所见即所得①	直接操纵	菜单选择	表格填充	命令语言	自然语言	问答对话
学习时间	low	low	med	low	high	low	low
使用速度		med	med	high	high	med	low
出错响应	low	low	low	low	high	high	low
可扩展性	low	low	med	med	high	high	high
录入能力需求		none	none	high	high	high②	high

① 由于“所见即所得”必须附带有某些输入命令的方法，而不是一种完整的界面风格，因此表格内有几个空白区段。
② 假定是键盘输入；没有语音识别器输入。

9.3 一些重要的设计问题

在本节中，我们将描述一些确保设计中好的人的因素的原则：一致性，提供反馈，减少错误发生率，提供错误恢复，容许多种熟练级别，并且减少记忆。对于一个成功的设计，必须满足这些原则，但还不够。有关这些以及其他原则的更多讨论见[FOLE74；GAIN84；HANS71；MAYH90；RUBE84；SHNE86]。

399
|
403

9.3.1 一致性

具有*一致性*的系统就是一个概念模型、功能、序列和硬件绑定都是统一的、遵循一些简单规则的系统，因此它可减少例外和特殊情况的发生。一致性的基本目的就是允许用户由系统的一个方面推广到其他方面。一致性还有助于避免由于系统不能按合理的和逻辑的方式运作时而使用户产生的挫折感。取得一致性的最好方法是，对整个系统进行精细的自上而下设计。

以下是一个用户界面输出部分有关一致性的简单例子：

- 通常使用同样的编码方法。用相同的方式编码颜色信息，就如红色总是表示停止，而绿色总是表示通行。
- 系统状态信息在固定的逻辑位置显示（虽然不一定是物理上的）。
- 菜单选项经常显示在菜单里相同的相对位置，这样用户就能下意识地选取要选的菜单项。

输入部分一致性的例子如下：

- 键盘字符，如回车、制表、换行和空格，通常具有同一功能并且只要是文本输入就可使用。
- 全局命令可以在任何时候进行激发，如 Help、Status 和 Cancel。

- 提供给系统的通用命令，如Move、Copy和Delete，可供任一类对象使用，且结果是可预知的。

但是，我们应该记住Emerson的看法，“一致性不好的界面就像是缺乏思想的怪物”[EMER03]。一致性可能与其他设计目标相冲突。例如，如果拖动一个文件图标到垃圾箱表示删除该文件，那么拖动一个文件图标到电子邮箱时会发生什么呢？为了与拖到垃圾箱的情况保持一致，是应该发送该文件然后将其删除吗？或者是应该发送文件的一个拷贝吗？如果拖动一个文件到打印机图标来打印该文件，那么是应该打印该文件然后将其删除以便与拖到垃圾箱的情况保持一致吗？的确，在后面这两种情况下，不应该删除该文件。作为一个较高的设计原则，最小惊讶定律表明正常且合理地考虑用户所想做的事情，比单纯保持一致性重要。

图9-9显示了状态图是如何有助于区分不一致性的。在这里我们可以看出帮助仅来自于移动状态，而其他状态没有。一旦激活了一个动作序列，系统采用一种混合的策略让用户改变他们的想法：移动和删除状态中用一个取消[⊖]命令，而旋转状态有一个撤消命令。对象/操作改变的顺序是：对于移动和删除的序列是先操作后选定对象，而对旋转则相反。反馈策略也是混合的：对于移动是动态的，而对于旋转则是静态的。

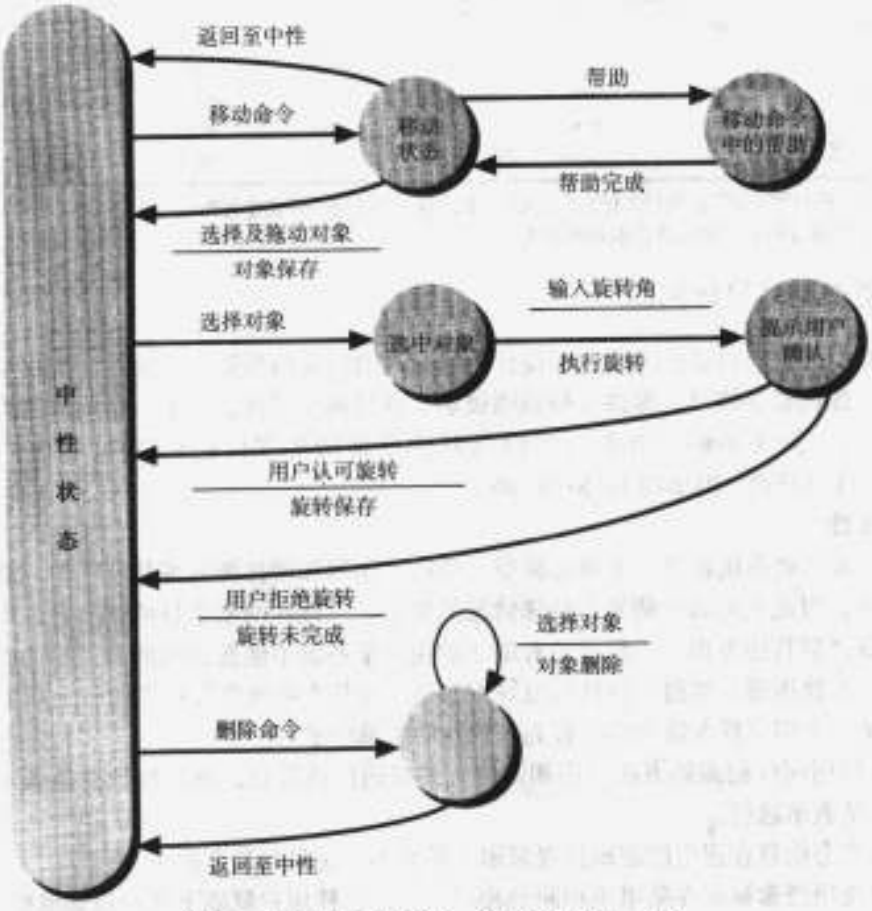


图9-9 用户界面的状态图，其中的语法不一致

⊖ 图中未标明“取消”，而用“返回”。——译者注

Reisner用实验展示了一个可凭直觉预期的结果：给定两个功能相同的用户界面，对于其中一个具有较简单的语法结构的界面[REIS82]，新用户犯的错误较少并且学得也较快。这样，要实现的另一个有用的设计目标是减少不同语法结构的数目。

在功能级别上，一致性要求使用通用命令，这些命令可用于尽可能广的范围。例如，在前面讨论过的房间布置程序中，应该以同样的方式来移动椅子和桌子；在各种应用程序中打开、删除、保存文件，应该采用同样的通用文件操作命令。

9.3.2 提供反馈

您是否试图和一个不苟言笑的同伴说话，这个同伴仅在要求做出回答的时候才响应？这是一个让人灰心的经历，因为很少有或几乎没有迹象表明这个同伴理解您正在说的一切。同样，在与计算机对话过程中反馈也是必要的。区别是在和人的正常交谈过程中，参与者提供了许多反馈方式（手势、身体语言、面部表情、视线接触），但没有任何一个参与者是有意识这样做的。相比之下，计算机却只有很少的自动反馈（仅仅是打开电源的灯和风扇的转动），所以必须计划并编写所有的反馈。

404 ~ 405

大概可以在三种层次上提供反馈，分别对应于用户界面设计的功能层、序列层和硬件绑定（语义、语法和词法）层。设计者必须有意识地考虑每一个层次并且明确是否应有反馈。如果应该有，那么用何种形式。反馈的最低层是硬件层。用户使用输入设备的每一个动作都应立即执行并有明显的反馈：例如，应显示从键盘键入的字符，应由光标响应鼠标的移动。

当输入语言（命令、位置、目标等）作为一个单元（单词）被系统接受时，在顺序设计层次上的反馈就应该发生。如果选中的对象或菜单命令以高亮度显示时，用户就知道动作已经接收（例如，“words”被理解后）。以类似形式的反馈来提示下一个输入，功能键在按下的同时被加亮，文本（话语）在输入的同时就有文本输出。

当输入一个完整的并得到很好组织的句子时，就应出现另一种形式的序列反馈。仅当命令的处理时间多于一两秒钟时，才需要确认接收到正确的命令。

只有当操作的执行时间达几秒钟以上时，才需要另一个类型的功能反馈——显示计算机正在工作。（如果没有这种反馈，用户就会对计算机表示不满，甚至对程序的设计者表示不满！）这种反馈可以采取很多形式；特别吸引人的是用表盘或进度指示尺来指示完成的百分比。用户可以了解计算机执行所需的时间。在实验中，Myers发现用户非常偏爱这种指示器[MYER85]。

功能反馈中最有用并且最受欢迎的形式是计算机告诉用户已完成所需要的操作。这可以用一种新的或修改过的显示方法显式地给出操作的结果。

区分问题域和控制域的反馈是比较有用的。*问题域反馈*关心的是正在操纵的实际对象：它们的形式、位置以及存在状态。*控制域反馈*必须处理用于控制交互式系统的机制：状态、当前值和缺省值、菜单以及对话框。

如果用户只能看到一大张视图的一部分就需要问题域反馈，这样用户才知道正在显示的是整个视图的哪一部分，图9-10所示的为其中一种方法。这种方法使用两个显示区，分别用于全局图及细节部分，从而使显示更为有效。不管当前使用哪个显示区，全局视图中的矩形表明正显示的细节是全局视图的哪一部分。拖

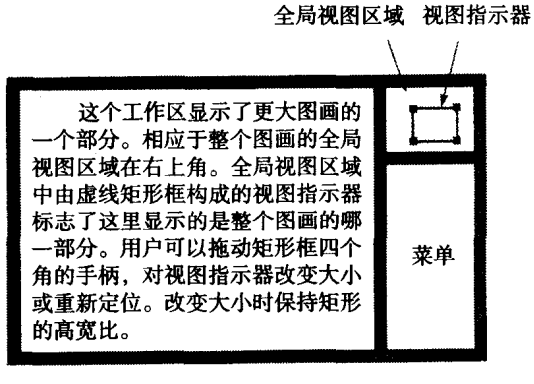


图9-10 使用视图指示器来显示全局视图，并且控制工作区中显示的内容

动或重置全局图的矩形，通常会引起细节图内容的平移和缩放。图9-11所示为如何通过滚动条来反映反馈的增长量。图9-12中所示为MacPaint采用的引导观察者的另一种方法。



图9-11 四种不同层次的滚动条反馈，滚动条显示从无 a) 到冗余 d)

控制域反馈的一种重要类型是当前设置。当前设置可以在反馈区域内显示，如果用户选中了设置中的菜单或其他工具，那么它们通常能显示出来，如图9-13所示，当前设置就可以在那里显示出来。在图8-14中描述的弹出式菜单也表示当前的设置。

反馈的位置是很重要的。有一种自然的趋势就是为反馈和错误信息指定一个固定的屏幕区

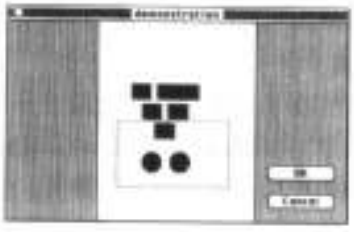


图9-12 MacPaint中的全局视图方式。由于屏幕非常小，全局视图随着正常的、更详细观察区域的选定而改变。用户将矩形的虚线框拖到想要观察的区域，并且选择“OK”来察看已圈区域的详细视图。(Claris公司版权所有，1988。)

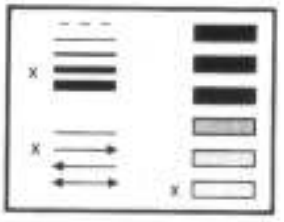


图9-13 在这个图形属性的菜单中，x表示当前设置。如果该菜单一直可见，那么用户就会有持续性的反馈。如果不是，那么应该使用一种更紧凑的持续反馈形式

域。然而，由于用户的视线必须在工作区和消息区之间移动，所以可能破坏视觉的连续性。事实上，用户经常不去注意在这些固定区域中的消息。增加声音反馈可以解决这个问题。

特别有效的方法是将反馈放置于用户正在看的地方，通常在光标或接近光标处。Tilbrook的Newswhole系统[TILB76]就是最早采取这种思想的方法之一：它用坐佛的符号鼓励用户在计算延迟期间保持耐心，用朝下的手指符号表示错误。

406 - 408

9.3.3 减少错误概率

不要让用户因为出错而伤心是阐述这一目标的另一种说法。例如：

- 不提供会产生“非法选择，命令无效”消息的菜单选项。
- 如果没有能删除的东西，就不允许用户选择删除。
- 如果对象不是文本字符串，就不允许用户试图改变当前选择的字体。
- 如果剪切板为空，就不允许用户粘贴。
- 当没有选取要拷贝的内容，就不允许用户拷贝。
- 当选中的对象不是曲线时，就不让用户选取曲线光顺命令。

在所有的这些情形中，系统应该禁用不可用的条目，并且通过改变菜单条的外表来提示用户；例如，将它置成灰色而不是黑色。

这些都是上下文有关的例子：用户只能使用在当前环境及方式下有效的命令。系统应该引导用户在给定的环境下工作，在这样的环境下，不能让用户做系统环境不容许做的事。

这个目标的另一个方面就是避免副作用，这些副作用是系统无法引导用户得到预想的结果。典型的副作用是“打印命令”也将所打印的文件给删除了。副作用是由不良设计或与用户的无效交互引起的。

9.3.4 提供错误恢复

我们每个人都犯错误。试想一下如果在您的计算机键盘上没有退格键会怎样！由于您在敲键时需要更加小心，所以您的效率会受很大的影响。有充足的实验证据表明：如果能随时纠正人们的错误，那么就会有更高的效率。如果有好的错误恢复机制，用户就会自由地探索未知的系统功能而不用担心失败。这种自由鼓励探索性的学习，是了解系统特征的主要途径之一。我们讨论四种类型的错误恢复：Undo（撤消）、Abort（中止）、Cancel（取消）和Correct（纠正）。

最严重的错误类型是功能性的：用户错误地激发了一个或一系列命令，并且得到预料不到的结果。这就需要一个撤消（Undo）命令来回退该命令的结果。有两种Undo类型：单级和多级。单级Undo能回退到最近执行的命令。Undo本身是个命令，因此两个连续Undo命令的第二个Undo纠正了第一个Undo，将系统返回到这两个Undo命令之前的状态。

相比之下，多级Undo命令是在前面一些命令的堆栈上操作的。累加起来的实际命令数目（也就是能够取消的数目）是与实现有关的：在一些情况下，保存了自交互过程开始以来的所有命令。如果Undo是多级的，就需要Redo命令，这样，如果用户在命令堆栈中返回得太多，409
就可以重做最近的Undo命令。Undo和Redo都没有存入堆栈中。关于Redo的几个棘手的问题在[VITT84]中有讨论。对于单级类型，Undo和Redo之间是互斥的，即在给定的任何时间不能同时存在；而对于多级类型，Undo和Redo可以同时存在。

用户在理解Undo命令的范围上经常需要些帮助（例如，该命令能撤消多少工作）。对窗口命令（如滚动条）和应用程序命令等是否可用Undo命令也经常搞混。如果在菜单中有Undo命令，则菜单栏的文字会指示可以撤消什么，并且给出的不仅仅是“撤消”，而可能是“撤消复制”或“撤消删除”，Kurlander和Feiner开发了一种图形方式，用于显示用户动作的历史记录和Undo的范围，参见彩图I-23[KURL88；KURL90]。

撤消的形式有时候被提供为显式接受、显式拒绝步骤。命令执行后，其结果就显示在显示器上，用户在进行其他操作之前必须接受或拒绝该结果。该步骤增加了用户在完成一个任务时的动作数目，但促使用户在确定接受危险操作之前再三考虑。然而，通常还是需要Undo命令，因为常常只需要用显式的操作来拒绝一个命令的结果。当键入下一个命令（而不是Undo命令）时，就已隐含地接受了该命令。于是，我们称撤消为一种隐式接受、显式拒绝策略。

不管撤消是如何提供的，它需要额外编程来实现，特别是对数据结构有较大改变的命令。

一种比较容易但不十分令人满意的Undo命令的替换方式是，要求用户清楚地确认该命令不能撤消，这通常用于文件删除。

在一个命令正在执行的时候，用户可能会意识到发生了功能级的错误，这表明需要一个中止（Abort）命令来提前终止当前正在执行的命令。类似于Undo，Abort必须恢复到所中止命令开始之前的系统准确状态。事实上，可以认为，本质上Abort和Undo是相同的：它们都回退最近指定的功能级动作。用户界面设计可以使这两种动作以同一个名字存在。

当用户在中途将通过指定信息来执行命令过程时，会产生一些较为平常的错误，而且会说“啊呀，我其实并不想这样做”。设计差的界面就会让用户只能继续该命令而没有别的选择，在此之后使用Undo或Abort（可能的话）来恢复。设计好的界面会让用户使用取消（Cancel）命令退出这种情况。在表格填充对话中，这种Cancel命令常常用在表格上，如图8-32所示。注意，Cancel也可认为是一种特殊的Undo命令，它使系统回到当前命令以前的状态。

在一类不太严重的错误中，用户可能想纠正某个命令所需信息单元中的一个。所使用的对话形式将决定了进行这种纠正的难易程度。对于命令语言的输入，可通过对错误条目进行多次退格键操作来纠正，然后重新输入正确的信息和被删去的信息。如果系统有“行编辑”能力，那么就可以将光标移动到出错的地方，而无需删除中间的信息。表格填充形式将允许类似的简
410 单纠正，而问答式的和菜单对话形式的就不是那么简单了。第8章中讨论的动态交互技术提供了一种错误恢复形式：比如，被拖动对象的位置改变容易。

9.3.5 容许多种熟练级别

许多交互式图形系统需要为广泛的用户设计；包括从新的、完全没有经验的用户到使用该系统很长时间的用户。使系统适合于各种熟练级别的方法有快捷键、提示、帮助、可扩展性以及复杂性隐藏。

新用户通常习惯于菜单、表格以及其他能提供较好提示的对话形式，因为这种提示可告诉用户怎么做从而容易学习。然而，较熟练的用户更注重使用速度，这需要使用功能键和键盘命令。这种快速交互技术叫作*快捷键*。典型的快捷键在前一节已出现过，比如，基于鼠标的菜单选择命令用一个字母命令代替。Sapphire窗口管理器[MYER84]将这种思想进一步延伸，提供了三种而不是两种方式激发命令：指点窗口工具栏的不同区域并且点击不同的鼠标键、标准的弹出式菜单和键盘命令。

Macintosh系统在一些菜单命令中使用快捷键，如图8-13所示。另外一种方法是将菜单命令编号，这样可从键盘键入数字，或者用光标来选择命令。还可以键入命令名或者是它的缩写。

一种最快的快捷键是多次点击鼠标键。例如，Macintosh用户可将光标置于图标上，点击鼠标键来选中一个文件（用图标代表）。接下来，自然是打开文件，可以用菜单选择、快捷键或双击鼠标键来打开文件。采用双击鼠标键比另外两种方法要快得多。从应用程序内部来看，可以使用图9-14中所描述的另一种方法来打开文件。可以用点击或键入方法在对话框中选取
411 文件名。如果文件名是键入的，那么只要键入

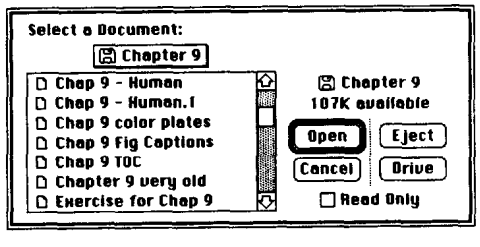


图9-14 从Macintosh程序中打开文件。用户键入Open命令，通过菜单选择或者通过组合键，使对话框出现。选中的加亮文件可用“打开”按钮或敲回车键打开。用户可用光标拾取，或者键入文件名的部分或全部，来选中（加亮）一个新文件。因此，用户可通过键盘输入、组合键、输入部分文件名和回车键来打开一个文件。（屏幕图形苹果公司版权所有。）

不至于引起歧义的足够字符数即可。双击文件名也可立即打开该文件。

快捷键的另外一种形式是提供命令行输入以代替其他形式。当用户使用熟练后，他们就会越来越多地使用命令行。在用其他方式输入时相应地显示其命令行形式，有助于这种转变。

与确认指定用户操作的反馈不同，*提示*的目的是建议下一步做什么。用户越熟练（特别是当提示过分炫耀并且减慢了交互过程或者使用了屏幕的较多区域时）越不需要提示。许多系统为用户提供了好几级可以控制的提示，没有经验的用户可“手把手来引导”，而有经验的用户则不用提示就可继续下去。

提示可以有许多形式，最直接的是显示一条消息，明确地解释下一步应做什么。例如“请指定位置”。语音合成器可以给用户发出听觉指令。也有一些更有意思的提示形式。在一个功能键框上，给出了可供选择的按钮的说明。当需要输入位置时，就显示一个突出的跟踪十字交叉或光标；一个下划线形式的闪烁光标，表示应该输入一个文本串；当需要一个数值时，就显示一个尺度或刻度盘。直接操纵的图形界面隐含地给出一些提示，可供操纵的图标就是提示。

*帮助*工具允许用户得到一些额外的信息，如系统概念、典型任务、各种命令及其使用方法以及交互技术等，理想情况下，可以在交互对话的任何时刻请求帮助，而且使用同一种机制。从帮助返回时，应该将系统置于与激发帮助前相同的状态，而且帮助应该与上下文有关。例如，当系统正在等待一个命令时，如果激发帮助，那么就应该显示在该状态下的命令列表（但使用菜单或功能键时，这就不需要）。与命令名相随的帮助命令应获得有关该命令的更多信息。当输入命令的参数时，如果请求帮助，那么就应该提供该参数的细节。“进一步帮助”命令应给出更详细的信息并且可能通过在线文档允许更一般的浏览。Sukaviriya[SUKA88]开发了一个系统，用当前上下文作为出发点，用动画的形式向用户动态展示如何完成一项任务。基于超文本系统的一些帮助功能允许用户在各种帮助主题中指向复杂的链接集。

激发帮助的一种简单方式就是点击在屏幕上所需帮助的实体，该实体可以是菜单项、状态指示器（该帮助应该解释这个状态和改变它的方法）、窗口标题、滚动条或者是以前创建的应用程序对象（帮助应该解释该对象是什么，可以应用什么样的操作）。但是，这种方法仅可用于可见对象，而不能用于较抽象的概念，也不能用于必须由一系列命令执行的任务。

即使有提示和菜单的显示，帮助功能也是适用的，因为它给用户提供了比简短提示更详细的信息。即使熟练用户也会忘记一些细节的东西，尤其是在一个大而复杂的应用中。 412

专家系统正开始集成在用户界面中以提供帮助，这些帮助不仅对上下文敏感，而且可根据不同用户的情况进行定制。随着新用户和系统之间进行彼此交互和学习，这样的定制是由系统自动创建的，就像老师了解他的学生并且为他们制定计划一样。

使用户界面*可扩展*意味着可以让用户通过组合已有命令给界面增加额外的功能。关键是能保存并重新执行用户的操作过程。宏定义功能特别有吸引力，那就是可以将用户动作自动保存在一个跟踪文件中。为了创建一个宏，用户可以编辑跟踪文件来标识该宏的开始和结尾，并且可以用参数来替换该宏的文字记录，还可以命名宏。几种商用的应用系统，如Ashton-Tate的Full Impact，就有这样的功能。Olsen已经开发了一个相当复杂的原型系统[OLSE88]。

*复杂性隐藏*使新用户只需要学习基本的命令就可以开始做生产性的工作，而不需要指定繁杂的选项、学习不经常使用的特定命令或经历复杂的开始过程。另一方面，功能强大的系统需要有许多命令，而且这些命令还经常有多种变化形式。解决这一困境的方法就是设计整个命令集，使得它有一个小的“初学者工具箱”。缺省值（当前设置或最初设置）遵循最小惊讶定律，通常也很有用。

例如，图表制作程序应该允许用户用饼状图显示数目，并且可以马上看到效果。如果用户

对图表布局的一些细节不满意，他应该能够修改饼图的半径，改变每个扇形条的颜色或纹理，添加注释，改变数据值的文字颜色和字形，或者改变每个扇形条上的数据值位置。但是在用户创建该图表时，这些显示模式不需要一一指定。

另一种设计策略就是设计复杂的和高级的命令，这些命令只能通过键盘命令或功能键使用。这种方法使菜单条较小，并且系统比较简单也不会令人生畏。此外，可以提供两个或更多的菜单条，每个菜单条都包含多个命令。

隐藏复杂性的另一种途径是使用控制键修改其他命令的含义。例如，Macintosh窗口管理器通常激活用户选中的窗口并且拖动到新位置。更熟练的用户可以通过按下控制键（在Macintosh上称为命令键，以适当照顾计算机的高手）重定位一个窗口，而不用激活它。对新用户就根本不告诉这个特征。

9.3.6 减少记忆

界面设计有时要求不必要的记忆。在一个图形设计系统中，对象是用数字引用的，而不是字母数字式的名字。为理解其中的含义，我们可以想像一个交互式操作系统，其中的文件名是
413 数字式的，那就不需要对文件名的记忆和有助于记忆的学习工具，仅需要死记硬背或使用笔录来帮助记忆。当然，直接选取所显示的对象或图标可进一步减少记忆。无论如何，唤起用户的识别要比回忆重要得多。

在一个交互式图表系统中，如“绘制按年的净收入”这样的命令，就会在同一个坐标系上产生一个年总收入和一个年净收入的趋势图。要控制曲线类型，较好的方法是使用如“Linestyle net dash”的命令，用虚线绘制净收入趋势图。不幸的是，实际命令是“Linestyle 3 dash”的形式，“3”指的是在最近的“绘制”命令中命名的第三个变量——在这种情况下就是net，由于最近的绘制命令一般不在屏幕上，所以用户必须记住参数的顺序。

一些帮助系统完全遮住了工作区，为了解释帮助信息，用户必须记住上下文。这样，一旦用户理解了帮助信息，他就必须记住返回到原来需要帮助的地方。窗口管理器解决了这个问题，帮助信息在一个窗口中，而应用程序在另外一个窗口中。

9.4 模式和语法

通俗地讲，模式（mode）是所有用户交互任务的可以执行的一个子集所处的一种状态或状态的集合。模式的例子如下：

- 一种命令集的状态，这些命令仅可用于当前被选中的对象。
- 执行另一种操作之前必须完成一个对话框的状态。
- 在文档准备系统中用于绘制图形时的状态。它可使用不同的程序来编辑文本、绘制图形和展示文档。
- 由当前数据簿区段决定的可用命令的状态。

这样，模式提供了系统和用户的一种操作环境。

有两类模式：有害模式和有用模式。正如Tesler[TESL81]以及Smith和他的同事[SMIT82]所讨论的那样，有害模式会持续一段时间，不与任何别的对象相关联，对用户也不可见，并且不服务于任何有意义的角色。有害模式使用户思路不清，用户陷进去了，就出不来了，或者用户会忘记他们在哪种模式下，并且会试图激发不存在的命令，有可能产生错误。影响用户工作效率的模式是非常有害的。

在另一方面，有用模式缩小了对下一步要做内容的选择，这样，提示和帮助就会更具体，

并且菜单会更短，从而操作起来更容易。组织较好的模式结构能降低对用户记忆的负担，而且有助于将有关该界面的知识组织到基于该模式的分类中。有用模式能提高用户的工作效率。

有用模式明确地指示了当前的模式，提供了对现有命令的反馈，并且包括了从现有模式退出的一种简单、明了、快捷的方法。窗口管理器提供了可见度高的模式，因为每个窗口代表一个不同的模式；窗口之间的激活转换影响了模式的切换。 414

靠提高肌肉紧张状态来维持的模式是没有生命力的。在第8章讨论过鼠标“按下-动态反馈-弹起”这样的交互技术，它是通过在按鼠标时的肌肉紧张过程来使用户了解模式的[BUXT86]。

命令语言语法对用户界面的模式结构有一个主要影响。传统的前缀语法：命令，参数1，……，参数*n*，会将用户锁进一种和指定命令一样的模式：只能输入参数，可能还要按规定的顺序。错误纠正机制（9.3.4节）在这里就特别重要，要不然用户在命令的参数选取错误时，必须继续这种繁冗的参数指定过程。

前缀语法的困难之一可以由一个称为分解因子或正交化的过程来解决。设置命令的每个参数为当前值；参数也可能有一个初始（缺省）值。例如，考虑下列非因子的命令语法，其中一个初始的大写字母表示一个命令，小写字母表示参数：

```
Draw_line        point  point  line_style  line_thickness  line_intensity
```

我们可以把属性说明分解成三个独立的命令或分解成一个总的属性设置命令。因此，用户要经过下列序列：

```
Set_attributes   attribute_values   {只有在当前的属性值不适用时}
Draw_line        point point
```

通过引入当前点(current point)的概念，我们还可以对这一序列做进一步分解。当前点在调用Draw_line命令前由用户选取：

```
Set_attributes   attribute_values   {只有当当前的属性值不适用时}
Select_point     point              {选取起始点}
Select_point     point              {选取终止点}
Draw_line                           {Draw_line命令已无参数，所有参数都被分解出去了}
```

不一定需要完全不带参数的命令，例如，在这里，指定一些点，然后告诉系统用线段将它们连结起来，这样就失去了进行画橡皮筋线的能力。

在什么样的应用程序中用户倾向于连续几次执行一个命令，每一次都在不同的对象上呢？这种情形建议用户使用命令方式语法，命令输入一次，后面可以有任意数目的参数组。例如，删除对象的语法如下：

```
Delete_object                       {建立Delete_object命令方式}
object
object
object
⋮
任何命令                            {建立新的命令方式}
```

415

Delete_object建立了删除模式，随后选定的每一个对象都可以删除，直到选定其他命令为止。注意，命令方式意指前缀语法。

若把命令的参数分解出来，其语法为：

```
Delete_object    object
```

那么我们引入当前选中对象（currently selected object, CSO）的概念。我们也需要给一个新的命令Select_object创建一个方式，其中具有CSO；该CSO可由Delete_object命令操作：

```
Select_object        object        {没有对象被选定时使用，或者需要另一个CSO时使用}
Delete_object                      {没有参数——对象已被分解出去了}
```

参数分解已创建了一个后缀语法：对象首先通过选取来指定，然后给出命令。如果用户正在使用指点设备，这一技术特别有吸引力，因为一种自然的倾向是，在说做什么之前先指着某件事物；顺序倒过来就很不自然了[SUKA90]。

CSO是一个有用的概念，因为用户可以在一个对象上执行一系列的操作，然后继续另外一个。此外，Select_object命令通常是隐含的：用户简单地点一下要选取的对象并且点击一个按键。这意味着参数分解不需要创建额外的步骤。

我们可以回想一下，命令方式有一个前缀语法。如果选取了后缀语法，那么它的优点能保持下来吗？答案是肯定的。如果一个Repeat命令（即重复最后一个未选取的命令）可以很方便地执行，比如说通过按下多键鼠标上的一个键。如果那样的话，删除多个对象的用户操作序列可以如下：

```
Select_object        object        {建立CSO}
Delete_object                      {删除CSO}
Select_object        object        {一个新的CSO}
Repeat                             {一次按键删除}
Select_object        object        {一个新的CSO}
Repeat                             {一次按键删除}
⋮
```

将这个与可用于命令方式的真实操作序列进行比较：

```
Delete_object
object
object
object
⋮
```

假设Select_object仅仅需要指点和按键，那么相对于使用命令方式，需要在对象方式中使用
416 Repeat_last_operation（重复最后一次操作）的额外步骤就是每次删除时单击鼠标键。

另外一种可选择的序列是任意的自由形式语法（无定式语法），它允许不同语法的相互混合。不管用户是先指定一个对象然后指定动作，或者是先指定一个动作再指定一个对象，动作总是在对象上执行。例如，

```
Set_attributes       attribute values
Select_object        object1           {应用到对象1的属性}
Set_attributes       attribute values
Select_object        object2           {应用到对象2的属性}
Select_object        object3
Set_attributes       attribute values  {应用到对象3的属性}
```

注意，这种语法结构不能假设一个当前选中对象；如果这样做的话，那么第二个Set_attributes命令就会立即作用于object 1，而不是object 2。

如果增加了Do_it命令，那么在当前选中对象方式中可以使用自由形式语法，这样用户可告诉系统在当前选中的对象上执行当前命令，但是，该命令就如同下列序列所描述的那样，增加了另外一个用户操作：

```
Select_object        object            {建立CSO}
Set_attributes       attribute values  {建立当前命令}
Do_it                                  {CSO接收新的属性}
Copy                                   {建立新的当前命令}
```

```
Select_object          object          {建立一个新的CSO}
Do_it                                  {拷贝CSO；现在副本就是CSO }
Do_it                                  {CSO被拷贝，新的副本就是CSO}
```

这种自由形式语法的另外一种形式是与方式相关的语法，比较下列两个序列可知，Set_attributes命令是与模式相关的：

```
Set_attributes      attribute values      {在此点上无CSO}
Select_object       object
```

以及

```
Select_object       object                {建立CSO}
Set_attributes      attribute values
```

模式相关性是更一般的上下文相关性的一种特殊情形，利用上下文相关性，命令的作用依赖于当前的上下文。在前面的第一个序列中，当使用Set_attributes命令时没有CSO，属性值就变成全局缺省值，在创建新对象时，也会起作用。在第二个序列中，有CSO，属性值作用于CSO，但不改变全局缺省值。也就是说，CSO方式的存在与否决定了该命令的作用。该技术创建了一组功能强大的命令而没有增加任何显式的新命令；然而，用户必须有方式反馈，以便知道该命令是如何作用的。还有，一些用户会被这种方法所迷惑，因为在理解这些规则之前，它看上去很不一致。 417

分解因子的一般概念很重要，其原因有以下几点。首先，新用户不必关心具有缺省值的已分解参数，因此提高了学习的速度。已分解参数的值不需要指定，除非缺省值不可接受，因此提高了使用的速度。将对象从命令中分离出来，创建了CSO的概念，这对具有指点设备的交互式图形学而言，是一个自然的概念。最后，分解因子减少或撤销了由多参数的前缀命令创建的短期命令。分解因子（参数）已集成到用户界面的设计工具中，这样设计者就可以要求分解指定的参数；[FOLE89]自动引入了必要的辅助命令（Select_object）。

在CSO概念上，有几种变化形式。第一，当对象创建时，如果已经有一个CSO就没有必要变成CSO，同样地，当删除CSO时，其他一些对象（最近的CSO或与CSO接近的对象）就会变成新的CSO。此外，可以使用由多个选中对象组成当前选取集（CSS）。

9.5 视觉设计

人机界面的视觉设计影响用户对界面的初始印象和系统的长期可用性。视觉设计由界面的所有图形单元组成，包括总体屏幕布局、菜单和表格设计、颜色的使用、信息编码以及信息单元相互之间的放置。好的视觉设计力求清晰性、一致性以及富有吸引性的外表。

9.5.1 视觉清晰性

如果图像的意义对观察者来说很清晰，那么我们就有视觉清晰性。取得视觉清晰性的一种重要途径是使用信息的视觉组织来加强基本的逻辑组织，要完成该目标有一些基本的视觉组织规则。这些规则具有重要的影响，如一些案例所显示的那样，这在下面的一些例子中将会看到。这些规则被图形设计者使用了好几百年[MARC80]，在20世纪30年代完形心理学家Wertheimer[WERT39]将它们进行了整理编辑。它们描述了观察者如何将单个的视觉刺激组织成比较大的整体形式（这样，术语完形（Gestalt）的字面意思是“形状”或“形式”，隐含着对整体的强调，而不是在组成部分上）。 418

视觉组织规则涉及相似性、接近性、封闭性以及好的连续性。相似性规则是指两个具有共同属性的视觉刺激是属于一起的。类似地，接近性规则是指彼此接近的两个视觉刺激是属于一起的。封闭性规则是指，如果一组刺激几乎封闭了一个区域或者可以被解释为封闭了一个区域，

观察者可以看到该区域。好的连续性规则是指，给定一些直线的接合点，观察者看到这些连续直线段是光滑连接的。

图9-15和图9-16给出了这些规则的一些例子，并且显示了如何将它们中的一些结合起来彼此增强。图9-17显示了在应用这些视觉组织规则前后的样式。在图9-17a中，每个事物都彼此接近，这样，基本的逻辑分组不清晰。相似性（这里，感觉都包含在一个框里）和接近性将样式和选择按钮捆绑到一起，如图9-17b所示。封闭性完成了加框，而框又被标签断开。

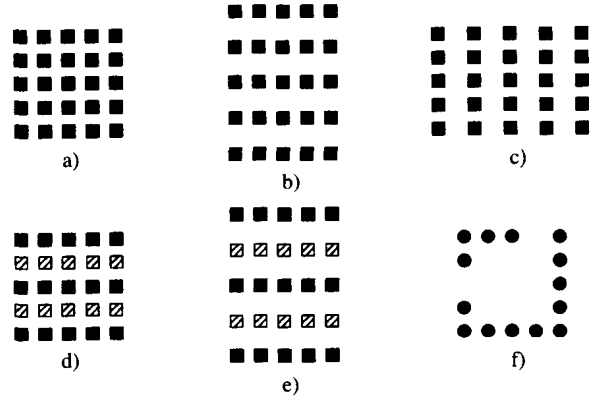


图9-15 完形规则。在a)中，没有区分开这些正方形。在b)中，接近性引起水平分组；在c)中引起了垂直分组。在d)中，相似性引发出水平分组，这种效果被e)中的接近性和相似性的组合进一步加强。在f)中，即使丢失了两个点，封闭性还是引发出一组正方形的图点

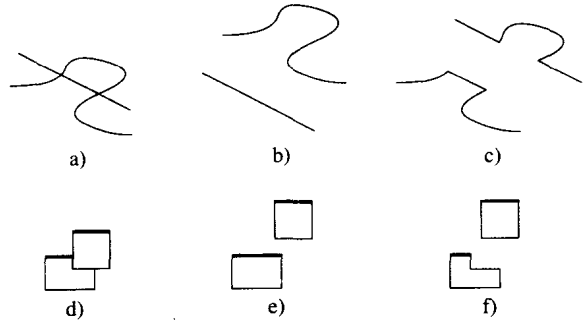


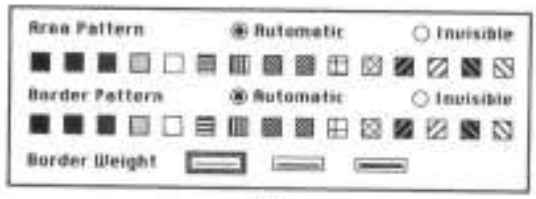
图9-16 更多的完形规则，在a)中的两条相交线可以被解释为如图b)或c)中所示的那样，好的连续性见b)。在更多的应用中，d)中的两个重叠窗口可被解释为e)或f)。好的连续性见e)

419

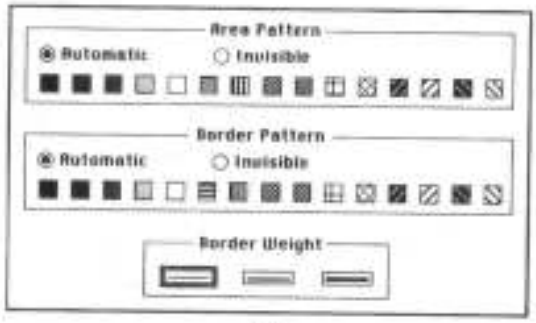
如图9-18所示，应用这些规则来改进菜单组织。最左边的组织在视觉上没有结构，并且几乎隐藏了它的逻辑组织。最右边的菜单使用接近性来形成分组，并且用缩进相似性来显示二级逻辑结构。在图9-19中，用两种不同的方式（相似的排字风格，相似的缩进层次）使用相似性规则来加强逻辑组织。

当忽视或错误使用组织规则时，就会引起错误的视觉暗示，并且会使观察者以为是错误的逻辑组织。图9-20给出了错误视觉提示的例子，并且显示如何用较多垂直间隔和较少水平间隔来校正他们。图9-21a显示了类似情形。

我们回想一下，使用这些原则的目标是通过加强逻辑关系获得视觉清晰性。安排信息的其他目标是，在用户获取任务所需要的各种信息单元时所必须的使视觉移动最小化，以及在屏幕的不同部位移动光标时使手移动最小化。这些目标可能相互矛盾的，设计者的任务就是发现最好的解决办法。



a)



b)

图9-17 a)视觉设计时没有考虑表格的布置。b)用视觉分组和封闭性创建的表格，用来加强表格中视觉元素之间的逻辑关系。(屏幕图经微软公司1983–1989版权所有。经微软公司许可引用。)



图9-18 同一个菜单的三种设计，表现了视觉设计技术的应用

A MAJOR CATEGORY
A LESS MAJOR CATEGORY
AN EVEN LESS MAJOR CATEGORY
AN EVEN LESS MAJOR CATEGORY
THE LEAST MAJOR CATEGORY
THE LEAST MAJOR CATEGORY
AN EVEN LESS MAJOR CATEGORY

a)

A MAJOR CATEGORY
A LESS MAJOR CATEGORY
An even less major category
An even less major category
The least major category
The least major category
An even less major category

b)

A MAJOR CATEGORY
A LESS MAJOR CATEGORY
An even less major category
An even less major category
The least major category
The least major category
An even less major category

c)

图9-19 代表同一种信息的三种设计。a)这种设计没有采用视觉加强，b)这种设计采用了各种层次排字风格(全部字母大写黑体，全部字母大写，大小写混合，较小字母的大小写混合)，c)这种设计增加了缩进，另一种类型的相似性，进一步地将它们联合运用

ATE	BAT	BET
BITE	CAT	CUP
DOG	EAST	EASY
FAR	FAT	FITS
GET	GOT	GUT
HAT	HIGH	HIT

a)

ATE	BAT	BET
BITE	CAT	CUP
DOG	EAST	EASY
FAR	FAT	FITS
GET	GOT	GUT
HAT	HIGH	HIT

b)

图9-20 a)中的列表具有水平方向的逻辑（字母顺序的）组织，但是明显的近似关系引出了一个垂直的视觉组织；在b)中，增强了字母顺序的组织效果

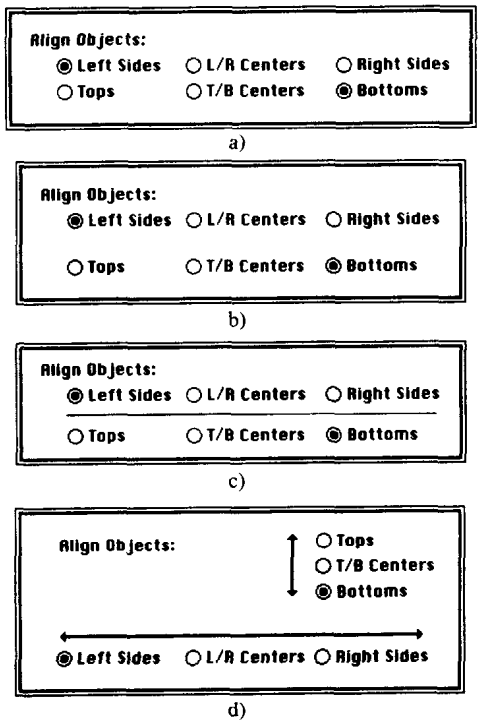


图9-21 校正对象对话框。a)中视觉暗示把两组按钮分成了三组，而不是把三组按钮分成两组；b)中垂直间隔增加，视觉暗示正确；c)中水平线代替垂直间隔，用较小的空间起到了同样的作用；d)中选项做了重新安排，箭头强调了每一组按钮与其相对应的含义间的空间对应关系。（Claris公司版权所有，1988。）

9.5.2 视觉编码

在界面设计中，*编码*意味着在几个不同类型的对象之间创建视觉上的区分。有许多不同的编码技术在计算机图形学中都很常用，如颜色、形状、大小或长度、字型、方向、亮度、纹理、线宽以及线型。编码技术的一个基本问题是确定一种特殊技术能对多少种不同类型进行编码。当引入越来越多的码值时，观察者混淆码值的可能性就增加了。用图例表示每个码值含义可以减小错误率。

许多实验确定了在不同编码技术中需要使用多少编码值，而且还允许几乎无错的代码识别（如果没有图例的话）。对于95%的正确率，最多可以使用10种颜色、6个区域、6种长度、4种

亮度、24种角度以及15个几何形状[VANC72]。当然，码值之间的间隔要适合；参见[VANC72，pp.70-71]中适当的颜色列表。

对用户来说，如果区分不同类型之间的信息很重要的话，那么可适当使用冗余编码：使用两种不同的编码来代表同一种信息。图9-19c就是冗余编码，图9-22表示三重冗余编码。为适应色盲用户，通常冗余地使用颜色的某些其他编码来实现。

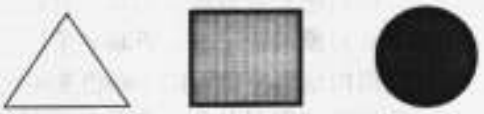


图9-22 利用线宽、几何形状、内部填充图案的三重冗余编码，其中的任一种编码足以区分三种码值

在选取编码前，我们必须知道需要多少级编码。了解被编码的信息是否是主格的、次序的或比率的也是很重要的。主格信息只是简单地指明或命名事物的不同类型，如飞机或轮船的不同类型等。主格信息没有强弱之分。次序信息是顺序排列的，并且有强弱的关系。但是次序信息没有定义度量，在不同的范畴之间没有距离变化的概念。比率信息有这样的度量，例子如温度、高度、重量以及数量等。

对于固定数目的主格编码值，颜色的区分比形状和大小更加精确，在某种程度上甚至比亮度还要精确[CHRI75]。这就说明应该使用颜色来编码，但是我们回想一下，有6%到8%的男性至少有轻度的色盲。正如在第13章所讨论的那样，这不应是一个特别的问题，尤其是在使用冗余编码时。

420 ↓ 423

用于显示主格信息的代码应该是缺乏顺序的，这样观察者在信息中得不到重要性的顺序。不同的形状（例如，在同一坐标轴上绘制多个变量时，正方形、圆、菱形等用于表示数据点）、线型、字型和交叉阴影图案等，都是合适的主格代码。但是多种不同的字型会导致图像混乱。通常可以被接受的一种方案是在一幅图像中不使用多于二或三种字型。同样，不同密度的交叉阴影图案可以创建一个明显的排序（图9-17）。

显示次序信息的代码，可以不断变化但没有必要，不过至少要有一个明显的次序。可用不同密度的线型和区域填充图案，就像文本改变字体大小那样（许多显示仅提供文本大小，在用于比率编码时，利用这些变化就很困难）。对于比率信息和次序信息，当被编码的信息值增加时，代码外观视觉的权重也应该增加。在图9-19中，对次序信息编码，排字层次结构有一个根据分类重要性而增加的视觉权重。

比率信息（如大小、长度或方向等）必须用一个可以连续变化的代码来表示，Cleveland和McGill通过显示实验对象图，研究了几种不同的连续变化的编码，以显示比率信息，这些图的信息相同但编码方法不同。他们发现了在下列的序列中1是最精确的识别编码[CLEV84；CLEV85]：

1) 沿着同一比例的位置。
2) 位置相同，但比率不同。
3) 长度。
4) 两条直线的角度和直线斜率。
5) 区域。
6) 体积、密度和颜色饱和度。
7) 颜色色彩。

类似地，Ware和Beatty[WARE88]发现在对物体进行分组方面，颜色是有效的，但在比率编码时没什么效果。

如果使用颜色来分组菜单项并在工作区域进行编码信息（比如说在VLSL片上区分层或在

地图上区分地理特征），那么用户就会错误地得出菜单中的红色命令仅可应用于工作区中的红色元素这样的结论。类似地，利用某种较亮和某种较暗的颜色，颜色编码不经意间分成了两个逻辑组：较亮对象的一个组，较暗对象的另一个组。前面讨论的相似性规则实际上是编码的核
424 心。所有类似的信息应该用同一码值来编码；所有不相似信息应该用其他的代码值。

量化数据的编码只是更一般的显示量化数据领域中的一部分。当设计数据表示（如条状，饼图以及趋势图）时，许多需要进一步考虑的东西就变得重要起来。这些虽超出了本文的范围，但相当重要，读者应参阅核心参考文献。Bertin[BERT81；BERT83]和Tufte[TUFT83]的书图表丰富，讨论了如何有效地表达量化数据。Bertin系统地分析了视觉编码，给出了如何有效地使用它们及将不同的表示格式分类。Tufte认为应该减少对表和图的修饰，而强调对数据的表达。他还回顾了自从1700年以来数据表示的精彩历史。Schmid提供了其他的指南[SCHM84]。

Mackinlay将Bertin的一些思想连同Cleveland以及McGill的结果一起集成到APT中，APT是一个自动创建数据表示的专家系统[MACK86]。彩图I-24是来自APT的一个例子。我们期望在这一有希望的领域中得到更大的发展。

与编码密切有关的是将观察者的注意力集中到特殊信息上的方法，例如错误或警告信息、当前选中的对象、当前命令、发生故障的设备或在冲突演习中的飞机。现有的吸引注意力的技术是单纯的颜色或形状，闪烁、跳动或旋转的光标，以及反相显示。单纯的颜色比单纯的形状、大小或亮度更吸引观察者的注意力。

可能会滥用吸引注意力机制，一个跳动的光标（即光标的大小不停地在大与小之间连续跳动），确实吸引用户的注意力，但是它也有影响注意力的倾向。当用户正在观察屏幕上的其他东西时，跳动的光标反而更不利于注意力的集中，虽然它仅在边缘处跳动。

量化信息的编码是用户界面设计的另一重要研究领域。Feiner和Seligmann[FEIN85；SELI89]的著作探讨了图片的自动设计，这里的图片说明了动作在三维环境中是如何执行的。根据输入的信息：假定图片是作为通信的，谁将观察它们，基于规则的系统确定要包含的对象、对象的属性和绘制类型，以及要输入到用于绘制图片的三维图形系统中的虚拟相机参数。彩图I-25就是一个自动产生的图片的例子，该图片用于维护和修复应用程序。

9.5.3 视觉的一致性

视觉组织规则和编码的一致应用以及视觉元素到高层图形对象和图标的一致性结合组成了视觉设计的另一重要元素。当然，视觉的一致性是在9.3.1节中所讨论的整个一致性主题中的一部分。

可以将视觉元素认为是图形字母表中的字母，由这些字母结合起来的单词的含义对观察者来
425 说应该是明显的，例如，用于Macintosh应用程序的对话框是由小的图形字母表构造的。图8-16、图8-32、图9-14、图9-17和图9-21都是这些对话框的例子，图9-23是它们的图形字母表。类似地，图9-24是使用小的图形字母表创建图标的方法，图9-25是单元素的图形字母表。



图9-23 在许多Macintosh应用程序中使用的图形字母表。正方形的选择框表示可选，可以同时选取几个。圆形的选择圈，称为“单选按纽”，表示相互之间是排斥的，仅可以选取一个。圆角的矩形框表示可以用鼠标选取的动作。此外，由黑体边界表示的按钮可以用回车键选取，矩形表示可以编辑的数据域。（苹果计算机公司版权所有。）

在单一图像内部以及它们之间必须保持一致性，必须将一致性的一组规则从一幅图像应用

于另一幅图像。例如，在编码方面，虚线的意义在应用程序的各部分之间发生变化是不可接受的。对于位置的一致性，相邻两幅图像或屏幕之间在相同的相对位置应保持相同的信息，这样用户就可以更迅速地定位信息。



图9-24 a)一个图形字母表。b)由该字母表元素结合形成的图标

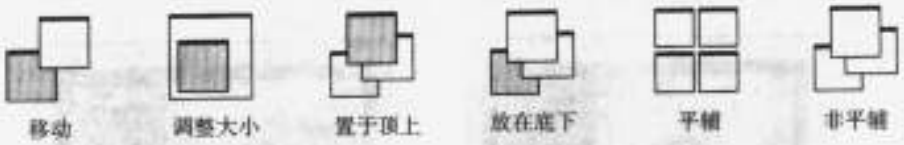


图9-25 各种图标，每个简单图形代表一个窗口

9.5.4 布局原则

屏幕上的每一个元素不仅要认真设计，而且还必须将它们放置在整个上下文中，以便一起工作。三个基本的布局规则是平衡、网格以及比例。图9-26所示为同一屏幕的不同设计。设计a)是平衡的，中心给出了很好的框定，并且将眼睛引向这个区域，设计b)是非平衡的，并且不必要地将眼睛引到该区域的右半边。设计b)还在右上角有一点不规则：滚动条箭头的基线和指针图标没有很好地排列，将眼睛毫无必要地引到这样无意义的非连续性上。

图9-27说明了在不同区域间使用空格的好处，并且描述了网格的概念：在情形b)和c)中，三个区域的边都排列在一个网格上，所以有一种简洁性，一种美感，这在a)和d)中都是缺少的。图9-28进一步强调了不使用网格的致命影响。[FEIN88]讨论了一个创建和使用设计网格的专家系统。

比例主要考虑放置在网格上的矩形区域的大小，矩形两条边长的一定比例要比其他的比例更显得有美感，并且从古希腊和罗马时代以来一直沿用至今。这些比例有：正方形的比例，即1.1；平方根，1:1.414；符合黄金分割的矩形，1:1.618；双正方形，1:2。双正方形特别有用，因为可以放置两个水平双正方形于一个垂直双正方形的边上从而组成一个网格。[MARC80；MARC84；PARK88]中讨论了这些设计规则和其他的一些设计规则。

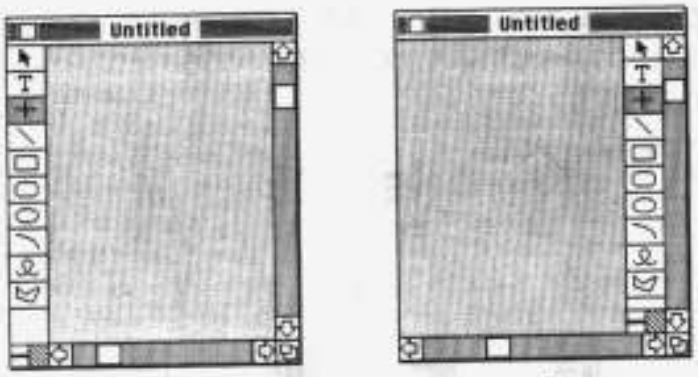


图9-26 两种不同的屏幕设计，设计a)是平衡的，设计b)强调了右边。（Claris 公司版权所有，1988。）

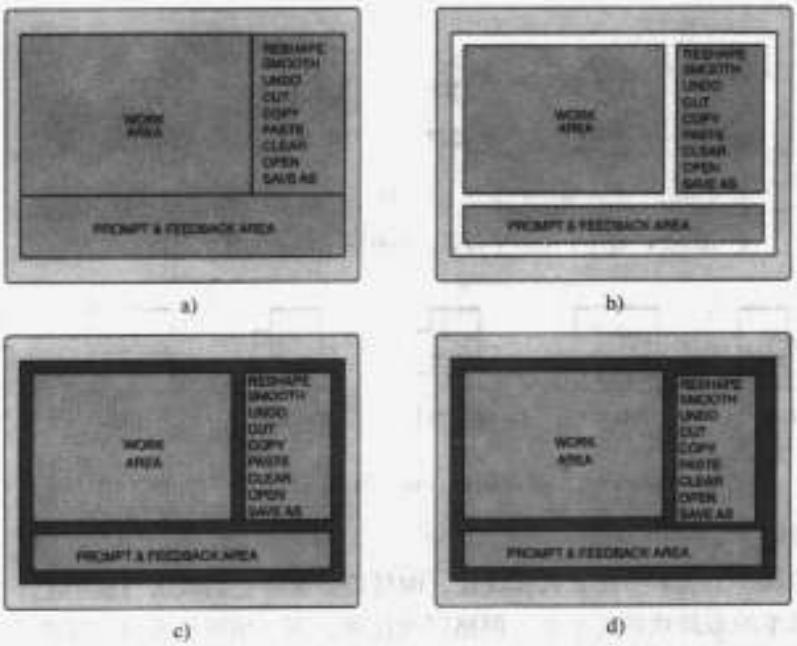


图9-27 四种屏幕设计。a)典型的初始设计，b)增加了边界区域，c)延长了边界从而进一步分离了这三个区域，d)没有将元素排列在网格上，其不良影响是明显的

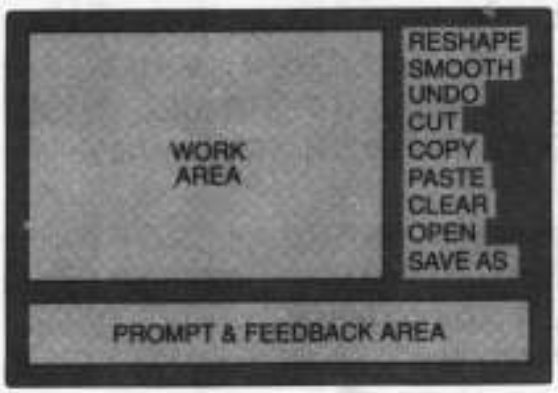


图9-28 将菜单区域周围的方框移去，从而产生了毫无意义的但吸引注意力的粗糙的右边界

426
↓
428

9.6 设计方法学

在第8章和第9章描述了很多思想，设计者如何集成它们并且以一种结构化的方法使用它们？虽然用户界面设计在一定程度上是一种艺术，而不是一门科学，但是在设计过程中我们至少可以建议一种组织方法。本节给出了该方法各要素的概貌。

设计界面的第一步是决定该界面要完成什么。虽然最初这种陈述可能比较烦琐，但是拙劣的需求定义至少在早期阶段会葬送许多用户界面设计项目。可通过研究如何解决当前考虑的问题，来部分地完成理解用户需求的任务。另外一种成功的方法就是让设计者学会如何用提问完成任务。设计的目标就是理解将来的用户当前会做什么，而且，更重要的是，他们为什么做它。

我们的意思并不是说界面应当刻意模仿当前的方法。弄清楚将来用户在工作时为什么这样做的原因，通常可以开发出新的和好的工具。但是，我们应该意识到有时模仿老的方法来避免大量的重新培训或者用已有的力量来避免信心问题会更好。一个典型的方法是首先模仿已有的方法，而且还要开发新方法，经过一定时间，用户就可以在新的扩展的能力方面得到训练。

还必须确定用户的特点，用户具有什么样的技能和知识？用户是对他们的工作很了解，还是一个计算机新手？他们是打字员？用户是很希望学会该系统，还是比较勉强？是经常使用还是偶尔使用，是全职的还是兼职的？当评估用户人数时，记住，系统设计者需要或喜欢的不一定就是设计对象所想要的，这一点很重要。不能凭系统设计者的想像来创作用户。

当列出需求时，下一步就是通过在9.1节所讨论的设计层次来完成自顶向下的设计：概念设计、功能设计、序列设计以及绑定设计。用户界面的自顶向下设计的基本原则最好是先做总体设计，再处理细节和较低层的问题。

首先开展概念设计。理想情况下，可做几种可替换的概念设计方案，在如何让用户在执行任务时做得更好的基础上，对它们做出评价，其中的任务是在需求定义中确定的。高频度的任务执行尤其要直截了当。简单性和通用性是设计的另外两条准则。

功能设计集中在命令以及它们能做的事情上。注意力要集中到每条命令所要求的信息上，集中到每条命令的功能上，集中到当激活命令时每条命令所能给用户的新的或修改过的信息上，以及集中到可能出现的错误情形上。图9-29所示为功能设计的焦点。注意，在随后的低层设计中会“设计出”表示错误的记号。功能设计的一个目标就是通过合理地定义单个命令减少可能发生的错误数目。

功能：Add_symbol_instance
参数：Symbol_identifier
Symbol_position
描述：生成符号的实例，并添加到所需位置的图表中。实例变成了当前选定对象（CSO）。即使想选中，前面的CSO也不再被选中。
反馈：实例在显示中可见，并且是高亮度的，因为它被选中了。（如果有一个CSO，则这个实例不再是高亮度的。）
错误：1. Symbol_identifier未知（利用菜单选择以选取符号）。
2. Symbol_position不在视口中（保持位置定位器的反馈，使位置在视口内）

图9-29 命令的典型功能说明。在确定了交互技术作为界面形式设计的一部分后，添加了错误说明

序列设计和绑定设计共同确定了界面的形式，它们是作为一个整体开发的，这样要比单独

设计好。设计包括：首先选取合理的一组对话框风格，然后将这些风格应用到特定的功能上。一系列屏幕图，有时也称为故事板（storyboards，有时也称情节串连图板），可用来定义这些设计的视觉方面和一些时间方面，像在9.3.1节和第8章中所讨论的状态图在处理用户动作序列方面也是有帮助的。

界面形式可以由风格指南来确定，该风格指南是用户界面形式中许多元素的成文规范。开发风格指南的最常见动机就是保证在应用程序中或它们之间“视感”（look and feel）的一致性。风格指南的一些元素可以利用交互技术库（第10章）来实现，其他元素必须由设计者和编程者实现。已经有了很多风格指南，它们中有Macintosh[APPL87]，OSF（开放软件基金会）的OSF/MOTIF[OPEN89]，NASA的Transportable Application Executive[BLES88b]，以及DEC的XUI[DIGI89]。

设计与用户界面原型的交替进行对整个设计过程很有帮助。用户界面原型是快速创建的最终界面的全部或部分，它通常有一些非常有限的功能，重点在于快速实现和快速修改。起初，一些设计问题看上去没法回答，但是一旦有了原型，回答就变得很清楚。因为原型给用户一个更具体的参考框架，所以它通常胜于使用设计文档，在这个参考框架内用户可以讨论他们的需求、喜好与厌恶。如同在10.6节所讨论的一些软件工具那样，HyperCard和Smalltalk被广泛用于快速建模。

429
~
430

一旦制定出概念设计就可以创建原型，而且可以同时开发功能设计和对话风格的各个元素。遵循Bauhaus定律（形式服从于功能）很重要，以防用户界面风格指导整个系统的功能。只要开发了界面的一些最基本的元素，就应该向未来的用户展示它们，从而提出一些界面的改进意见。一旦做了调整，并且原型变得更容易理解时，用户就应该使用该系统进行工作。循环往复已经被视为高质量用户界面软件开发的必要手段。原型开发和循环开发的更进一步讨论，可参见[ALAV84；HART89]。

习题

9.1 请确定在你所熟悉的交互图形应用程序中，一些命令是如何应用于直接操纵中的。

9.2 考察几个交互图形应用程序，刻画它们的对话风格。列出该界面设计的各种特点，说明它们遵循还是未遵循本章所讨论过的设计原则。对你所列出的每一点，确定设计层次。例如，颜色的一致性运用是在硬件绑定层次上。

9.3 许多字处理软件的概念模型是部分地建立在类似于打字机这样的基础上的。对于试图把这类软件用得更好的用户来说，这种类比可能会带来一些困难，列出几种产生这些困难的途径。

9.4 分析一种用户界面，确定其错误纠正的方法，如果有这种方法的话，是一些什么方法。把这些方法按照9.3.4节所讨论的四种类型进行分类。

9.5 表示一个完全非模式的用户界面的状态图的形式是什么？

9.6 设计并实现一个简单图形编辑器，它具有下列功能：线段的生成、删除和移动；线段端点的移动；线段类型的改变（虚线、点线、实线）；在线段尚未生成前设定线段的类型模式。这一系统要支持下列五种语法中的两种或两种以上，并且要包括一条两种语法之间的转换命令：对象方式，命令方式，带有Repeat_last_operation操作的对象方式，自由形式语法，以及带模式的和Do_it命令的自由形式语法。

9.7 做一项正式的或非正式的受控实验，用它实现练习9.6中的每一种语法。测试其容易学习的程度，以及使用速度。给用户五项预先设定的任务来完成，鉴别具有哪一种语法的任务在执行中的较快。

9.8 研究三种不同的交互图形应用程序。

a. 利用9.4节中的定义，确定每一种应用程序中的模式和语法的类型。单个应用可能有几种类型的模式和语法。如果是这样，是否有明显的差别表明哪一种类型何时使用？

b. 确定因子（参数）分解。具有其他的参数分解的可能性吗？你认为应用参数分解能改善用户界面吗？

9.9 考察图9-30所示的三种窗口组织形式。记录鼠标移动的数目和需要移动右下角时鼠标的点击数目。对于每一种情形，假设光标从窗口中央开始，并且必须返回到窗口中央位置。鼠标移动的时间按1.1秒计算，鼠标压下的时间按0.2秒计算，完成每一个窗口组织任务需要多长时间？这是否意味着一种组织形式比另一种好？ 431

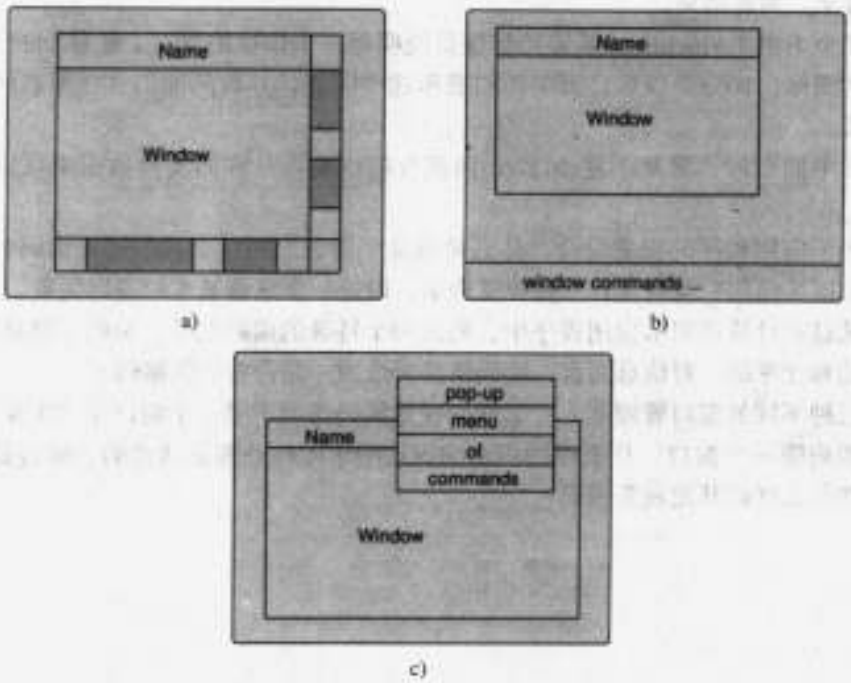


图9-30 激活窗口命令的三种方法。在a)（Macintosh类型）中，利用了窗口区域的装饰。要重新确定窗口大小时，用户选取并拖动窗口的右下角；在b)中，命令在屏幕底部的命令区域内。要重新确定窗口大小时，用户选取一个命令，然后拖动窗口的右下角；在c)中，在光标处，按下鼠标按钮，弹出菜单。要重新确定窗口大小时，用户选取弹出菜单中的命令，并拖动窗口的右下角

9.10 为你的交互图形应用程序实现单级Undo命令。在几种实现策略中，你使用哪一种，并验证你的选择。这些策略包括：(1)每一命令后面，做应用数据结构、状态变量等的完全拷贝；(2)保存应用程序数据结构中被改变的记录；(3)保存登录后的所有命令，由此响应Undo命令；(4)每10分钟保存一次应用数据结构，并保存上一次保存以后的所有命令；(5)对每一条用户命令，需要一条或多条命令，以便撤消用户命令。对于多级的Undo命令，你的选择有何不同？

9.11 检查一个交互图形应用。它包含多少条命令？列出“初学者工具箱”，以便初学者可以做一些简单的工作。相对于全部命令集，初学者工具箱应该多大？列出利用缺省或其他方法，使初学者工具箱的复杂性最小化。在你的选择中，系统具有良好的初学者工具箱吗？ 432

9.12 考察重新设计图9-17a的其他方法，以增强视觉元素的逻辑关系。

9.13 研究一个应用的对话框设计。可利用视觉结构的哪一种方法去增强该应用的逻辑结构。对设计能否做进一步的改进？

9.14 图9-6显示了表示Macintosh程序的12个图标：a)磁盘拷贝实用程序，b)资源移动工具，c)图标编辑器，d)菜单编辑器，e)警报/对话编辑器，f)编辑程序，g)引导配置程序，h)切换程序，i)文件检查程序，j)MacWrite（书写器），k)MacDraw（绘图器），l)MacPaint（画图板）。在表示相关的应用程序时，其中一些图标比其他图标更好。

a. 设计一个可更换的图标集，以表示12个应用程序。

b. 给10个不熟悉Macintosh系统的程序员展示这个图标集，并询问他们每一个图标所表示的含义。列表结果。

c. 向10个不熟悉Macintosh系统的程序员说明每一个图标的含义。给他们2分钟时间研究这些图标。10分钟以后，再向他们展示这些图表，并询问他们这些图表的含义。列表结果。

d. 对图中的图标，重复上述(b)和(c)两部分的试验。从你的实验数据中可以得到什么结论。

9.15 分析图形应用程序的视觉设计。什么是视觉字母表？用什么样的视觉编码？建立什么样的视觉层次结构？重新设计一些视觉提示，以进一步强调基本的逻辑关系。

9.16 在你见过的计算机图形应用程序中，列出10个特殊的编码例子。编码信息是否是主格的、次序的和比率的。对信息而言，编码值是否恰当？是否有错误编码？

9.17 检查三种不同的窗口管理器。什么样的视觉编码指明了哪一个窗口是“听众”，即键盘输入时指向哪一个窗口？什么样的视觉编码指明了哪种处理是活动的，哪些是阻塞的？使用一些什么样的其他视觉编码？

433

第10章 用户界面软件

第8、9章讨论了人机界面中的外部特征，除了已讨论过了基本图形包外，这里我们还讨论实现用户界面的软件构件，图10-1说明了用户界面的不同层次和各层次的作用，由图可知，应用程序涉及到所有的软件层，编程人员可以利用各软件层所提供的服务，但要注意的是，访问其中的一层会影响到另一层的行为。这里我们不讨论操作系统层，图形子程序包的基础也已讨论过了。10.1节将比较并评估一些与设备无关的图形子程序包的输入特性。从10.2节到10.4节讨论的窗口管理系统负责屏幕空间资源和交互设备的管理，从而多个应用程序或者同一个应用程序的几个视图可以同时显示。某些窗口管理系统有集成的图形子程序包，这些图形子程序包提供与设备无关的抽象，而其他应用程序则可以通过简单的图形调用，就可以访问底层的图形硬件或软件。

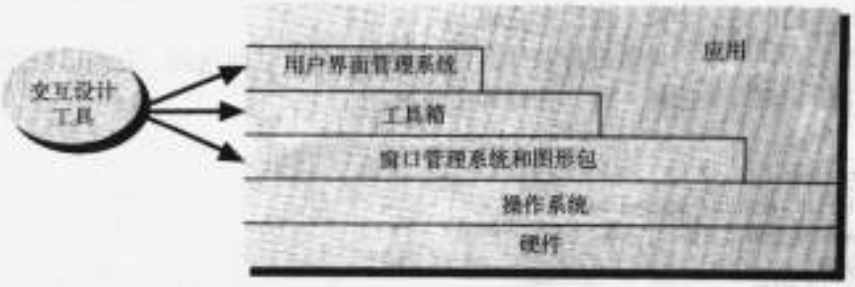


图10-1 用户界面软件的层次。应用程序可以访问操作系统、窗口管理系统、图形包、工具箱和用户界面管理系统（UIMS）。交互设计工具允许非编程人员设计窗口、菜单、对话框和对话序列

第8章中讨论过的交互技术对许多应用都是有用的，但仍需要仔细开发以提供良好的视感。10.5节介绍交互技术工具箱，以窗口管理系统为基础，给应用程序开发人员提供一个通用的技术集。最上一层是用户界面管理系统（UIMS）（将在10.6节中讨论）为相邻的设计层（9.1节）提供额外的通用用户界面的支持。UIMS加速了用户界面的实现，便于在9.6节讨论过的用户调试过程中进行快速改变。

10.1 基本的交互处理模型

本节将详细讨论在第2章（SRGP）和第7章（SPHIGS）讨论过的两个与设备无关的图形子程序包的交互处理能力。这两个图形子程序包的采样和事件驱动过程源于GKS[ANSI85b; ENDE86]和PHIGS[ANSI88]，它们共享一个共同的交互模型。窗口管理系统利用事件机制，这一点与本节讨论的GKS/PHIGS模型相同，但功能要强一些。

GKS和PHIGS有六类逻辑输入设备，并且每一类设备中，至少有一种设备与工作站有关（一种显示和与交互关连的设备），每一类逻辑输入设备可以按三种方式中的一种来操作：采样、请求和事件。每一种设备都有相应的度量，它是一种从设备返回的信息类型。这些设备及其度量如下所示：

设备	度量
定位器	在世界坐标当中的位置
拾取	SPHIGS的拾取路径，GKS的图段标识
选择	表示选择的整数
定值器	实数
串	字符串（在SRGP和SPHIGS中称为键盘设备）
笔划	世界坐标系中的位置序列

SRGP和SPHIGS采用与这些稍有不同的度量。

在*请求模式*中，应用程序从一种设备中请求输入，只有当用户利用这种设备进行一个操作后，图形包才返回设备的控制和度量。这一操作称为触发。对每一种逻辑设备类型，特定的触发操作与实现相关，但常用的是一种按钮触发。例如，鼠标按钮触发定位器和拾取设备，键盘的回车键触发字符串设备。

请求模式在同一时刻只能用于一种设备，并且是用来支持较旧的图形终端的有限功能，它通常通过RS-232接口与计算机相连。不能用键盘加速这样交互技术，原因是应用程序必须预先知道哪一种设备请求输入。另外，当设备的度量被修改（比如说，移动鼠标改变定位度量）时，应用程序一般不能提供动态反馈，原因是应用程序直到触发动作发生才恢复控制。这一困难可以通过改变触发动作的一个微小度量来克服。

435 ~ 436

在*采样模式*下，对单个设备进行采样，设备的度量可以立即返回。我们可以让用户在可用的几种设备中用轮询方法选择一种，选择算法如下：

```
terminate = FALSE;
while (!terminate) {
    SamplePick (&status, &segmentName);
    /* status=OK意即拾取成功；segmentName */
    /* 是拾取项的标识 */
    Process pick input
    SampleString (string);
    Process string input
    /* 作为处理串或拾取的一部分，terminate设置为 TRUE */
}
```

但是，用这种方法采样是危险的。若用户在开始输入时，选择了多个输入设备，这些输入对应用程序而言，是不可见的，因为在处理第一个输入时，其他采样被终止。另外，通常需要保持的用户事件队列也可能在采样中丢失。但是与请求模式不同，采样模式很适合从应用程序获得动态反馈，因为触发动作不需要给应用程序返回设备度量。

*事件模式*避免了采样模式和请求模式中的问题，它允许在同一时刻各个不同设备的同时输入。正如第2章2.3.6节所讨论的那样，应用程序首先允许所有设备可使用。一旦这些设备可用，则对这些设备中的任一触发动作按动作发生次序在输入队列中放置事件报告。如图10-2所示，应用程序检查这一队列，判别哪一个用户动作实际发生，并在这一事件从队列中移走时处理这一事件。

在事件模式下，下列代码块可重新实现前述的轮询的例子：

```
terminate = FALSE;
while (!terminate) {
    WaitEvent (timeout, &deviceClass, &deviceId);          /* 等待用户响应 */
    switch (deviceClass) {
        case pick: Process pick;
            break;
        case string: Process string;
```

```
            break;
        } /* 在处理串或者拾取过程中terminate设置为TRUE */
    }
```

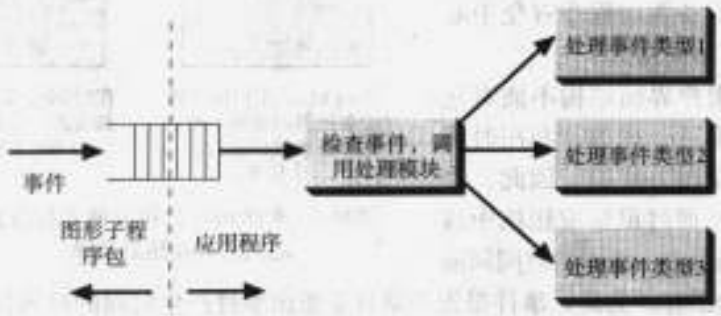


图10-2 应用程序将事件从队列中移去（删除），并且将控制传送到合适的程序，让它来处理这些事件

与请求模式输入不同，事件模式输入是异步的：一旦一种设备被启用，当用户利用可用输入设备并发地输入信息时，应用程序被执行。有时称之为键入在前（typeahead），或者，用鼠标操作时，称之为鼠标在前（mouseahead）。

事件队列机制的typeahead能力为加快与计算机交互提供了机会。假设一种按键动作（选择逻辑设备）用于拖动滚动条，比如说每一次拖动x英寸。若用户按键的速度比滚动条移动的速度快，事件存放到事件队中。应用程序可以从队列中找到多次连续按键事件；如果有n次这样的事件，那么就执行一次滚动nx英寸这样的操作，这样要比移动n次而每次移动滚动条x英寸快得多。

但必须小心管理事件队列。如果队列中两事件中第一个可以导致程序启用不同的逻辑输入设备，则会调用WaitEvent，现在程序可能不希望发生第二个事件，以出现不希望的结果。提供FlushDeviceEvents调用就可以解决这一问题；应用程序可以清空队列，以保证队列中没有不希望的任何事件。但是清空队列可能令用户疑惑：为什么第二件事件没有处理。

与事件队列有关的另一个问题是，在一件事件发生及与该事件所需有关信息获取之间的时间间隔内，可能会有时间延迟。假设我们想要在鼠标所在位置显示一个菱形。若鼠标上的按钮被设成逻辑选择设备，鼠标的坐标(x, y)被设成逻辑定位器（这一点与SRGP的定位相反，它的度量包括按钮的状态），则可以用下列方法来实现。

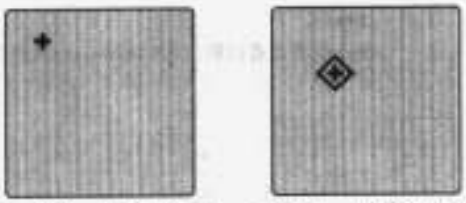
```
WaitEvent (timeout, &deviceClass, &deviceId);
if (deviceClass == CHOICE && deviceId == 1) {
    SampleLocator (MOUSE, x, y);                    /* 确定菱形的位置 */
    DrawDiamond (x, y);                             /* 在坐标(x, y)画菱形 */
}
```

437
↓
438

问题是，在时刻t_1（WaitEvent过程返回的时刻）和时刻t_2（SampleLocator过程获得(x, y)坐标的时刻）之间，可能浪费了好几秒钟。这段时间内，定位器可能被移动了一段距离，从而导致了如图10-3所示的不希望的结果产生。对于分时计算机，当另一程序正占用处理器时，实际上的延迟就容易发生，对于支持虚拟内存的计算机，当页面错误中断发生时，这样的时间延迟也会发生。因此，虽然我们希望应用程序在时刻t_1到t_2之间不至于中断，但不能保证这样的事不发生。

在GKS和PHIGS中，可以通过激活具有相同触发动作的多个逻辑设备来降低甚至消除这样的时间延迟。如果我们定义触发器为三个按钮中的任一个，并且将该触发器与三钮选择设备和定位器设备联系起来，那么两个事件可同时放入事件队列中（没有指定的次序）。在图形包中

与设备相关的驱动程序，对时间延迟处理方面，比应用程序执行前面的代码段的方法要来得快，从而中断发生的可能性降低了。如果操作系统可以保证设备驱动程序有免中断的权限，则系统不会有时间延迟。



在时刻t_1，用户用选择设备选择画菱形。然后用户移动指针进行下一个任务。

在时刻t_2，定位器取样，画菱形。定位器在时刻t_1和t_2之间移动。

图10-3 事件和与之相关的光标位置的采样之间的时间延迟效果

不幸的是，一些用户界面结构不能对这些逻辑设备概念提供支持。例如，利用时间间隔来区分两个命令是很方便的。因此，当光标移到一图标上，并通过鼠标按钮选中这一图标，然后就可以通过在一个Δt的时间间隔内再次点击来打开这个图标。为此，事件报告必须具备给出事件产生时间的时间戳。在GKS和PHIGS中没有这样的概念，尽管它们提供时间戳较为容易。

10.2 窗口管理系统

窗口管理系统提供了现代人机界面的许多重要特征：应用程序在不同的区域上显示结果，
439 改变这些应用程序所运行的屏幕区大小的能力，弹出式和下拉式菜单，以及对话框。

窗口管理系统首先而且最重要的是资源管理，这与操作系统一样，只是两者所管理的资源类型不同，它把屏幕区域的资源分配给需要用屏幕的不同的程序，并管理这些屏幕区域，使得程序间不冲突。这一方面将在10.3节进一步讨论。窗口系统也把交互设备资源分配给需要用户输入的程序，然后将输入信息流从相应的输入设备引导到相应程序的软件队列中。输入处理将在10.4节讨论。

这三节的目标是提供一个主要的窗口管理概念的概述。这一主题最全面的阐述可见[STEI89]，历史发展概况在[TEIT86]中给出，相关论文的著作参见[HOPG86b]。

窗口管理系统有两个重要部分，首先是窗口管理器，它可以生成用户交互请求所需的窗口，并可进行定义窗口大小、移动、打开、关闭窗口等操作，其次是潜在的功能组件，即窗口系统，窗口系统实际完成上述操作。

窗口管理器在窗口系统的最上层：窗口管理器使用窗口系统提供的服务。窗口管理器的作用与下层的窗口系统中的命令行解释器相同，是操作系统中的核心。同时处于窗口系统上层的是高级图形包和应用程序。窗口系统生成的程序有时也叫客户程序，反过来，它又使用窗口系统的功能，它有时也称为服务程序。在一些客户-服务器窗口管理系统中，如在X Window系统[SCHE88a]和NeWS[SUN87]，窗口管理器自身也作为窗口系统的另一个客户程序。而在另一些系统中，窗口管理器和窗口系统的关系要比客户与服务器的关系更接近些。

一些窗口系统（包括X Window系统和NeWS）被设计成自由策略(policy free)，即有多个窗口管理器（每个窗口管理器有不同的视感）并且构建在窗口系统之上。窗口管理器（而不是窗口系统）决定窗口是什么样的，用户与窗口是怎样交互的。自由策略支持图9-30所示的所有窗口风格，同时也支持其他风格。正如可以在图形包上创建许多不同的应用程序一样，在自由策略的窗口系统上也可以创建许多不同的窗口管理器：窗口管理器和图形应用程序都控制系统的外观和行为。可能的话，窗口系统应该设计成可以执行范围更宽的窗口管理器策略（见习题10.10）。当然在一些特定的环境中，窗口管理器和应用程序需要有共同的用户界面视感。

如果窗口系统的程序界面是清晰地定义并通过进程间通信功能来实现的，那么，窗口系统
440 的客户就可以常驻在与窗口系统不同的客户机上了，使用窗口系统的计算机之间通过高速网络

相连。如果窗口管理器自身是窗口系统的另一个客户，则它也可以常驻在另一台计算机上。用这种方法使用进程间通信机制可以允许计算密集的应用程序常驻在功能强的计算机上，而用户和应用程序之间仍通过工作站来交互。从这点来看，客户-服务器模型恰恰是虚终端协议的典型实例，这种协议一般都有这一优点。

窗口管理系统不必建立在客户-服务器模型之上。例如，Macintosh并没有把窗口管理器和窗口系统很清楚地分隔开。这种分隔对于单活动进程和Macintosh的单处理器设计目标并不是必须的，而且会导致额外的运行时间开销。

在窗口管理器和窗口系统之间能够提供进程间通信的窗口系统（X-Window系统、NeWS和Andrew[MORR86]）中，界面需要设计成最小的通信延迟。解决这一问题有许多种策略。首先，在客户-服务器之间尽可能使用异步通信（而不是同步通信）。这样，当客户向服务器发送消息后，不必在处理和发送另一条消息之前等待回应。例如，当客户程序调用DrawLine命令时，消息发送到服务器，控制命令立刻返回到客户端。

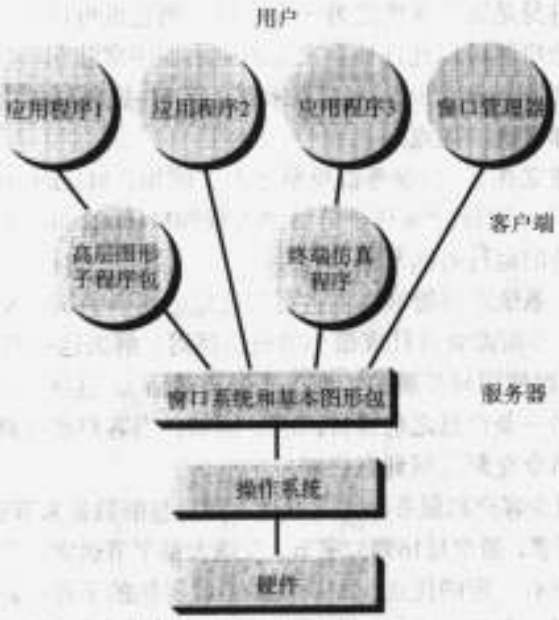
有时我们可以通过减少客户和服务器之间发送不同消息的数量来节省时间。大部分网络通信协议都有最小包大小的要求，通常是16到32字节，发送大量字节通常比发送最小包所需时间要少。因此在批量消息发送方面有一定的优点，如同某些系统提供的子程序调用BeginBatchOfUpdates和EndBatchOfUpdates那样；在BeginBatch和EndBatch之间的所有调用都在一条消息内传送。采用单一消息替换多个消息的设计有一定的优点，如单一消息可设置多个图形属性。

第三种减少通信的策略是把更多的功能和通用规则移到服务器中。大多数客户送到窗口系统的命令都是比较基本的：画线，创建窗口，复制像素图。例如，在X Window系统中，需要很多命令来创建菜单或对话框。一种更可靠、更强大的策略是传送命令到服务器，这种命令是用可高效解释的语言编写的程序。这样，这些命令很通用并且能实现这种语言可能表达的任何功能。当然，通用性的代价是要为解释性执行程序花费的时间，以及需要为解释器付出的额外空间开销，这对当前计算机而言价格并不贵。得到的好处是通信流量的急剧下降。将更通用规则和功能移到工作站上的策略并不新颖；二十年前首次创建分布式图形系统时，就恰好讨论了这两个问题[FOLE71；FOLE76；VAND74]。

第三种策略已用于NeWS窗口系统，该系统接受用扩展的PostScript语言编写的程序作为命令[ADOB85a；ADOB85b]。PostScript将传统的编程语言结构（变量、数据结构、表达式、赋值、控制流、I/O）和嵌入式图形功能结合在一起，这些图形功能可用于绘制输出的图元、对任意区域的剪切及变换图元。NeWS增加了对进程、输入和窗口的扩展，这一语言将在19.9节深入讨论。 441

为了在NeWS中定义对话框，当一个程序开始执行时，就会将定义该对话框的PostScript程序发送给服务器；对话框的每一次出现，就会发送一个消息来调用该程序。这种策略避免了每次重发该对话框的定义。类似地，还能向服务器发送对执行限时操作的程序，例如直线（第8章）或曲线（第11章）的橡皮筋画法，这样就避免了为更新橡皮筋线过程而在服务器和客户端之间每一个消息来回所涉及的时间延迟。

图形包通常是和窗口系统结合在一起的，典型的例子就是功能上类似于第2章的SRGP的2D不分段图形包。如果下层的硬件具有3D或分段功能，那么窗口系统层就可以调用硬件的图形功能。图10-4显示了窗口系统及其图形包与其他的系统构件之间的典型关系；图10-5显示了窗口、客户以及输入事件之间的关系。用户产生的事件，包括窗口事件（调整窗口大小、重新定位、推拉、弹出、滚动等等）由窗口系统发送到窗口管理器；其他事件被发送到适当的应用程序中。



442

图10-4 窗口系统到操作系统和应用程序之间的关系

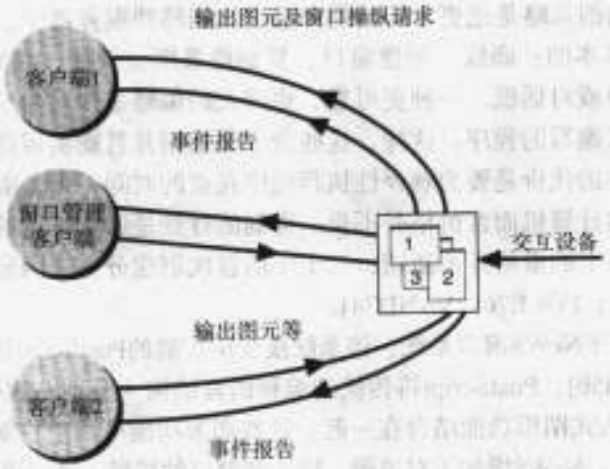


图10-5 窗口、客户以及事件之间关系的另一种视图。每一个客户输出到窗口；输入事件被发送到适当的客户端事件队列中

10.3 窗口系统中的输出处理

窗口系统分配给客户程序的输出资源是屏幕空间，屏幕空间的管理使得客户之间对屏幕的使用彼此不会相互影响。不同的窗口系统，分配的策略有很大不同，但大致可分为三类。主要的区别在于如何显示显露的窗口（当将窗口放大、展示或滚动时）的各个部分。这些策略逐渐将更多的任务放在窗口系统本身这个结果上，这样客户可以逐渐不去注意窗口的存在。系统也需要管理一个查找表来避免客户之间的冲突。

小型窗口系统不负责绘制窗口新显露出来的部分；相反，它发送一个“显示窗口”事件给负责该窗口的客户。这样的窗口系统不保存窗口外的遮挡部分。当客户发送输出单元到一个窗

口时，输出图元在窗口系统的窗口（这里窗口对应于图形包的视口）中被剪裁。如果在部分可见窗口中画一条直线，只画直线的可见部分。

当负责窗口的客户接收到“显示窗口”事件时，客户处理事件的方式是对窗口系统所显示的整个部分建立剪裁窗口，然后处理所有的输出图元，或者使用较小的裁剪窗口显示窗口新显示出来的部分，如图10-6所示。另外，客户可以保存被遮挡窗口部分的像素图，以避免再次显示窗口时重新生成该窗口的内容。客户只要利用从保存过的像素图到屏幕像素图的命令PixBlt来显示保存过的像素图的部分或全部内容。在某些情况下，窗口系统重画窗口背景及其边框来提供快速响应的假象，即使实际上的重新显示可能产生可感觉到的延迟。 443

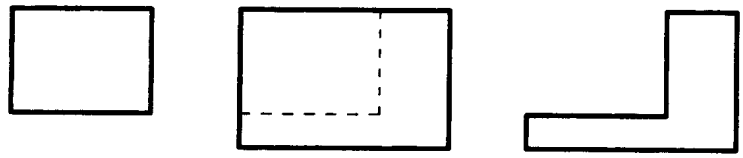


图10-6 当把窗口a)放大到b)时，更新被放大的窗口有以下几步：使用在原始窗口a)中的像素图，剪裁针对剪裁区域c)的世界坐标输出单元，以及画出在剪裁区域内部的输出单元。注意剪裁区域是非凸多边形，这样剪裁一条直线可能导致多于一条直线段的显示

小型窗口系统方法在支持不同应用程序数据模型的系统中是有意义的，数据模型范围可以从像素图、结构化层次到更复杂的关系模型。窗口系统本身没有一个很好地适用于所有应用程序的模型。相反，应用程序要完全负责提供有效的存储和重绘功能。

Macintosh窗口系统就是这种按最低要求设计的一个例子[APPL85]。嵌在Quickdraw图形包中的算法允许对任意区域的裁剪，如图10-6中所示的区域。NeWS和X Window系统也支持任意形状的窗口。19.7节将讨论一些需要支持任意窗口的裁剪算法。

更多的具有较大内存的窗口系统将窗口中被遮挡的部分保存起来，以便客户就没必要显示窗口最近显露的部分。其他窗口系统让客户选择是否保存窗口被遮挡部分。在任何情况下，都有这样的问题：需要保存多大的被遮挡的窗口。一般情况下，尽可能保存最大的窗口，通常也就是整个屏幕的大小。一些窗口系统保存比显示屏更大的像素图，虽然当像素图变得很大或位数更多时，这种方法的代价比较高。但是，降低内存的价格在节约成本上有很大的作用。如果窗口滚动到已保存过的像素图以外的部分或者改变了视图的大小，那么客户必须进行重新绘制。

另外一种略有不同的策略是，对每一个窗口，窗口管理器保存一个包含整个窗口的画面外的像素图。只要当部分窗口变得可见时，画面外的像素图的相应子区域就会被PixBlt拷贝到屏幕区域内。这种方法对窗口更新来说比较慢，因为部分被遮挡的窗口由客户程序写入。因此，在客户程序写入一个窗口（就是画面外的像素图）后，窗口系统把窗口显示部分拷贝到屏幕上。另一种方法是，窗口系统可以直接把新的输出图元扫描转换成两个部分，即画面外的像素图和窗口的可见部分，方法是把每一个输出图元裁剪成两个不同的区域：一个是像素图可见部分，另一个是整个画面外的像素图。仅仅更新窗口的可见屏幕区域就可迅速更新完整的未遮挡窗口；只有当窗口要被遮挡的时候，才将窗口拷贝到它的画面外的像素图中。为了避免这个问题，一些窗口系统要求正在被写入的窗口应当完全不被遮挡。不幸的事，当一个活动进程需要写入部分遮挡的窗口时，这种解决方法也阻止了多个进程。几种有特殊用途的硬件系统不需要拷贝画面外的像素图到屏幕像素图。在这些系统中，硬件知道每一画面外的像素图存储在什么地方 444

并且其中的哪一部分在屏幕上是可见的。在每一个刷新周期中，视频控制器从画面外的像素图中拾取合适的像素。显示窗口中的更多内容并不是靠拷贝像素，而是通过给硬件提供新的窗口显示信息来完成。这些硬件解决方案的进一步讨论，可参见第18章。

第二种用来实现这种窗口系统的方法是由Pike[PIKE83]开发的，我们将在19.8节进一步讨论。这种方法避免了任何信息的两次存储。每一个窗口被划分为矩形的像素图。不可见的像素图被存储在画面以外，而可见的像素图被存储在屏幕的刷新存储器内。

但另一种重要的设计策略是让窗口系统为每一个窗口保存一个显示列表，正如在Lantz和Nowicki[LANT84]设计的VGTS系统中那样。实质上，窗口系统保存一个基于显示列表的图形包（如SPHIGS）作为系统的一部分。无论何时窗口需要被重绘时，列表都被遍历和裁剪。理想的方法是用快速转换扫描的硬件实现，以便重绘时间不会变得非常长。虽然VGTS的确包括像素图图元，但这种方法并不适用于面向像素图的应用，比如绘图程序。

一个经常碰到的问题是一个改变窗口大小的操作对窗口显示的信息量的影响：当窗口管理器的窗口被改变时窗口的世界坐标窗口会发生什么改变？客户程序将出现两种可能。第一种是，当用户改变窗口的大小时，窗口的世界坐标会做出相应的改变。纯效果是根据窗口变大或变小，用户看到更大或更小的范围，如图10-7c所示。第二种可能性是，当用户改变窗口的大小时，世界坐标系的窗口大小保持不变。这样，当窗口变大，将看到同样大小的范围，但比例被变大了。有一种方法，即使全局窗口和窗口系统窗口外观比例不一样，也是用统一的缩放比例；这样会使窗口系统中的窗口的一部分不起作用，如图10-7d所示。另一方面，还可以用不一致的缩放比例，调节全局窗口的内容以适应窗口系统窗口，如图10-7e所示。

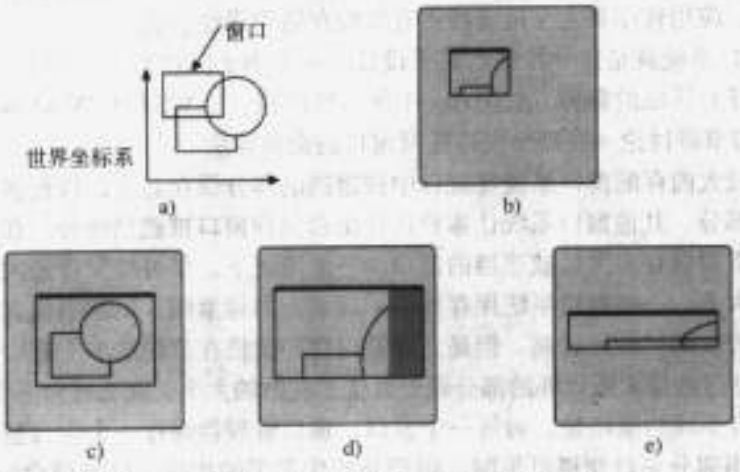
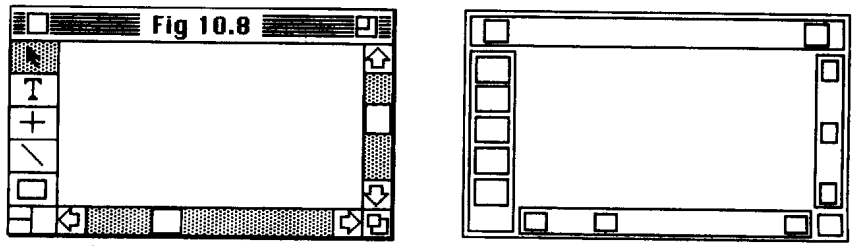


图10-7 世界坐标系窗口和窗口管理器窗口之间的关系：a)世界坐标系场景；b)通过窗口的场景视图；c)当放大窗口管理器窗口时，更多的场景可看到；世界坐标系窗口被放大。在d)中，放大窗口管理器窗口创建世界坐标系窗口内容的放大的视图。放大使用统一的缩放比例，所以窗口管理器窗口的一部分（灰色部分）没有用到。在e)中，放大窗口管理器窗口也创建了世界坐标系窗口内容的放大的视图，但放大没有使用统一的缩放比例，所以整个窗口管理器窗口被填充

一些窗口系统应用了层次窗口，也就是窗口包含子窗口，如图10-8所示。子窗口被包含在父窗口内。层次窗口可以用于实现第8章所述的各种类型的对话框和表格。整个对话框被定义为一个窗口，它的每一个域、单选按钮和复选框被定义为一个独立的子窗口，每一个子窗口都可以接收鼠标的按钮按下事件。当用户选择任何一个子窗口时，事件报告包含子窗口的名字。一个典型的限制是子窗口被包含在它们的父窗口内，因此当对话框被移动到应用程序窗口以外，

它不能作为应用窗口的子窗口被实现。



a) b)

图10-8 a)绘图程序的窗口；b)将窗口分为层次窗口。在b)里，被包含的窗口是包含窗口的子窗口。(Claris公司版权所有，1988。)

层次窗口系统的设计包括很多技巧，比如确定父窗口大小变化对子窗口的影响。另外，当一个客户进程产生一个创建窗口的子进程时，除非子进程可以有它自己的事件队列来接收来自它的窗口的输入，否则该子进程的窗口被看成是产生进程窗口的子窗口。关于层次窗口的进一步讨论可以参见[ROSE83；SCHE86]。

445
~
446

一个典型的窗口系统必须处理的子程序调用应包括下面的函数：

CreateWindow (*name*)	创建一个新窗口并使其成为当前窗口
SetPosition (*xmin,ymin*)	设定当前窗口的位置
SetSize (*height,width*)	设定当前窗口的大小
SelectWindow (*name*)	使窗口成为当前窗口
ShowWindow	使当前窗口可见，置于所有其他窗口之上
HideWindow	使当前窗口不可见；显示当前窗口遮挡的其他窗口
SetTitle (*char_string*)	将当前窗口的名字设为*char_string*
GetPosition (*xmin,ymin*)	得到当前窗口的位置
GetSize (*height,width*)	得到当前窗口的大小
BringToTop	将当前窗口置于所有其他窗口之上
SendToBottom	将当前窗口置于所有其他窗口的底层
DeleteWindow	删除当前窗口

另一种由窗口系统分配的输出资源是查找表项。假如一个窗口系统运行在一个每像素8个位的硬件系统上，有两个窗口系统的客户，每个都需要有一个查找表。对这两个客户，每一个都可以用表项的一半(128)，但每个客户的查找表项的数目取决于客户数。可以规定一个固定的客户上限数用来决定每个客户查找表项数目，但如果实际上的客户较少，许多查找表项就会被浪费。一个客户一次能独占查找表——经常是其窗口包含光标的客户。这种解决方案是可行的，但是有害的，因为当查找表首先分给一个客户，然后分给另一个，以此类推，整个屏幕的显示会发生显著的改变。

另一个解决方案是不分配查找表项，而是用颜色。假如客户申请纯蓝色，并且一些表项已经包含了这种颜色，客户被赋予相同的索引来用这些表项（但并不允许改变表项的内容）。如果表项中没有这种颜色并且还有空表项，就分配客户一个空表项。否则，分配一个与申请颜色最相近的颜色的表项。缺点是所申请的颜色与实际的颜色可能差别很大；另一个缺点是不能对查找表进行改变。遗憾的是，还没有一个完全满意的解决方案。

10.4 窗口系统中的输入处理

窗口系统为系统客户分配和管理的输入资源是输入设备和由输入设备产生的事件。窗口系

统必须知道不同的事件对应哪个客户。这个过程有时被称为信号分离，因为指向不同的客户的事件从一个来源按顺序到达，然后被分到不同的客户中。

事件类型在10.1节中已讨论，另外还有一些窗口系统的特殊事件。许多窗口系统产生进入窗口和离开窗口事件，它允许用户界面突出显示包含光标的窗口，而不需要指针设备不断采样的开销。每当窗口需要重绘时，如果窗口系统不保留每一个窗口所显示的内容记录，那么窗口
447 系统就产生窗口受损事件。受损事件出现在放大窗口、窗口显露或者滚动时。窗口系统在直接响应用户动作时产生进入窗口和离开窗口事件，但是；窗口受损是客户要求改变窗口时产生的辅助事件。所有这些事件都被发送到客户事件队列里。一些用户动作（比如关闭一个窗口）会导致给很多客户发送窗口受损事件。事件报告里的消息与10.1节所讨论的类似，但是这里增加了事件类型和特定窗口信息。

如果窗口有层次结构，则可以像父窗口一样操纵子窗口，并且子窗口有相应的事件。与事件相关联的窗口名是层次中包括光标、事件对于它们是激活的、最底层的窗口名字。这意味着不同的事件可以和不同的窗口相联系。每一个事件队列里的事件都记录了与它相关联的窗口的名字。也可以将一个子窗口中的事件不只报告给这个子窗口，而且还可以报告给包括这个子窗口的所有窗口。客户完成这种报告的方法是使同一事件对层次路径中的所有窗口（从最外层包含窗口到最底层子窗口）可用。因此，每一个事件都带有一个不同的窗口名的多个事件报告，将放置到客户的事件队列中。

如果窗口有层次结构，一个弹出式菜单可以被定义为一个被分成若干子窗口的主窗口，子窗口的数目同菜单项一样多。当光标从菜单项i的窗口移动到菜单项$i+1$的窗口时，一个带有菜单项i窗口名的离开窗口事件被放到事件队列里，接着放一个带有菜单项$i+1$窗口名的进入窗口事件。客户程序处理第一个事件的方法是撤销菜单项i的高亮反馈，而处理第二个事件的方法则是在子菜单项$i+1$上创建一个高亮反馈。如果光标进入没有被子窗口覆盖的弹出菜单区，比如顶部的标题区，则进入窗口事件包含弹出菜单的窗口名。层次窗口就是这样使用的，但是由于管理层次窗口和及其大量事件需要处理时间，用户响应时间可能延长。由于反馈可以在服务器上处理，NeWS型的窗口系统不产生如此多需要大量处理的事件。

层次窗口可用于选择显示对象，其使用方式与SPHIGS结构非常相似。当然，子窗口不像结构一样通用，但是它们能够方便地获取矩形的、层次嵌套的区域（NeWS和X支持非矩形窗口）。层次窗口的其他用途包括：当光标从屏幕的一个部分移动到另一个部分时，产生光标形状的改变；有时在用来操纵图形对象时，允许对操作的拾取检测（参见图8-42）。

两个基本的方法广泛运用于窗口系统中，用于向客户发送事件：基于不动产（real-estate-based）的发送和监听者发送。实际上，很多窗口系统都提供这两种策略，并允许窗口管理器指定使用哪一种策略。有些还允许窗口管理器提供其他策略来代替这两种基本方法。在基于不动产的发送中，当一个事件发生时，查找光标在哪个窗口里面；所有的事件都直接发送到创建窗口的客户，窗口名被包括在事件报告中。

对于基于不动产的发送，窗口系统必须维持一个数据结构来存储每一个窗口的边界，如图10-9所示。当一个事件发生时，在数据结构中搜索包含光标位置的可见窗口。如果数据结

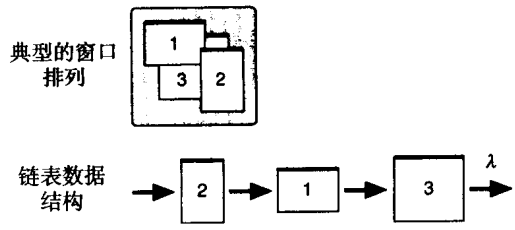


图10-9 窗口管理器用来决定光标在哪个窗口的数据结构，搜索列表中第一个包括光标位置的窗口

构是有序的，最近完全显示的窗口放在头部，则搜索在遇到第一个包含光标位置的窗口时终止。对于层次窗口，则需要一个更复杂的数据结构和搜索算法（参见习题10.2）。

当一个客户让窗口系统发送某一类型的所有事件到另一个窗口（接收客户可以是、但不必是发出请求的客户）时，进行监听事件发送，也叫作click-to-type发送。例如，窗口管理器可以有一个允许用户将所有的键盘输入发送到拥有一个特定窗口的客户的命令。窗口管理器通过指定窗口系统发送键盘事件到一个适当的客户程序来实现这个命令。键盘事件是最普通的显式发送事件，而一个鼠标键按下事件也能被重定向。

事件分发能导致令用户意外的结果。例如，假定用户意外地双击了一个窗口的关闭（也称为“离开”）按钮，尽管只需要一个单击按键事件来关闭窗口。窗口系统发送第一个点击事件到窗口管理器，这个事件将关闭窗口。接着发送的第二个点击事件则被发送到任何在关闭按钮下层的窗口，也许会转而选择了一个菜单命令！

在网络中的消息传输延迟也会对用户产生破坏。考虑在窗口*A*中运行的一个制图程序。为了画一条橡皮筋线，用户先在窗口*A*中产生一个鼠标键按下事件，接着将光标移动到线段要结束的任何地方。如果这条线是在通过窗口不可见的绘图区域里结束，这个点可能在窗口*A*以外。预期到光标将移动到窗口以外，制图程序向窗口系统发出一个请求，要求把所有的鼠标键放开事件都发送给自己，无论事件本身发生在哪里。现在，如果用户确实将光标移动到了窗口*A*以外，我们会有一个*竞争条件*。鼠标键放开事件是发生在窗口系统收到所有的鼠标键放开事件都发送给制图程序的请求之前还是之后？如果鼠标键放开事件是在请求接收前发生，事件将被发送到光标所在的任何窗口，导致一个不可预料的结果。如果鼠标事件在请求收到以后发生，则一切正常。避免这种竞争条件的惟一可靠的方法是要求客户在服务器向客户发送事件$i+1$前告诉服务器事件i已经被处理。

448 ~ 449

一旦事件到达一个客户中，它们就进入一个按图10-2排序的事件队列。客户发送或分发事件给不同的事件处理程序。一个典型的分发器伪代码是：

```
while (!quit) {
    WaitEvent (timeout, deviceClass, deviceId);
    switch (deviceClass) {
        case CLASS1: switch (deviceId) {
                case DEVICE1: ProcedureA (); break;
                case DEVICE2: ProcedureB (); break;
            }
        case CLASS2: switch (deviceId) {
etc.
```

随着事件的发生及程序移到不同的状态，响应特定事件时调用的过程可能会改变，进而使程序逻辑更为复杂。

分发器模型(dispatcher model)（也叫作*通告程序模型*）用一个响应用户动作的过程来增强输入处理系统，如图10-10所示。应用程序在通告程序上注册一个过程，并告诉通告程序在何种条件下调用该过程。被通告程序调用的过程有时叫作*回调过程*，因为它们被通告程序回调。对于层次窗口，不同的回调过程能够同定位事件相关联，这个定位事件发生在对话框的不同部分或者菜单的不同部分。这些回调过程可能会在事件报告给应用程序之前对事件进行修改，在这种情况下它们被叫作*过滤过程*。

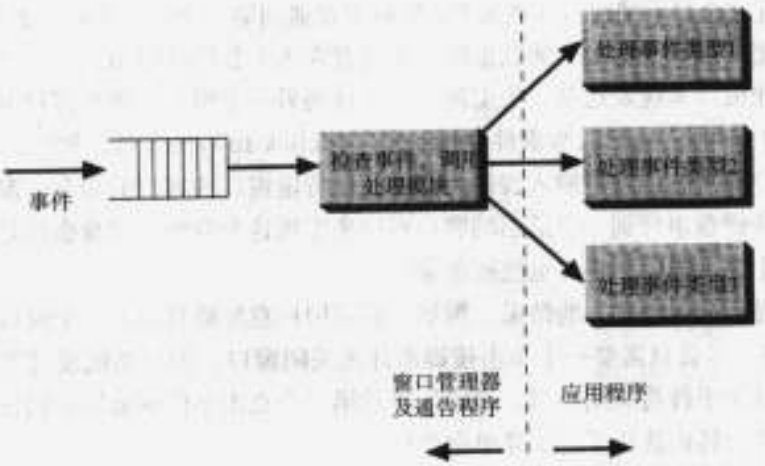


图10-10 窗口管理器的通告程序检查事件队列，并调用预先注册的过程来处理一个特定事件

一个典型的窗口系统必须处理的输入子程序调用包括：

EnableEvents (*eventList*)	启用列出的事件集
WaitEvent (*timeout,eventType, windowName,eventRecord*)	从事件队列里取出下一个事件
SetInputFocus (*window, eventList*)	指定具有*eventList*中类型的所有输入事件都发送给*window*
CursorShape (*pixmap, x, y*)	*pixmap*定义了光标的形状；*x*，*y*给出了光标像素图中的位置，它用来报告光标位置

放入事件队列中的典型事件类型如下：

KeyPress	键盘上的键按下
KeyRelease	键盘上的键松开
ButtonPress	定位器（比如鼠标）的按钮按下
ButtonRelease	定位器的按钮松开
Motion	光标移动
LeaveNotify	光标离开窗口
EnterNotify	光标进入窗口
WindowExpose	窗口被部分或完全显示
ResizeRequest	请求窗口改变大小
Timer	原先指定的时间或事件发生的时间增量

这些事件类型都有一个时间戳（为什么需要这样，参见10.1节）、事件发生时光标所在窗口的名字和其他事件特定的信息，比如，ResizeRequest的新窗口大小。

以上窗口管理系统的简略综述还没有讨论一些重要的问题，比如，确保窗口系统是足够通用的，以便能够提供任何类型的窗口管理器策略。窗口系统是一个对客户独立的进程，还是一个与客户链接的子程序库，或者是操作系统的一部分，这也是很重要的。这些与其他问题一起在[LANT87；SCHE86；SCHE88a；STEI89]中有全面的论述。

10.5 交互技术工具箱

用户-计算机界面的视感很大程度上由提供给它的交互技术集所决定。回想一下，交互技术实现了用户-计算机界面设计的硬件绑定部分。设计和实现一组好的交互技术很花费时间。作为交互技术的子程序库，交互技术工具箱是一种供应用程序开发人员使用的技术集合。这种

方法有助于在应用程序中保证一致的视感，显然是一种有效的软件工程准则。

交互技术工具箱不仅可以被应用程序使用，而且也可以被窗口管理器使用，窗口管理器毕竟也是另一种客户端程序，在所有的应用程序中，使用同一个工具箱是一种重要而常用的方法，它提供了与多种应用程序和窗口环境本身统一的视感。例如，用来选取窗口操作的菜单风格在应用程序中应保持一致。

如图10-1所示，该工具箱可以在窗口管理系统的顶层实现。在没有窗口系统的情况下，工具箱可以直接在图形子程序包的顶层实现；然而，由于工具箱的元素包括菜单、对话框、滚动条以及所有能在窗口中方便实现的类似的部件，所以通常使用窗口系统的底层。广泛使用的工具箱包括Andrew窗口管理系统的工具箱[PALA88]、Macintosh工具箱[APPL85]、X窗口系统使用的OSF/Motif[OPEN89a]和InterViews[LINT89]、在X和NeWS上实现OPEN LOOK的几个工具箱[SUN89]、Presentation Manager[MICR89]及SunView窗口管理系统的工具箱[SUN86]。彩图I-26和彩图I-27所示为OSF/Motif界面，彩图I-28和彩图I-29所示为OPEN LOOK界面。

450 ↕ 451

在X Window系统中称交互技术为widget（窗口小部件），在这里我们就采用这种叫法。一组典型的widget包括对话框、文件选取框、警告框、帮助框、列表框、消息框、单选按钮和单选按钮组、选取按钮组、切换按钮、切换按钮组、固定菜单、弹出式菜单、文本输入框、滚动条和应用程序窗口。这些widget通常都是作为窗口来实现的，在X窗口系统中，也可以使用子窗口。例如，单选按钮组就是一个窗口，其中的每一个子窗口表示一个单选按钮。复杂的对话框可能有许多子窗口，应用程序窗口可能有许多子窗口，用来表示滚动条、大小可变的按钮等等，如图10-8中所示。

交互技术工具箱有在10.4节所讨论的典型的通告程序，当事件出现在子窗口时以调用回调过程。在某些情况下，过程就是工具箱的一部分，例如，加亮当前菜单项，选定和取消选定单选按钮，以及滚动列表或文件选择框等过程。应用程序也提供这些过程；例如，在菜单中选择命令，在文本输入区域内检查每一个字符的有效性，或者简单地记录选择按钮这样的事实等过程。图10-11表示了对话框的一部分和某些过程，这些过程可能与对话框有关。

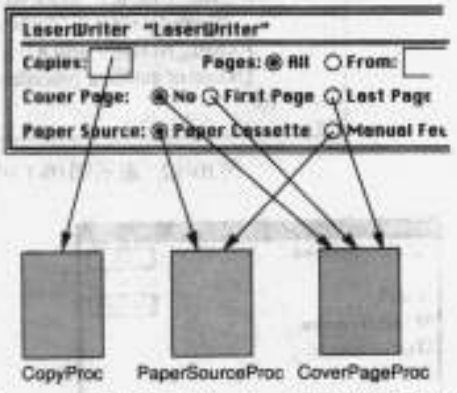


图10-11 与对话框中的widget相关联的回调过程。CopyProc检查并保证每一个字符都是数字的，并且字符输入总数不超过所规定的上限。PaperSourceProc管理进纸来源的单选按钮组，以保证有且只有一个按钮选中并维持当前选择。CoverPageProc为封面页提供类似的单选按钮组功能。（苹果公司提供。）

注意到前面的widget列表包括了低层和高层选项，其中有的是由其他一些选项组合而成。例如，对话框可能包含几个单选按钮组、切换按钮组以及文本输入区域。因此，工具箱包含了一种把widget组合起来的方法，通常是通过子程序调用。图10-12的代码用来说明图10-13中的SunView[SUN86]的对话框的。某些工具箱是用面向对象的编程思想构造的：每一个widget是widget定义的一个实例，可能带有与这个定义相关的方法和属性的替代值。一个组合widget是由多种实例组成的。

用程序设计生成组合widget，不管用什么手法，都是乏味的。图10-14和图10-15所示的交互编辑器，允许快速生成和修改组合widget，促进9.6节所讨论的快速原型的生成。Cardelli开发了一个复杂的交互编辑器，它可以指定widget间的空间约束[CARD88]。在运行时，当用户可以改变对话框的大小时，空间约束可用于保持widget的整齐。

```
print_frame =
  window_create(
    frame,    FRAME,                    {Surrounding box}
    WIN_SHOW,                           TRUE,
    FRAME_NO_CONFIRM,                   TRUE,
    FRAME_SHOW_LABEL,                   TRUE,
    FRAME_LABEL,      "Print",          {Header at top of window}
  0);                                   {Zero means end of list}

print_panel =                           {Panel inside the window}
  window_create(print_frame,            PANEL,
    WIN_ROWS,                           PRINT_WIN_ROWS,
    WIN_COLUMNS,                        PRINT_WIN_COLS,
  0);

print_uickb_name =                      {Header at top of panel}
  panel_create_item(print_panel,        PANEL_MESSAGE,
    PANEL_LABEL_STRING,                 "UICKB: Untitled",
    PANEL_ITEM_X,                       ATTR_COL(PRINT_NAME_COL),
    PANEL_ITEM_Y,                       ATTR_ROW(PRINT_NAME_ROW),
  0);

print_report_choice_item =
  panel_create_item(print_panel,        PANEL_CHOICE,
  {List of mutually exclusive options}
    PANEL_ITEM_X,                       ATTR_COL(PRINT_REPORT_COL),
    PANEL_ITEM_Y,                       ATTR_ROW(PRINT_REPORT_ROW),
    PANEL_LABEL_STRING,                 "Report",
    PANEL_LAYOUT,                       PANEL_VERTICAL,{Or horizontal}
    PANEL_CHOICE_STRINGS,
      "Completeness", "Consistency", "Schema", 0,
    PANEL_NOTIFY_PROC,                  print_report_choice_proc,
    {Name of callback procedure}
  0);
```

图10-12 表示图10-13中的对话框所需的某些SunView代码



图10-13 利用SunView窗口管理系统的工具箱生成的对话框。表示这个对话框的代码如图10-12所示。（乔治华盛顿大学Kevin Murray提供。）

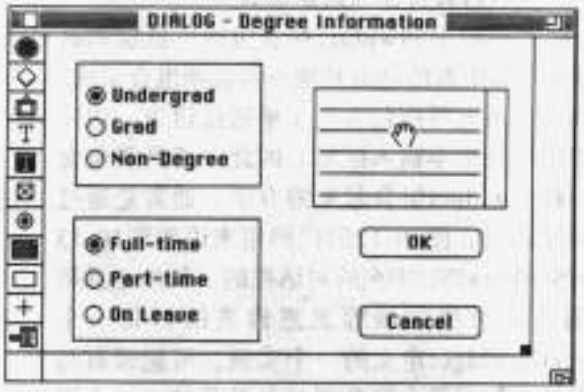


图10-14 Macintosh系统的SmethersBarnes Prototyper对话框编辑器原型。一个滚动列表框正被拖到指定位置。左边的菜单表示可创建的widget，自顶向下分别是按钮、图标、图片、静态文本、文字输入、复选框、单选按钮、滚动列表、矩形框（用于视觉成组，如单选按钮组）、线段（用于视觉分离）、弹出式菜单和滚动条。（由SmethersBarnes公司提供。）

452
|
454



图10-15 窗口设计的交互编辑器。窗口的大小、位置、边框的风格、标题以及退出框的出现等都是可控的。编辑器表示窗口的实际大小；右上角是描述窗口的文本文件，左上角是控制窗口特征的对话框。窗口的大小和位置可以通过直接操纵来修改，此时，对话框中的值是可以修改的。文本文件是窗口设计的永久记录。应用程序可以通过窗口的I.D.和Ref#形成的名字引用窗口。（苹果公司提供。）

这些编辑器的输出是组合widget的一种表示，可以是能转化为编码的数据结构、编码或者编译过的代码。但不管何种情形，所提供的机制就是把组合widget表示连接到应用程序中。使用编辑器不需要编程技巧，人机界面设计人员，甚至熟练的用户，都可以使用。这些编辑器通常就是如图10-1所示的交互设计工具。

创建菜单和对话框的另一种方法就是利用高级程序设计语言描述。在Mickey[OLSE89]中介绍在Macintosh系统中使用的一种扩展的Pascal语言，其中的对话框由一种记录声明来确定。每一项记录的数据类型用于确定在对话框中使用的widget的类型；枚举类型变成单选按钮组，字符串变成了文本输入，布尔量变成了复选框，等等。图10-16是对话框和创建它的代码。交互对话框编辑器可以用来改变widget的位置。图10-17是窗口的菜单及其代码。

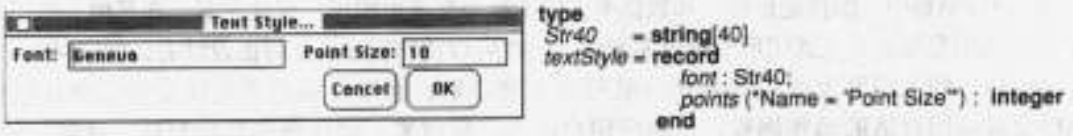


图10-16 由Mickey使用扩展的Pascal记录声明自动创建的对话框。（Brigham Young大学Dan Olsen, Jr.提供。）

Peridot[MYER86；MYER88]使用完全不同的方法生成工具箱。例如，界面设计者交互创建widget和组合widget。设计者不是从生成widget的基本集开始，而是利用交互编辑器来生成有某种视感的对象。Peridot画出要设计的widget的例子，推断出要求widget符合指定环境的关系。例如，从菜单widget可知，它的大小要与菜单选择集中的项数成比率。为了指定widget的行为，如对用户操作做出响应的反馈类型，设计者从Peridot菜单中选择反馈类型，Peridot把这样的方法推广到所有的菜单选项中。




```
procedure NewDrawing (
    (* Menu = File Name = 'New...' Key = N *)
    DrawFile : OutFileDesc);                    {将要显示的对话框的名字}
procedure OpenDrawing (
    (* Menu = File Name = 'Open...' Key = O *)
    DrawFile : InFileDesc);                     {将要显示的对话框的名字}
procedure CloseDrawing;
    (* Menu = File Name = 'Close' Key = W *)
procedure SaveDrawing;
    (* Menu = File Name = 'Save' Key = S *)
procedure SaveDrawingAs (
    (* Menu = File Name = 'Save As...' *)
    DrawFile : OutFileDesc);                    {将要显示的对话框的名字}
```

图10-17 由Mickey使用扩展的Pascal记录声明自动创建的窗口菜单。(Brigham Young大学Dan Olsen, Jr.提供。)

10.6 用户界面管理系统

用户界面管理系统(UIMS)至少可以有助于实现用户界面的外形，而且在一些情形下，也有助于实现部分意图。所有的UIMS提供了定义合理的用户动作序列的方法，而且可以支持整个屏幕设计、帮助和错误信息、宏定义、undo操作和用户文档。最近的一些UIMS也可以管理与应用程序有关的数据。这与交互工具箱形成了鲜明的对比，后者提供的支持非常之少。

455
|
456

UIMS可以提高程序员的工作效率(在一项研究中，交互式程序中有50%的代码是用户界面代码[SUTT78])，加快了整个开发进度，并且在使用过程中有助于用户界面的反复优化。正如在图10-1中所提议的那样，UIMS介于应用程序和交互技术工具箱之间。UIMS越强大，应用程序与操作系统、窗口系统和工具箱进行直接交互的必要性就越小。

在一些UIMS中，由指定操作符和数据类型的编程语言来说明用户界面元素。在其他一些UIMS中，这些说明是通过交互式图形编辑器实现的，这样可以使非程序员的界面设计者访问UIMS。

在UIMS的上层开发的应用程序一般写成子程序集，通常称为动作例程或语义动作例程。UIMS负责调用合适的动作例程，以响应用户输入。反过来，动作例程影响对话，比如，在计算输出的基础上，可通过修改用户下一步要做的事来影响对话。因此，UIMS和应用程序共享对话的控制，这叫作共享控制模型。在UIMS中，动作子程序对对话没有影响，则UIMS称为遵循外部控制模型；控制仅仅存在于UIMS中。外部控制模型不像共享控制模型那样强大。

UIMS在提供给用户界面设计者的特定功能上变化很大，但是一个基本要素是对话序列说明，它用于控制交互技术提供给终端用户的顺序。正是因为这个原因，在下一节，我们将把注意力转向对话序列；然后在10.6.2节，我们将讨论更深入的UIMS概念。有关UIMS更多的背景知识，可参见[HART89；MYER89；OLSE84b；OLSE87]。

10.6.1 对话序列

可以用很多方式来定义用户动作的许可序列：通过转换网络(也称为状态图)、递归转换网

络、事件语言或者通过实例。其中设计者给系统演示允许的动作序列，然后系统弄清楚可能的序列。所有这些方法有一个共同点，那就是用户界面的状态概念和从该状态执行的相关的用户动作。9.3.1节（状态图）和9.4节（模式）讨论了状态这个概念。每一种说明方法都以略微不同的方式对用户界面状态进行了编码。其中每一种方法对一个或多个状态变量的用法进行了推广。

如果要创建一个与上下文相关的用户界面，对用户动作的系统响应肯定依赖于界面的当前状态。对用户动作的响应包括一个或几个动作例程的激发，一个或多个状态变量的变化，以及在对下一个用户动作的准备中激活、禁用或者修改交互技术或菜单项。帮助也依赖于当前状态。因为由动作例程执行的计算结果应该影响用户界面的行为，所以动作例程必须能够修改状态变量。这样，状态是与上下文相关的核心，而与上下文相关是当前用户界面的核心。

最简单的、最不强大的但是很有用的序列指定方法是转换网络或状态图。转换网络有一个简单的状态变量，一个指示当前状态的整数。用户动作导致从一个状态到另外一个状态的转换；每一 457
个转换都有与之联系的零个或多个动作例程，当转换发生时，就要调用动作例程。此外，一旦输入状态，状态就与执行的动作例程联系起来。这种简洁描述对所有送入状态的转换动作都非常方便。

动作例程可以以下面两种方式中的一种影响转换网络的当前状态。第一种方式是，它们可以将事件放在事件队列中，该事件队列反过来驱动交互处理。虽然要保证状态变化是即时的，事件必须放在事件队列的前面，而不是后面，但这种方法隐含地修改了状态。另一种方式是，动作例程可以直接通过简单地给状态变量设置一个新值来改变状态。从软件工程的角度来看，第一种方法比较巧妙，而第二种方法比较灵活但更容易出错。

许多UIMS是基于状态图的[JACO83；JACO85；SCHU85；RUBE83；WASS85]。其中有一些提供交互式转换网络编辑器，这些编辑器使网络指定变得简单。第一个UIMS由Newman开发，并且叫作The Reaction Handler，它就有这样的一个编辑器[NEWM68]。实现一个简单的转换网络驱动UIMS较为容易，可参见习题10.8。

如在9.3.1节中所讨论的那样，转换网络在发现序列的不一致性方面特别有用，并且容易用来确定完成一个任务序列所需的步骤数目。因此，它们也可以充当一个预测特殊设计好坏的方法，甚至可以用在实现一个完整的用户界面之前。例如，考虑一种简单情况：对结果的接受是明确的还是隐含的。图10-18代表具有明确的接受和拒绝的一个操作数的命令。图10-19所示为隐含的接受和明确的拒绝。在第一种情形中，通常需要三个步骤：输入命令，键入操作数，接受。在第二种情形中，通常仅仅需要两个步骤：输入命令，输入操作数，仅当发生一个错误时，才需要三个步骤。减少每个任务的步骤是界面设计中的一个目标，特别是对于熟练用户，因为熟练用户能够输入命令的速度与所需要非连续步骤（击键，手移动）的数目成正比[CARD80]，这一点并不令人惊讶。

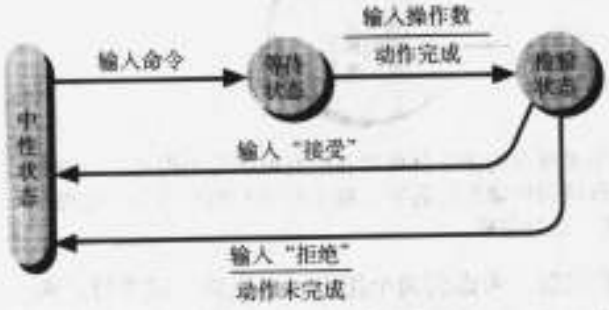


图10-18 用于对话的转换网络，该对话对结果作明确的接受和拒绝处理



图10-19 用户对话的转换网络，该对话对结果作隐含的接受和明确的拒绝处理

然而，转换网络也有不足之处。首先，用户界面状态一般基于许多状态变量，并且必须将这些变量值的所有可能组合映射到一个状态，这种情况比较笨拙而且对用户界面设计者来说不直观。举个例子，当有一个当前选择对象（CSO）时，命令以一种方式表现，当没有当前选择对象（CSO）时，命令以另一种方式表现。状态的数量必须加倍，才能为“CSO存在或不存在”条件编码。这些与上下文有关的类型可以扩展到状态空间，并且使得转换网络难于创建和理解。例如，图10-20显示了一个具有简单应用功能的转换网络，它包含如下命令：

- 选择一个对象（建立一个CSO）。
- 取消选定的CSO（从而没有CSO）。
- 创建一个对象（建立一个CSO）。
- 删除CSO（从而没有CSO）。
- 拷贝一个CSO到剪贴板（需要一个CSO，填充剪贴板）。
- 从剪贴板上进行粘贴操作（需要剪贴板是充满的，创建一个CSO）。
- 清除剪贴板（需要剪贴板是充满的，从而清除剪贴板的内容）。

为剪贴板和CSO的两种可能条件编码需要四种状态。注意到由于在开始状态，对于选择对象的命令，必须有对象存在，因此应该编码来判断是否存在任何对象，这样又需要四个状态。

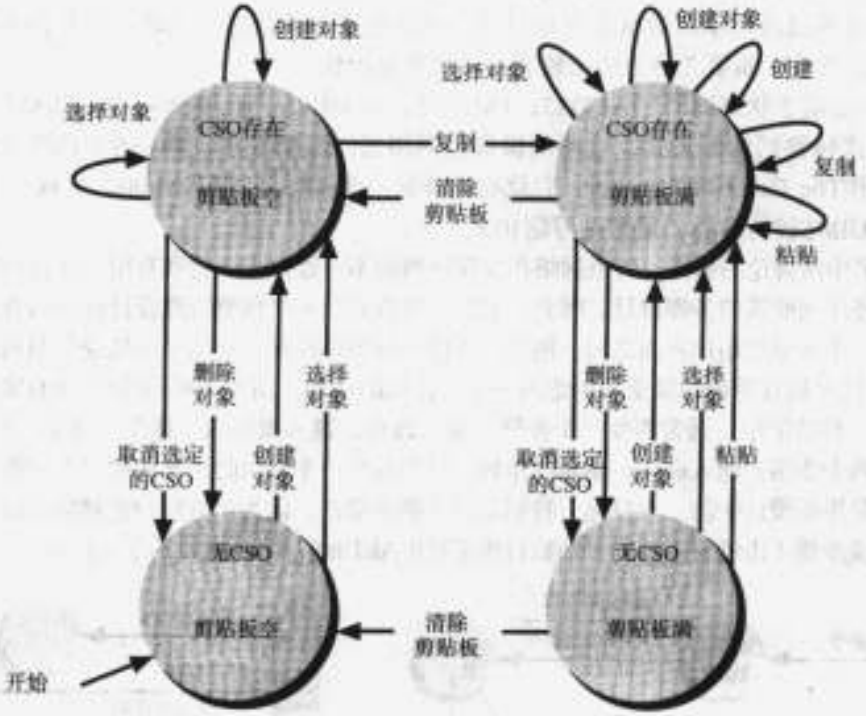


图10-20 一个具有四个状态的转换网络。不是所有的命令都可以在所有的状态中得到。一般地说，与转换有关的动作子程序应该随用户动作的名字（在本例中是用户名令）出现在这种图表中。由于动作是很明显的，此处忽略

并发性导致了一个相似的状态空间增长问题。考虑到两个用户界面元素，或者说，两个并发的活动对话框，每个都有自己的状态，根据最新允许或创建的选择项编码。如果每个对话框编码后有10个状态，它们的组合就需要100个状态；如果是3个对话框，则需要1000个状态，依此类推下去，这种状态空间的爆炸性增长是无法接受的。Jacob[JACO86]把转换网络与面向对

象的程序设计概念结合起来，在限制了状态空间爆炸的同时，定义了完整的用户界面。对象是在界面内自我封装的实体。每个对象有自己的转换网络来定义自己的行为，这些行为是独立于其他对象的。HUTWindows系统的UIMS部分，即赫尔辛基工业大学窗口管理器和UIMS，利用了类似的策略[KOIV88]。

所有可能的命令同样扩大了转换网络。如果帮助是完全可得到的，那么每个状态都必须有一个相应的帮助状态、一个到帮助状态的转换和一个回到原始状态的逆转换。帮助也必须是与上下文有关的。Undo操作也必须是类似的，但下列转换除外，这个转换是从一个undo状态回到一个不同于它进入时状态的状态。当转换数相对于状态数增加时，就用复杂的“空心粉条”式转换网络结束。

为了简化转换网络，我们已经开发出几种特定结构。例如，我们可以通过一个类似于子程序的方式用子网络来减轻帮助问题，并隐藏局部重复的细节。能递归调用子网络的转换网络称为*递归转换网络*。在这种情况下，状态变量是保存状态的整个堆栈加上最近激活的转换网络的状态。其他几种功能强大的图形表示技术都可以从转换网络中导出，详细的论述可参见[WELL89]。

458 ~ 460

Backus-Naur形式（BNF）可以用来定义序列，它与递归转换网络具有相同的表示能力（两者都等同于下推自动机）。如图10-21所示，BNF也可以用图10-22中那样的图表形式来表示。从阅读BNF以便得到序列规则的有效评价是很困难的，但是BNF可以用于提供关于用户界面质量的某些方面的评估[BLES82；REIS82]，或者生成命令语言的分析程序[JOHN78]。一些旧的UIMS是基于BNF说明的[HANA80；OLSE83；OLSE84a]。

```
<command> ::= <create> | <polyline> | <delete> | <move> | STOP
<create> ::= CREATE + <type> + <position>
<type> ::= SQUARE | TRIANGLE
<position> ::= NUMBER + NUMBER
<polyline> ::= POLYLINE + <vertex list> + END_POLY
<vertex_list> ::= <position> | <vertex_list> + <position>
<delete> ::= DELETE + OBJECT_ID
<move> ::= MOVEA + OBJECT_ID + <position>
```

图10-21 一个简单用户界面的序列规则的Backus-Naur形式表示

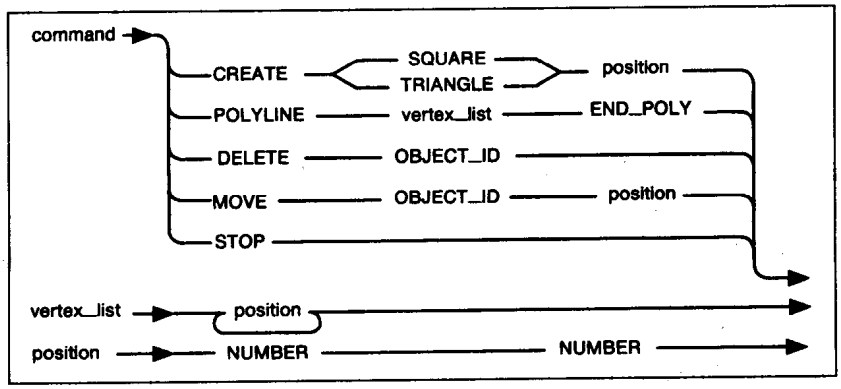


图10-22 与图10-21中的表示等价的Backus-Naur形式的示意图表示

不论是否递归，转换网络用少数状态变量为用户界面的状态编码。*扩充转换网络*（ATN），是一个更灵活的转换网络的派生物，它利用ATN的活动节点和外部状态变量的值为用户界面的状态编码。响应的方式可能是动作子程序的调用、外部状态变量的赋值或者是ATN活动节点的改变。当然，状态变量也可以被动作子程序赋值。图10-23显示了一个ATN，其中*cb*是有几个转换所赋

[461] 值的布尔状态变量，用来影响状态之间的控制流程。如果剪贴板是满的，则变量值*cb*为真。

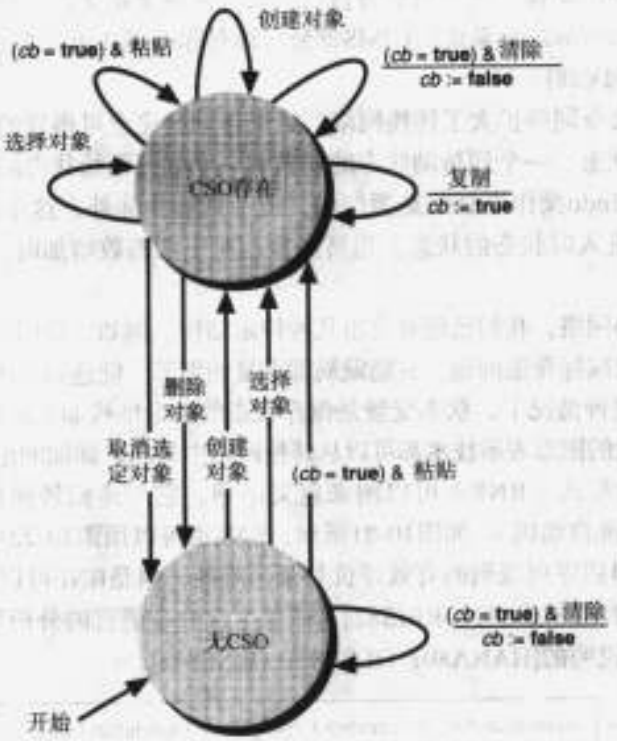


图10-23 扩充转换网络，它表示与图10-20的用户界面相同的用户界面。转换可以有条件地随外部状态变量（在这个例子中是布尔量*cb*）的值而定，也可以设置状态变量。一般来说，应用程序可以设置状态变量

正如转换网络可以被分成更多的子网络一样，ATN也可以调用低层的ATN。可以递归调用别的ATN的ATN称为*扩充递归转换网络*[WOOD70]，研究人员曾用这种网络模拟用户界面[KIER85]。

连同与转换和子程序调用相关的逻辑表达式，转换网络变得越来越复杂，我们采用了更类似于程序的说明。毕竟，程序设计语言是生成指定序列和通常与转换有关的多种条件的最有力的方法。为了用户界面的说明，特别开发了几种*事件语言*[CARD85；FLEC87；GARR82；GREE85a；HILL87；KASI82；SIOC89]。在图10-20和图10-23中描述的用户界面能用一种典型的事件语言来描述，如图10-24所示。注意到事件语言不同于传统的程序设计语言，它没有明确的控制流程；相

[462] 反，只要一个**if**条件为真，相关的动作都会执行。因此，事件语言是一个规则生成系统。

Green[GREE87]综述了事件语言和所有我们以前提到过的序列定义方法，并且表明一般的事件语言要比转换网络、递归转换网络和语法更强大；他还提供了从这些形式到事件语言的转换算法。具有与圆弧相关联的一般计算的ATN，也与事件语言等价。

如果事件语言如此强大，为什么我还要为各种不同的转换网络烦恼呢？因为对于简单的例子，图表表示较简单。UIMS的目标之一是使得从事用户界面设计的非专业编程人员直接创建用户界面。这个目标也许最适合与尽管不是很强、但简单易用的面向转换网络工具相结合。转换网络为设置对话框提供了一个强大的、被时间所证实的工具，它们能帮助设计者证明和理解自己的设计。如果用户操作是在像菜单、对话框和其他可见对象等交互对象上执行的，则图表的表示法是特别勉强的。图10-25中所示的图表类型可以用交互式的方法来创建，以定义对话框序列。如果需要的话，条件（例如图10-23中的*cb* = **true**）可以与圆弧相关。图10-26显示了

在这样的图表上建立链接的一种方法。

```
if Event = SelectObject then
  begin
    cso := true
    perform 动作子程序名
  end
if Event = DeselectCSO and cso = true then
  begin
    cso := false
    perform 动作子程序名
  end
if Event = CreateObject then
  begin
    cso := true
    perform 动作子程序名
  end
if Event = DeleteCSO and cso = true then
  begin
    cso := false
    perform 动作子程序名
  end
if Event = CopyCSO and cso = true then
  begin
    cb := true
    perform 动作子程序名
  end
if Event = PasteClipboard and cb = true then
  begin
    cso := true
    perform 动作子程序名
  end
if Event = ClearClipboard and cb = true then
  begin
    cb := false
    perform 动作子程序名
  end
```

图10-24 一个典型的运用类Pascal语法的事件语言

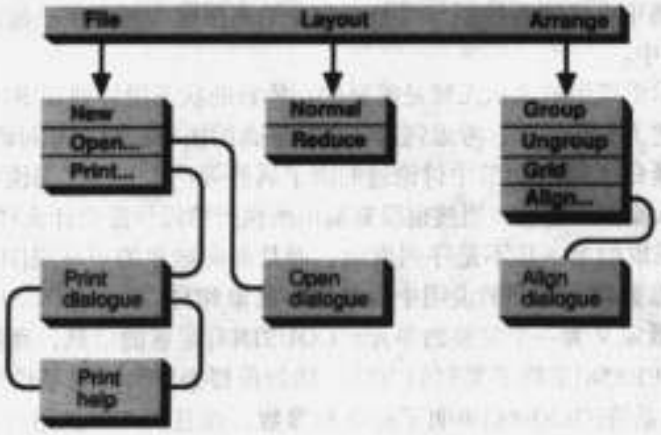


图10-25 一些连在一起的菜单和对话框，没有显示从对话框到主菜单的返回路径

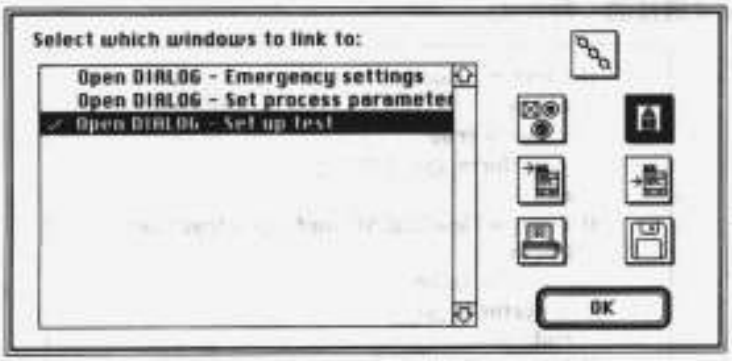


图10-26 用SmethersBarnes Prototyper将不同的交互技术链接在一起。一个菜单项被关联到对话框"Set up test"，该对话框在列表中被选中。用右边的图标来选取链接到菜单选项的响应类，在"链接"图标下面从左往右看，可能是激活或非激活复选框、单选按钮和对话框中的按钮；打开一个窗口或对话框（一类可选取的响应）；激活或禁用一个菜单；激活或禁用一个菜单选项；打开一个打印对话框；或打开一个文件对话框。（经SmethersBarnes公司允许使用。）

定义语法的另外一种相当不同的方法是示例枚举法。这里，用户界面设计者将UIMS用作一种学习模式，然后经过所有可能的动作序列（除非UIMS可以从示例中推断出通用的规则，否则这个过程在复杂应用中很繁琐）。设计者可能从一个主菜单开始，从中选取一个表项，然后通过目录来定位子菜单、对话框或者是应用程序特定的对象。这一对象是用户在选取主菜单时出现的，显示在屏幕上，而且设计者可指示位置、大小或者对象应具有的其他属性，其中的属性由应用程序执行时确定。设计者可以继续在该显示对象上执行一些操作，而且还可以示意该对象下一次应该出现的位置，或者该显示对象对上述操作是如何响应的；设计者重复这个过程直至定义了作用在该对象上的所有动作。该项技术按照用户界面设计者定义的表项序列工作，但还没有通用到可以处理任意的应用程序功能。具有一定程度的示例枚举序列说明的用户界面软件工具有Menulay[BUXT83]，TAE Plus[MILL88c]和SmethersBarnes Prototyper[COSS89]。较早提到过的Peridot，通过示例枚举（也就是，硬件绑定）创建了交互技术。

10.6.2 高级UIMS概念

UIMS倾向于集中在序列控制和视觉设计，转换网络提供了用于序列控制的良好基础，交互式编辑正好用于视觉设计。但是，如同第9章所讨论的那样，用户界面设计包括概念的、功能的、序列的以及硬件绑定等层次。最近的UIMS发展也已经开始强调功能的和概念的设计。因此，已经更多地集中在将序列控制与对象、命令的高层模型绑定在一起，也集中在将智能帮助系统集成到UIMS中。

比转换网络表示更高层的表示无疑是需要的。将新的状态增加到如图10-20所示的转换网络中，来记录是否创建了一些对象，考虑到这样做是多么的困难。对一些对话做修改也是困难的，比如CSO、当前选择命令和在9.4节中讨论过的因子属性等。另外，序列说明并没有提供关于在什么对象上执行什么操作的信息，当然也没有给出所执行的操作需要什么样参数这样一些信息。

COUSIN系统采取的第一步不是序列取向，而是面向抽象的更高层[HAYE83；HAYE84]，它从命令、参数和参数数据类型的说明中自动产生菜单和对话框。COUSIN的创新在于将一个命令所需的所有参数定义为一个完整的单元。COUSIN有足够的信息，而且可以产生前缀和后缀语法。Green[GREE85b]采取了类似的方法，增加前提条件和后置条件来指定用户命令的语义。Olsen 的MIKE系统[OLSE86]声明了命令和参数，而且所产生的用户界面样式与COUSIN类似。此外，MIKE支持对象的直接操纵来确定对象的位置，并且当窗口或子窗口发出按键事

件时，可以引发命令的执行。

463 ≀ 465

所有重要的进展都集中在命令上。但是，在直接操纵界面中，如果UIMS在用户和应用程序之间取折中，那么就必须有被操纵对象的有关知识。HIGGINS是第一个包含数据模型的UIMS[HUDS86；HUDS87；HUDS88]，它是建立在对象以及对象之间的关系上的。该UIMS和动作子程序分享数据模型，这样，对数据对象的改变就会立即在显示中反映出来。*活动值*用来传递相互依赖对象之间以及从对象到它们的视觉表示的变化。乔治华盛顿大学用户界面管理系统(GWUIMS)采用了活动值和面向对象的思想来达到类似的目的[SIBE86]。和Serpent UIMS[BASS88]一样，GWUIMS II 也采用数据模型[HURL89]。虽然具体细节不一样，但所有的数据模型都利用了面向对象的编程思想和活动值，并且在语义数据模型方面，与数据库管理系统中的一些发展密切相关[HULL87]。

用户界面设计环境（UIDE）项目[FOLE89]开发了一个新的用户界面说明，该方法集成了最近发展中的一些成分，包括数据模型、可应用于数据模型中每一种对象的命令、命令所需要的参数、参数数据类型、可使用命令的条件（也就是命令前置条件）以及当执行一个命令时针对状态变量而发生的变化（也就是，命令后置条件）[FOLE87；FOLE88]。为说明该方法，我们首先从在10.6.1节所开发的样例应用程序开始。我们增加一个数据模型，该模型只有一个对象类，它有两个子类：triangle和square。另外，有该对象的两个不同的实例，即CSO和剪贴板对象，两者在给定时刻可能同时存在或者不同时存在。有关说明在图10-27中。前置条件是要激发一个命令所要满足的状态变量条件，而后置条件是在状态变量中的变动。

```
class object                                    {First the data model}
  subclasses triangle, square;
  actions CreateObject, SelectObject;
  attributes position range [0..10] x [0..10]   {Attribute name and data type}
class triangle, square;
  superclass object;
  inherits  actions
  inherits  attributes
instance CSO
  of object
  actions DeselectCSO, DeleteCSO, CopyCSO
  inherits  attributes
instance CB
  of object
  actions ClearClipboard, Paste
  inherits  attributes

{状态变量的初始值}
initial Number (object) := 0; csoExists := false; cbFull := false;

{对象上的动作，带有前置条件、后置条件和参数}
precondition Number (object) ≠ 0;
SelectObject (object);
postcondition csoExists := true;

precondition csoExists := true;
DeselectCSO (CSO);
```

图10-27 含有数据模型、序列信息以及命令参数的用户界面的一个高层说明

```
postcondition csoExists := false;

precondition;
CreateObject (position, object);
postcondition Number (object) := Number (object) +1; csoExists := true;

precondition csoExists := true;
DeleteCSO (CSO);
postcondition Number (object) = Number (object) - 1; csoExists := false;

precondition csoExists := true;
CopyCSO (CSO);
postcondition cbFull := true;

precondition cbFull := true;
Paste (CB);
postcondition csoExists := true;

precondition cbFull := true;
ClearClipboard (CB);
postcondition cbFull := false;
```

图10-27（续）

该说明不仅足以用来自动创建一个可用于应用程序动作子程序的可操作界面，而且它的表示也可以使得

- 利用前置条件，可以激活和禁用菜单。
- 再次利用前置条件告诉用户为什么要禁用一个命令。
- 通过反向链接，确定必须激活哪一个命令以满足所考虑命令的前置条件，告诉用户在激活这个命令时需要做什么。
- 使用后置条件给用户一个命令能做什么的部分解释。
- 检查某些用户界面设计的一致性规则。
- 在指定命令和命令参数方面可使用不同的交互技术。
- 可以预测对不同任务序列和不同交互技术的界面的使用速度。

另一种定义包含互连处理模块的用户界面的方法是利用数据流图。例如，图10-28中所示的NeXT Interface Builder允许对象之间互连，以便一个对象的输出信息是另外一个对象的输入信息。使用类型检查来确保只发送和接受相容信息。

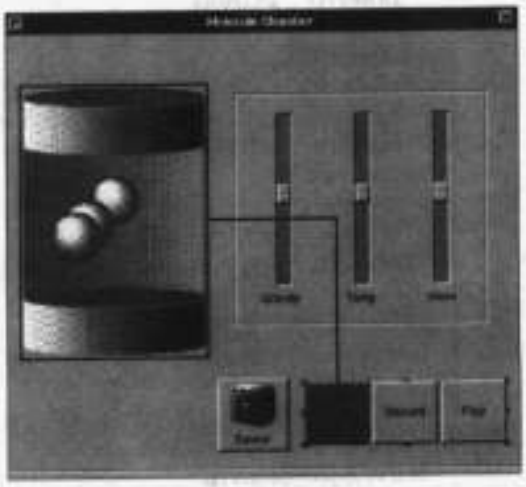


图10-28 NeXT Interface Builder，显示了一个正在生成的连接。用户已经选取了要从Stop按钮发送的信息的类型，并且已经画出了到柱形腔的连接来指示该信息的目的地。（经NeXT公司许可，1989。）

也可用数据流图来指定用户界面的部分或全部的详细行为，虽然这样做需要相当大的编程量，而且会遇到同样在流程图

和转换网络中出现过的比例问题。这方面工作的综述可参见[BORN86a]；一个更近的项目在[SMIT88]中阐述。彩图I-30所示的是用于科学数据可视化的一个专用系统。

UIMS正设法进入日常的应用。早期的UIMS受严格的交互风格的影响，这种风格不允许采取用户定制来满足用户的需要，而且过于依赖转换网络。商用UIMS已得到大规模的使用，而且与图形程序包、窗口管理器以及交互技术工具箱一样，UIMS正成为开发交互图形应用程序的必备工具。

习题

10.1 把用户界面分成两种不同的窗口系统。每一类按照9.3节所讨论的设计问题分类。

10.2 为与主窗口相重叠的基于不动产事件程序来设计搜索机制，其中每一个主窗口中的子窗口可以是空间层次嵌套的。

466 ~ 468

10.3 考察三种窗口管理系统，确定它们是否

a. 带有层次结构的窗口。

b. 实现客户-服务器模型，如果是，该实现是否允许服务器和客户端在网络上分布。

c. 提供基于不动产或监听输入事件调度，或者是其中的某种混合。

d. 是图形软件包和窗口管理系统的集成，或者是直接通过图形硬件进行图形程序调用。

10.4 写出交互对话框或者菜单编辑器。

10.5 用程序设计语言和你所掌握的工具箱实现MICKY[OLSE89]中给出的思想。

10.6 检查你所熟悉的几个用户界面模型。确定在实现每一种用户界面中所使用的用户界面状态变量集。这些状态变量中，有多少用于可提供与上下文有关的菜单、帮助等功能的用户界面中？

10.7 提供用于画笔或制图程序的用户界面对话框文件。要求：(a)带有状态图；(b)使用10.6.1节所介绍的指定的语言。你觉得哪一种方法更容易？为什么？试与你的同学的看法进行比较。回答像“如何画一个圆”，“出现错误后我该做什么？”这样的问题，哪一种方法更容易？

10.8 写出基于转换网络的UIMS。状态图中的每一个转换应该由具有下列信息的状态表表项来表示：

- 当前状态号。
- 下一个状态。
- 导致转换的事件。
- 当转换发生时调用的过程名称。

事件应包括从菜单、命令名类型、鼠标移动、鼠标按键按下、鼠标按下键放开等发出的命令选择。应自动显示包含所有可能命令的菜单（从状态表的事件中导出这一列表），只能从当前状态中激活这些选项。

10.9 对于9.3节所讨论的状态图的每一种扩展，确定修改是否产生了下推自动机（它是有界的还是无界的？）或者图灵机。

10.10 仔细研究包含一个不依赖策略的窗口系统的窗口管理器。检查几种窗口管理器，确定它们是否可以用窗口系统来实现。例如，一些窗口系统在窗口的边框上有滚动条、标题，并且可能还有选择按钮。对于完全不依赖策略的窗口系统，你必须能够分别确定四条窗口边框的宽度。

469

第11章 曲线与曲面的表示

许多计算机图形学的应用系统都要求生成光滑的曲线与曲面。许多现实中物体本身就是光滑的。对现实世界的模拟，正是计算机图形学所要研究的。计算机辅助设计（CAD）、高质量字符的字形、数据的绘图以及艺术家的素描等都包含大量的光滑曲线与曲面。在一段动画（详见第21章）中，物体运动和视角变化的轨迹也几乎总是光滑的；类似地，物体亮度或颜色的变化通常也必须是光滑的（详见第13章和第16章）。

在下面两种情况下，我们需要表示曲线与曲面：模拟已存在的物体（比如一辆汽车、一张脸、一座山等）和根据“草图”进行造型，对于后者，所构造的物体是原先不存在的。在第一种情况中很难找到一种对物体的数学描述。当然，我们可以把物体表示成无限多个坐标点的模型，但这对存储空间有限的计算机来说是不可能的。通常，我们只是通过一些平面、柱面或者其他在数学上很容易表示的形状来近似物体，并让我们的模型上的点尽可能地接近实际物体中相应的点。

在第二种情况中，用户在建模过程中直接构造出原先并不存在的物体，因此这个物体形状将与它在计算机中的表示完全吻合，因为表示只是物体形状的一种表现形式。构造物体时，用户可以交互地构造物体，用数学方法描述它，或者只给出一个大概描述而让某个专门程序去“填充”。在计算机辅助设计中，计算机设计出的抽象物体还要转化为现实的产品。

本章将对*曲面造型*的一般领域做一个总体的介绍。这一领域十分广泛，这里只对其中三种最常用的三维曲面的表示方法进行详述：多边形网格曲面、参数曲面和二次曲面。同时还讨论参数曲线，不仅因为其本身值得研究，而且因为参数曲面就是对参数曲线的一个简单推广。 471

第12章要介绍的*实体造型*是对体的表示，这个体被表面完全包围，比如一个立方体、一架飞机或一幢建筑物等。本章所讨论的曲面的表示方法可以用在实体造型中，去定义包围实体的每一个面。

*多边形网格*是由一系列彼此相连的多边形平面构成。像体可以被平面包围一样，打开的盒子、壁橱、建筑物外表等都可以用平面网格很容易并且很自然地表示出来。多边形网格也可以用来表示表面弯曲的物体，当然，这样的表示比起表示平坦表面的物体困难一些，并且只是近似表示，如图11-1所示。图11-2给出了曲线形状的横截面和表示该形状的多边形网格。这种表示中的误差是显然的，但通过增加更多的多边形的边数，以得到更好的分段线性逼近，可以使误差任意小。但这样也同时增加了对存储空间和处理这种表示的算法执行所需的时间的要求。不仅如此，当图像被放大时，这些误差又会变得十分明显。（Forrest把这个问题称为“几何走样”[FORR80]，以区别于我们在第3章和第14章中讨论的一般的走样概念。）

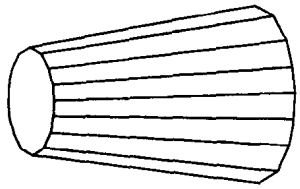


图11-1 用多边形表示的三维物体



图11-2 弯曲物体的横截面及其多边形表示

*参数多项式曲线*使用以t为参数的三个多项式（分别对应x, y和z）来定义三维曲线上的点。选择多项式的系数，以便确定曲线的形状走向。虽然多项式的次数可以是任意的，但我们这里只讨论其中最常用的三次多项式（参数的最高次数为3）。用三次参数多项式表示的曲线通常称为*三次曲线*。

*参数双变量多项式曲面片*使用三个双变量多项式（分别对应x, y和z）来定义曲面上点的坐
472 标。这样的曲面片的边界是参数多项式曲线。在精度给定的条件下，用双变量多项式曲面片来表示弯曲表面比用多边形来表示所用的片数要少得多。但是前者所用的算法也要复杂得多。与曲线情形一样，多项式的次数可以任意，但我们这里只讨论最常用的三次多项式，即两个参数的最高次数为3。相应地，这样的曲面称为*双三次曲面*。

*二次曲面*是用二次方程$f(x,y,z)=0$隐式定义的，其中，f是关于x、y、z的二次多项式。对于我们熟知的球体、椭球体和柱体等，二次曲面是一种很方便的表示方法。

第12章介绍的实体造型将在系统中采用这些表示方法表示物体表面以及被表面包围的（实）体。本章所介绍的几种曲面的表示方法有时也结合起来使用，以围成一个三维实体。

11.1 多边形网格

*多边形网格*是由边和顶点构成的集合。这些多边形彼此相连，每一条边至少属于两个多边形。一条边连接两个顶点，一个闭合的边序列构成一个多边形。一条边可以同时属于两个相邻的多边形，一个顶点至少可以被两条边共享。多边形网格有几种不同的表示方法，它们各有利弊，应用程序的编写者必须从中选择最合适的表示方法。一个应用程序也可以在外部存储、内部结构以及与用户交互构造网格时采用几种不同的表示方法以适应不同需要。

在评价不同的表示方法时有两个基本的标准：时间与空间。对一个多边形网格的典型操作包括搜索一个顶点的所有邻边，搜索共用顶点或共用边的所有多边形，寻找一条边的两个顶点，寻找多边形的边，显示多边形网格，以及识别出表示中的错误（譬如，缺少了一条边、一个顶点或一个多边形）。一般来说，多边形、顶点和边之间的关系越直接，操作的速度就越快，同时所要求的空间也就越大。Woo[WOO85]曾经分析了对一个多边形网格数据结构的九种基本存取操作和九种基本更新操作的时间复杂度。

本节将讨论与多边形网格相关的几个问题：多边形网格的表示，保证给定表示方法的正确性，以及多边形平面系数的计算。

11.1.1 多边形网格的表示

本节将讨论三种多边形网格表示法：直接表示，顶点表指针表示和边表指针表示。在*直接表示法*中，每一个多边形用一个顶点坐标序列来表示：

$$P=((x_1,y_1,z_1),(x_2,y_2,z_2),\ \cdots\ ,(x_n,y_n,z_n))$$

这些顶点按照沿多边形一周的顺序被存储，相邻顶点以及头尾顶点之间都包含一条边。对于单
473 独一个多边形，这样表示的空间效率很高，但对于一个多边形网格来说，公共顶点坐标的重复浪费了大量空间，而且这样做也没有将公共顶点和公共边表示出来。比如，要拖动一个顶点和它的所有邻边，就必须先找到所有包含这个顶点的多边形，这需要将一个多边形的顶点坐标三元组与另外一个多边形的相比较。最有效的办法是将N个坐标的三元组排序，但这个过程的时间复杂度至少是$N\log_2 N$，甚至有时还会由于计算时舍入产生的误差，使在每个多边形中同一顶点的坐标值不完全相等，这样就不可能得到正确的匹配。

在用这种表示法时，不管是用填充多边形还是用多边形边框来显示多边形网格，都必须先

对每个顶点进行坐标变换并对每条边进行裁剪。在画边时，每条公共边会被画两次，这样的重画在笔式绘图仪、胶片记录器和向量显示器中就会产生问题。即使用光栅显示设备，当一条边从相反方向绘制两次时，也可能会出现肉眼可见的多余像素点。

SPHIGS中采用的是另一种表示法，用顶点表指针定义多边形，多边形网格中的每个顶点仅在顶点序列$V=((x_1, y_1, z_1), \cdots, (x_n, y_n, z_n))$中被存储一次，用一个指向这个顶点序列的索引表（指针表）来定义多边形。若一个多边形由顶点3，5，7和10构成，则它被表示为$P=(3, 5, 7, 10)$。

图11-3有一个顶点表指针表示法的例子，这种表示法与直接表示相比有几个优点。因为每个顶点只被存储一次，节省了大量空间，而且这些顶点的坐标可以很方便地进行修改。但是另一方面，这种方法还是没有解决搜索具有公共边的多边形和公共边被画两次这两个问题。下面介绍的方法通过边的直接表示，比较好地解决了这两个问题。

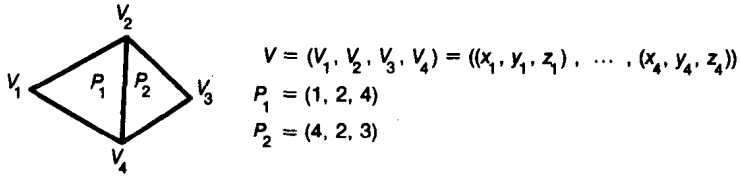


图11-3 用顶点的索引表定义的多边形网格

用边表指针定义一个多边形时，还是要有一个顶点序列V，但表示多边形的指针不是指向顶点序列，而是指向一个边序列。在这个边序列中，每条边只出现一次，边序列中的每条边都指向定义该边的顶点序列中的两个顶点，还指向一个或两个该边所属的多边形。这样，一个多边形描述为$P=(E_1, \cdots, E_n)$，一条边为$E=(V_1, V_2, P_1, P_2)$。若一条边只属于一个多边形，则P_1或P_2为空值。图11-4给出了一个这种表示法的例子。

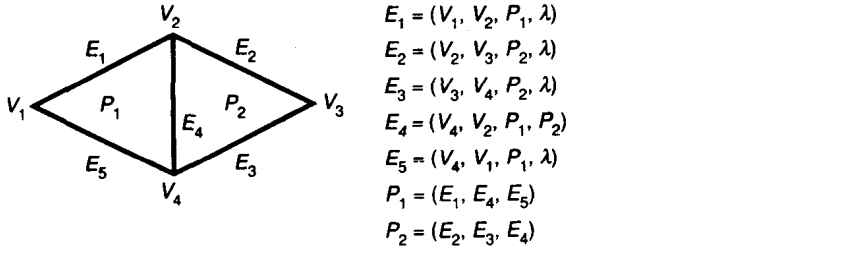


图11-4 用每个多边形的边表定义的多边形网格（λ代表空值）

通过显示所有的边，而不需要显示每一个多边形，就能显示出整个多边形网格的框架，这就避免了冗余裁剪、变换、扫描转换等复杂操作。同样，也容易显示填充的多边形网格。对某些情形，比如三维的蜂窝状钣金结构的表示，一些边由三个多边形共享。这时，可以对边的描述信息进行扩展，使其包含任意多个多边形$E=(V_1, V_2, P_1, P_2, \cdots, P_n)$。

在这三种表示法（直接表示、顶点表指针、边表指针）中，要找到与一个顶点相邻的边，都比较困难：因为要检查所有的边。当然，可以直接添加一些信息来确定点边之间的关系。例如，Baumgart[BAUM75]使用了翼边表示，其中边的描述增加了指向每个多边形中两条邻边的指针，同时顶点描述增加了指向与该顶点相接的（任意）一条邻边的指针，这样，边的描述就包括了更多的多边形与顶点的信息。翼边表示将在第12章中详细讨论。

11.1.2 多边形网格表示法的一致性

多边形网格通常是交互生成的，比如通过数字绘图设备，但这样会有一些误差。因此，必须保证所有多边形是闭合的，所有的边至少使用一次但不超过某个最大使用次数（由具体的应用决定），每个顶点至少被两条边引用。在一些应用中，我们还期望多边形网格是完全连通的（任意两点之间通过一条或多条边相连），是拓扑平面的（相邻顶点的二元关系可以用一个平面图表示出来），或者是无洞的（只存在一个由相连的边序列组成的边界，其中每一条边都被一个多边形使用）。

```
for (边集中的每条边E_j)
    use_count_j = 0;
for (多边形集中的每个多边形P_i)
    for (多边形P的每条边E_j)
        use_count_j++;
for (边集中的每条边E_j) {
    if (use_count_j == 0)
        Error ();
    if (use_count_j > maximum)
        Error ();
}
```

图11-5 直接表示法中所有边至少被使用1次并不超过一个最大使用次数的代码

在以上讨论过的三种表示法中。对边的直接表示法最容易进行一致性检查，因为它包含了大多数的信息。譬如，要保证所有的边至少属于一个多边形，且并不超过某个最大使用次数，只要用图11-5中的代码即可。

这个过程显然还不能构成一个完整的一致性检查。例如，若一条边在同一个多边形出现了两次，这个错误就不能被检查出来。类似的过程可用于保证每一个顶点至少属于一个多边形，这只要检查一个多边形是否有两条边都包含了这个顶点即可。另外，如果多边形边的长度不为零，多边形任一条边的两个顶点相同就是一个错误。

多边形之间“具有公共边”的关系是一个二元的等价关系，由此可以将多边形网格划分成若干个等价类，并称之为连通部分。通常人们期望一个多边形网格只有单个连通部分。确定某个二元等价关系中的连通部分的算法是很著名的[SEDG88]。

还可以进行更详细一些的测试。譬如，检查每一个被某条边E_i引用的多边形同时也引用了这条边E_i，这可以保证从多边形到边的索引是完整的。类似地，可以检查每一条被某个多边形P_i引用的边E_i同时也引用了这个多边形P_i，这可以保证从边到多边形的索引是完整的。

11.1.3 平面方程

在处理多边形或多边形网格时，我们经常需要知道多边形所在的平面的方程。当然，在某些情况下，用交互方法定义多边形时，平面方程是已知的，如果该方程未知，可以用三个顶点的坐标确定一个平面。平面方程为

$$Ax + By + Cz + D = 0 \tag{11-1}$$

系数A，B和C确定了平面的法向量$[A\ B\ C]$。若已知平面上的三个点P_1，P_2，P_3，就可以通过向量叉乘$P_1P_2 \times P_1P_3$（或者$P_2P_3 \times P_2P_1$，等等）计算出平面的法向量。如果这个叉乘为零，那么这三点共线，不能确定一个平面。可能的话，可以用另外一个顶点来代替。由这一非零叉积，将向量$[A\ B\ C]$和三点中的任意一个的坐标代入式(11-1)即可得到D。

如果有三个以上的顶点，由于数值误差或产生多边形的方法所产生的误差，这些顶点可能不在同一平面上。那么就需要另一种技术来确定方程系数A，B，C以使这个平面最接近于所有的顶点。可以证明，A，B和C分别与多边形在(y, z)，(x, z)和(x, y)平面上的投影面积成正比。比如，若多边形与平面(x, y)平行，那么$A = B = 0$，因为多边形在(y, z)和(x, z)平面上的投影面积分别为零。这一方法的优点是投影面积是所有顶点坐标的函数，从而与顶点的选取是否恰好共线或共面的情况关系不大。举例来说，如图11-6所示，多边形在(x, y)平面上投影的面积（从而也

是系数）C等于梯形A_3的面积减去A_1和A_2的面积。一般地，

$$C = \frac{1}{2}\sum_{i=1}^{n}(y_i + y_{i\oplus 1})(x_{i\oplus 1} - x_i) \tag{11-2}$$

其中操作符⊕除$n\oplus 1 = 1$外是正常的加号。A和B的面积也可以用类似的公式求出，注意B的面积是负的（见习题11.1）。

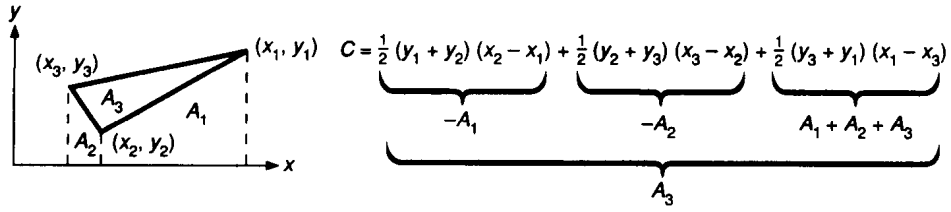


图11-6 利用式(11-2)计算三角形的面积C

由式(11-2)得到所有由多边形的相邻边构成的梯形面积的总和。若$x_{i\oplus 1} < x_i$，则这块面积为负数。总和的符号也是有用的：若顶点是按顺时针方向排列(投影到平面上时)，则符号为正，否则为负。

一旦用所有顶点坐标得到了平面方程，我们可以通过计算每个顶点到平面的垂直距离来估计出这个多边形不共面的程度。从顶点(x, y, z)到平面的距离d为：

$$d = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}} \tag{11-3}$$

这个距离可以是正数或者负数，取决于点在平面的哪一侧。若顶点落在平面上，则$d = 0$。当然，若只需判断点在平面的哪一侧，只要d的符号就够了，所以就没必要除上那个平方根。

平面方程并不是惟一的；方程两边可以都乘上一个常数k，表示的还是同一个平面。通常把平面系数规范后再存储，可以将k设为

$$k = \frac{1}{\sqrt{A^2 + B^2 + C^2}} \tag{11-4}$$

即法向量长度的倒数。这样用式(11-3)计算距离就更容易了，因为分母为1。 477

11.2 三次参数曲线

曲线与曲面可以分别用折线和多边形进行一次线性分段逼近。除非被逼近的曲线与曲面也是分段线性的，否则为了达到一定的精度，就要生成并存储大量的顶点坐标。由于大量的点要精确给定，从而在做形状逼近时，使数据的交互控制变得繁琐。

本节要介绍一种结构更紧凑、更易于控制的分段光滑曲线的表示方法；在下一节，这一方法还将推广到曲面情形。一般的方法就是用比线性函数更高次的函数来表示形状，这种函数仍然只能是近似表示，但比线性函数占用更少的存储空间，并提供更灵活的交互操纵。

这种较高次数的逼近基于以下三种方法。第一种方法是将y和z直接表示成x的显函数，即$y = f(x)$，$z = g(x)$。这种方法的困难在于：(1)由一个x值不能得到多个y值，所以像圆、椭圆这样的曲线必须用几段曲线表示；(2)这一定义不是旋转不变的（描述一个曲线旋转需要很大的工作量，而且可能需要将一段曲线分成若干段）；(3)描述具有与坐标轴垂直的切线的曲线是很困难的，因为这种方法不能表示无穷大的斜率。

第二种方法是用一个形如$f(x, y, z) = 0$的隐式方程的解来表示曲线，但它也有其自身的缺

点。首先，给定方程的解可能比我们想要的多。举例说，描述一个圆时，用方程$x^2 + y^2 = 1$最好不过了，但一个半圆又怎么描述呢？为此，必须加上一些原隐式方程所不包含的限制条件，比如$x \geqslant 0$。另外，用隐式方程定义的曲线段在做连接时，很难确定它们的切线方向在连接点上是否相等，而在很多应用中要求切向连续。

这两种数学形式的共同优点是很容易判断点是否在曲线上或在曲线的哪一侧，正如我们在第3章中所看到的那样。曲线的法线也不难计算。在11.4节中我们还会讨论这种隐式方法。

第三种方法是曲线的*参数表示*，即$x = x(t)$，$y = y(t)$，$z = z(t)$，它解决了函数表示和方程隐式表示引发的问题，同时还具有其他很多优点。参数曲线用参数切向量（不会等于无穷大）代替几何斜率（可能为无穷大）。这里，曲线用*分段多项式曲线*逼近，而不用上节所用的分段线性曲线段。曲线的每一曲线段Q由三个函数定义：x，y和z，其中x，y和z分别是关于参数t的三次多项式。

三次多项式最常用，因为低于三次的多项式在控制曲线形状时不够灵活，而高于三次的多项式又会增加不必要的摆动和更多的计算量。低于三次的多项式不能表示一条通过（插值）两端点
478 并指定其端点处切向的曲线段。给定有四个系数的三次多项式，可以用四个已知条件求得未知系数。这四个条件可以是两个端点以及端点处的导数。类似地，一次多项式（直线）的两个系数由两个端点确定。对于直线来说，端点处的导数由直线本身决定而不能随意变动。平方（二次）多项式有三个系数，只要两个端点再加上另外一个条件，比如斜率或者第三个点，就能确定下来。

另外，三次参数曲线是三维空间中次数最低的非平面曲线。事实上，只要三个点就能完全确定一个二次多项式的三个系数，而这三点也确定了多项式曲线所在的平面。

定义高次曲线需要更多的条件，这样在交互生成时会造成曲线的摆动而难以控制。虽然如此，在一些应用中还是需要使用高次曲线，比如，汽车和飞机的设计，通过对高阶导数的控制来设计出符合空气动力学的曲面。事实上，参数曲线和曲面的数学表达式通常可以是任意次数n，而在本章n为3。

定义曲线段$Q(t) = [x(t)\ \ y(t)\ \ z(t)]$的三次多项式形如

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \qquad 0 \leqslant t \leqslant 1 \end{aligned} \tag{11-5}$$

处理有限曲线段时，不失一般性，可以将参数t限制在[0,1]区间上。

令$T = [t^3\ \ t^2\ \ t\ \ 1]$，并把三个多项式的系数矩阵定义为

$$C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix} \tag{11-6}$$

就可以把方程(11-5)写为

$$Q(t) = [x(t)\quad y(t)\quad z(t)] = T \cdot C \tag{11-7}$$

这是对方程(11-5)的一个简洁表示。

图11-7给出了两条相接的三次参数曲线段及其多项式，图中还说明了虽然参数多项式本身是单值的，但对于一个x值，可以有多个y值与之对应。（曲线的这张图同本节中所有图一样给出的都是用$[x(t)\ \ y(t)]$表示的二维曲线。）

$Q(t)$的导数表示曲线的*切向量*。由式(11-7)，得到

$$\frac{d}{dt}Q(t) = Q'(t) = \left[\frac{d}{dt}x(t) \quad \frac{d}{dt}y(t) \quad \frac{d}{dt}z(t)\right] = \frac{d}{dt}T \cdot C = [3t^2 \quad 2t \quad 1 \quad 0] \cdot C$$

$$= [3a_x t^2 + 2b_x t + c_x \quad 3a_y t^2 + 2b_y t + c_y \quad 3a_z t^2 + 2b_z t + c_z] \qquad (11\text{-}8)$$

479

如果两条曲线段拼接成一条曲线，就称这条曲线具有G^0几何连续。若两条曲线段在拼接点处的切向量方向相同（大小不一定等），则称曲线具有G^1几何连续，计算机辅助设计常常要求曲线段之间具有G^1连续。G^1意味着曲线段在拼接点处的几何斜率相同。若两个切向量TV_1和TV_2方向相同，那么其中一个必定是另一个乘以一个倍数：$TV_1 = k \cdot TV_2$，其中$k>0$[BARS88]。

若两条曲线段在拼接点处的切向量相等（方向和大小都相同），称曲线具有关于参数t的一阶连续，或者说*参数连续*，表示为C^1。若拼接点处的n阶导数$d^n/dt^n\,[Q(t)]$的方向和大小都相同，则称曲线是C^n连续的。图11-8显示了三种具有不同阶的连续性的曲线。注意参数曲线段本身是处处连续的，这里关心的连续性指的是拼接点处。

切向量$Q'(t)$表示了曲线上一个点相对于参数t的速度，而$Q(t)$的二次导数则表示了加速度。假设一台照相机沿着三次参数曲线每隔相同时间间隔记录一张图片，则切向量给出了照相机沿曲线的速度。为了避免在最后的动画序列中出现突然的跳跃，照相机在拼接点前后的速度必须连续。在图11-8中，正是由于通过拼接点时的加速度的连续性，使曲线C^2与C^1在绕到终点前伸得更远。

一般来说，C^1连续蕴含了G^1连续，但逆命题一般不成立。也就是说，C^1连续性比G^1更强，所以具有G^1的曲线可以不具有C^1。然而，具有G^1连续的拼接点与具有C^1的拼接点看起来光滑度也差不多，如图11-9所示。

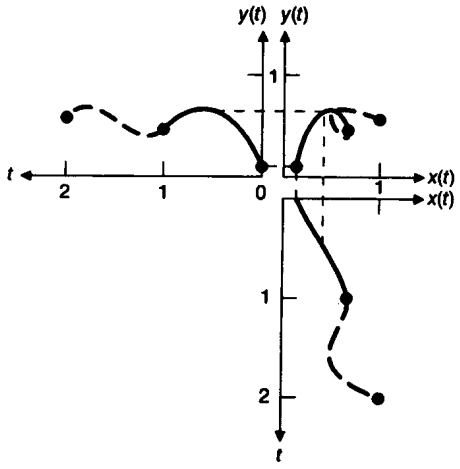


图11-7 两条相接的平面参数曲线及其多项式。(x,y)象限和$x(t)$以及$y(t)$象限之间的虚线表示曲线上的点(x, y)与相应的三次多项式之间的对应关系。第二段曲线的$x(t)$和$y(t)$经过参数变换后不从$t=0$，而从$t=1$开始，这样可以显示出连接点处的曲线连续性

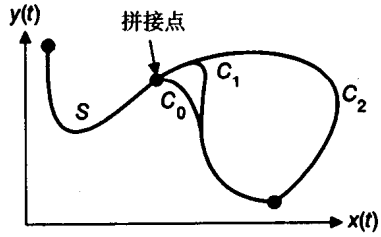


图11-8 曲线段S与曲线段C_0、C_1和C_2在拼接点处分别是0阶、1阶和2阶参数连续。C_1和C_2在拼接点附近差别很小，而远离拼接点时就有明显差异

有一种特殊情况C^1连续性不蕴含G^1：当两条曲线段在拼接点的切向量都为[0 0 0]时，它们的确相等，但它们的方向可以不同（如图11-10所示）。图11-11用另一方法说明这一点。再考虑沿曲线移动的照相机，它在拼接点减速到零，改变方向，然后在新的方向上加速。

参数曲线的坐标系中的表示与一般函数很不一样，后者的自变量标在x轴上，因变量标在y轴上，而在参数曲线的坐标系表示中，自变量t根本就不标出来，这意味着不能仅从参数曲线坐标系中确定曲线的切向量。可以确定切线的方向，但不能确定其大小。若$\gamma(t), 0 \leqslant t \leqslant 1$是一条参数曲线，它在时刻0的切向量是$\gamma'(0)$。如果令$\eta(t) = \gamma(2t)$、$0 \leqslant t \leqslant 1/2$，则$\gamma$和$\eta$在参数坐标系中相同。另一方面，$\eta'(0) = 2\gamma'(0)$，因此，坐标系相同的两曲线可以有不同的切向量。这就

是为什么要定义几何连续性的原因：为了两条曲线的光滑拼接，只要求它们的切向量方向相同，而不要求大小相等。

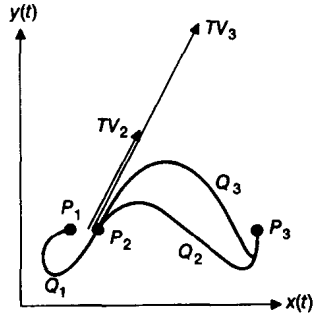


图11-9 曲线段Q_1、Q_2、Q_3在点P_2相接，除了P_2附近的切向量外其余条件都相同。Q_1和Q_2的切向量相等，因而它们在P_2处具有G^1和C^1连续。Q_3和Q_1切向量方向相同，但大小为Q_1的两倍，所以它们在P_2只有G^1连续。切向量越大，曲线Q_3在到达P_3之前沿切向量方向拉伸得越远。向量TV_2是Q_2的切向量，TV_3是Q_3的切向量

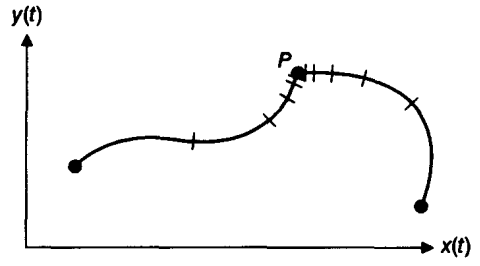


图11-10 C^1连续性不蕴含G^1的一个例子：拼接点P的切向量（沿曲线的参数速度）为零。短线标记了相同时间间隔内沿曲线移动的距离。曲线到达P点时速度减为零，经过P后逐渐增加

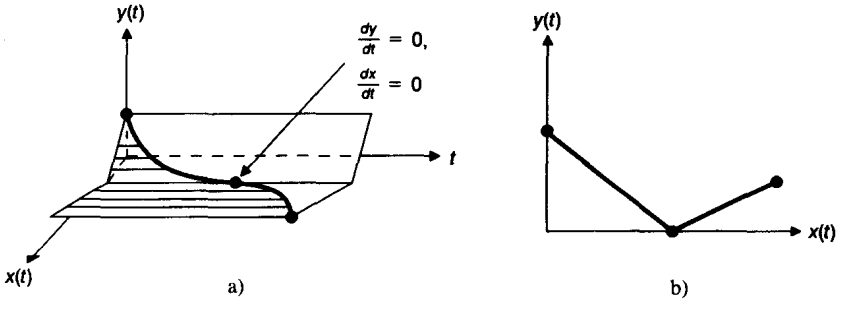


图11-11 a)从三维空间(x, y, t)中看二维三次参数曲线，b)平面中的曲线。在拼接点处参数速度为零，即 $dy/dt = 0$和$dx/dt = 0$。注意到曲线在拼接点附近平行于t轴，因而x和y都没有变化。在拼接点处参数是C^1连续的，但不是G^1连续的

一个曲线段$Q(t)$可以用端点、切向量和曲线段之间连续性等约束条件来定义。由式(11-5)表示的三次多项式有四个系数，所以需要四个约束条件来列出四个方程，然后求解得到。本节主要讨论的曲线有：用两个端点以及两个端点上的切向量定义的Hermite曲线；用两个端点和另外两个控制端点切向量的点定义的Bézier曲线；以及几种由四个控制顶点定义的样条曲线。样条曲线在拼接点处具有C^1和C^2连续性，并靠近它们的控制顶点，但一般不插值这些点。样条曲线的类型有均匀B样条、非均匀B样条和β样条等。

480 ≀ 482

考察怎样用四个约束条件确定式(11-5)中的系数，前面将三次参数曲线定义为$Q(t) = T \cdot C$。把系数矩阵改写成$C = M \cdot G$，其中M是4×4的基矩阵，G是几何约束的四个元素的列向量，称之为几何向量。几何约束就是指限定曲线的一些条件，比如端点或切向量等。用G_x表示由几何向量中的x分量组成的列向量，G_y和G_z有类似的定义。对于不同类型的曲线，可能M不同，也可能G不同，或者两者都不同。

M与G中的元素都是常数，所以乘积$T \cdot M \cdot G$是三个关于t的三次多项式。将积$Q(t) = T \cdot M \cdot G$展开得到

$$Q(t) = [x(t) \quad y(t) \quad z(t)] = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} \tag{11-9}$$

仅仅将$x(t) = T \cdot M \cdot G_x$展开，得到

$$\begin{aligned} x(t) = {} & (t^3 m_{11} + t^2 m_{21} + t\, m_{31} + m_{41}) g_{1x} + (t^3 m_{12} + t^2 m_{22} + t\, m_{32} + m_{42}) g_{2x} \\ & + (t^3 m_{13} + t^2 m_{23} + t\, m_{33} + m_{43}) g_{3x} + (t^3 m_{14} + t^2 m_{24} + t\, m_{34} + m_{44}) g_{4x} \end{aligned} \tag{11-10}$$

式(11-10)强调曲线是几何矩阵中各元素的加权和。每一个权是关于t的三次多项式，称为调配函数。调配函数B由$B = T \cdot M$得到。注意到与分段线性逼近的相似性，分段线性逼近只要两个几何约束（线段的两个端点），每个曲线段是由端点G_1和G_2定义的直线段：

$$\begin{aligned} x(t) &= g_{1x}(1 - t) + g_{2x}(t) \\ y(t) &= g_{1y}(1 - t) + g_{2y}(t) \\ z(t) &= g_{1z}(1 - t) + g_{2z}(t) \end{aligned} \tag{11-11}$$

三次参数曲线其实就是直线段逼近的推广。

为了计算基矩阵M，现在介绍几种特殊类型的三次参数曲线。

11.2.1 Hermite曲线

Hermite曲线（以数学家Hermite的名字命名）由端点P_1、P_4以及端点处的切向量R_1、R_4上的约束确定。（不用下标1、2而用1、4是为了与后面几节的表示一致，后面几节在定义曲线时将用中间点P_2和P_3代替切向量。）

Hermite基矩阵M_H建立了Hermite几何向量G_H和多项式系数之间的联系，为计算Hermite基矩阵M_H，由四个约束条件建立四个关于未知多项式系数的方程，然后求解未知数。 483

将Hermite几何矩阵的x分量G_{H_x}定义为

$$G_{H_x} = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x \tag{11-12}$$

根据式(11-5)和式(11-9)，把$x(t)$改写为

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x = T \cdot C_x = T \cdot M_H \cdot G_{H_x} = [t^3 \quad t^2 \quad t \quad 1]\, M_H \cdot G_{H_x} \tag{11-13}$$

直接代入(11-13)得$x(0)$和$x(1)$上的约束为

$$x(0) = P_{1_x} = [0 \quad 0 \quad 0 \quad 1]\, M_H \cdot G_{H_x} \tag{11-14}$$

$$x(1) = P_{4_x} = [1 \quad 1 \quad 1 \quad 1]\, M_H \cdot G_{H_x} \tag{11-15}$$

就像一般情况下对式(11-7)求导得到式(11-8)一样，现在对式(11-13)求导得到$x'(t) = [3t^2 \; 2t \; 1 \; 0] M_H \cdot G_{H_x}$。因而，切向量约束的方程可写为

$$x'(0) = R_{1_x} = [0 \quad 0 \quad 1 \quad 0]\, M_H \cdot G_{H_x} \tag{11-16}$$

$$x'(1) = R_{4_x} = [3 \quad 2 \quad 1 \quad 0]\, M_H \cdot G_{H_x} \tag{11-17}$$

四个约束方程(11-14)、式(11-15)、式(11-16)和式(11-17)可以写成矩阵形式

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_z = G_{\mathrm{H}z} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} M_{\mathrm{H}} \cdot G_{\mathrm{H}z} \tag{11-18}$$

要满足这个方程（以及相应的y和z的方程），M_{H}必须是式(11-18)中4×4矩阵的逆矩阵

$$M_{\mathrm{H}} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{11-19}$$

M_{H}是惟一确定的，将它代入$x(t) = T \cdot M_{\mathrm{H}} \cdot G_{\mathrm{H}_x}$，得到基于几何向量$G_{\mathrm{H}_x}$的$x(t)$。类似地，$y(t) = T \cdot M_{\mathrm{H}} \cdot G_{\mathrm{H}_y}$，$z(t) = T \cdot M_{\mathrm{H}} \cdot G_{\mathrm{H}_z}$，从而有

$$Q(t) = [x(t) \quad y(t) \quad z(t)] = T \cdot M_{\mathrm{H}} \cdot G_{\mathrm{H}} \tag{11-20}$$

其中G_{H}是列向量

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}$$

484

将$Q(t) = T \cdot M_{\mathrm{H}} \cdot G_{\mathrm{H}}$中的$T \cdot M_{\mathrm{H}}$展开，就得到了Hermite 调配函数$B_{\mathrm{H}}$，用多项式作为权因子对几何向量中各元素做加权和得到：

$$\begin{aligned} Q(t) &= T \cdot M_{\mathrm{H}} \cdot G_{\mathrm{H}} = B_{\mathrm{H}} \cdot G_{\mathrm{H}} \\ &= (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4 \end{aligned} \tag{11-21}$$

图11-12给出了四个调配函数。注意，当$t = 0$时，只有标有P_1的函数不为零：只有P_1能影响曲线在$t = 0$时的值，当t大于零，R_1，P_4和R_4对曲线形状有影响。图11-13给出了四个由几何向量中y分量加权的调配函数、这些调配函数的加权和$y(t)$以及曲线$Q(t)$。

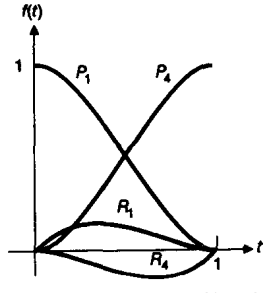


图11-12 Hermite 调配函数，标号表示此调配函数加权的几何向量中相应的分量

图11-14是一组Hermite曲线。它们之间的惟一区别是切向量R_1的长度，而切向量的方向是一定的。切向量越长，对曲线的影响也就越大。图11-15是另一组Hermite曲线，其中的切向量长度一定，但方向不同。在交互式的图形系统中，用户通过控制端点和切向量来得到所需要的形状。图11-16是实现这类交互的一种方式。

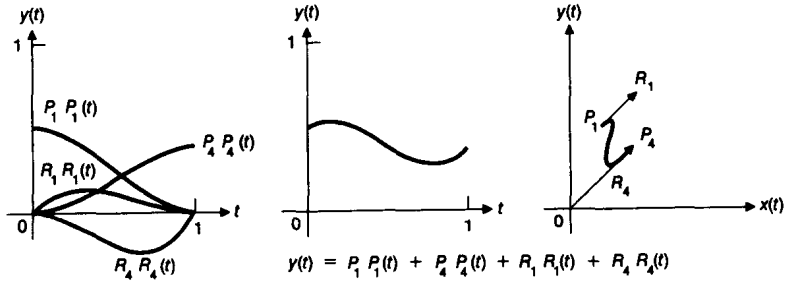


图11-13 Hermite曲线表示几何向量的四个分量与相应的调配函数的乘积(最左边的四条曲线)、加权和$y(t)$、以及二维曲线本身(最右边)。$x(t)$也可由类似的加权和定义

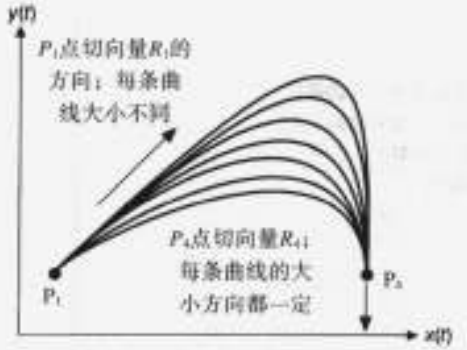


图11-14 一组Hermite三次参数曲线，其中只有在P_1处曲线的切向量R_1大小不同，曲线R_1的大小越大，曲线越高

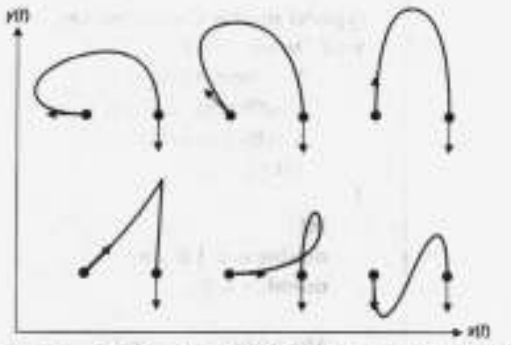


图11-15 一组Hermite三次参数曲线。只有左端点的切向量方向不同；所有切向量大小都相同。减小切向量的大小可以清除曲线的圈

要使每条Hermite曲线共享一个端点并具有G^1（几何）连续性，如图11-17所示，两个几何向量必须有如下形式：

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} \text{且} \begin{bmatrix} P_4 \\ P_7 \\ kR_4 \\ R_7 \end{bmatrix}, \qquad k > 0 \tag{11-22}$$

485 ↓ 486

也就是说，必须有一个公共端点(P_4)和一对至少方向相同的切向量。更严格的C^1（参数）连续性条件则要求$k=1$，即切向量的大小和方向都要相等。

Hermite曲线和其他类似的三次参数曲线的显示较为简单：给定步长δ，取n个连续的t值，计算式(11-5)。图11-18是显示曲线的源代码。在{}之间的计算对每个三维点坐标有11步乘法运算和10步加法运算。利用Horner法则分解多项式化，

$$f(t) = at^3 + bt^2 + ct + d = ((at + b)t + c)t + d \tag{11-23}$$

将每一三维坐标点的计算稍稍简化为9步乘法和10步加法。在11.2.9节还将介绍更有效的曲线显示方法。

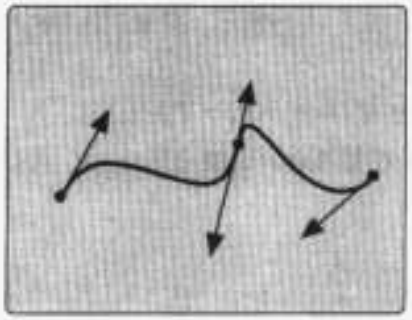


图11-16 两条Hermite三次曲线段，交互控制较为方便。端点可以通过拖动点来重新定位，拖动箭头可以改变切向量。拼接点处的切向量要求共线（以保证C^1连续性）；通常可以由用户给出命令指定C^0，C^1，G^1或不连续。为了显示清晰和方便用户交互，每条曲线的$t=1$端点的切向量为相反方向画出

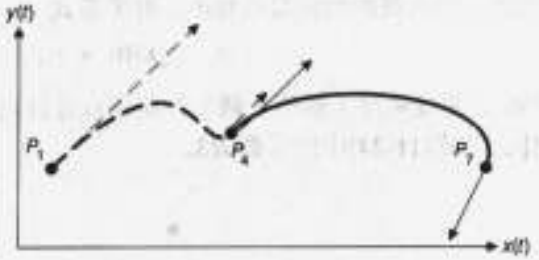


图11-17 在P_4处拼接的两条Hermite曲线。两曲线在P_4处的切向量方向相同但大小不同，使曲线G^1连续但不满足C^1连续性

```
typedef double CoefficientArray[4];
void DrawCurve (
        CoefficientArray cx,          /* x(t): Cx = M · Gx的系数 */
        CoefficientArray cy,          /* y(t): Cy = M · Gy的系数 */
        CoefficientArray cz,          /* z(t): Cz = M · Gz的系数 */
        int n)                        /* 步数 */
{
    int i;
    double δ = 1.0 / n;
    double t = 0;

    MoveAbs3 (cx[3], cy[3], cz[3]);   /* t = 0: 在x(0), y(0), z(0)开始 */
    for (i = 0; i < n; i++) {
        double t2, t3, x, y, z;

        t += δ;
        t2 = t * t;
        t3 = t2 * t;
        x = cx[0] * t3 + cx[1] * t2 + cx[2] * t + cx[3];
        y = cy[0] * t3 + cy[1] * t2 + cy[2] * t + cy[3];
        z = cz[0] * t3 + cz[1] * t2 + cz[2] * t + cz[3];
        DrawAbs3 (x, y, z);
    }
} /* DrawCurve */
```

图11-18 显示三次参数曲线的程序

从式(11-10)可以看出，三次曲线是几何向量中四个分量的线性组合（加权和），因此可以通过变换几何向量来变换曲线，也就是说曲线在旋转、缩放和移动下不变。与把曲线变成一系列短直线段然后变换每个直线段以生成变换曲线相比，这种策略更为有效。但曲线在透视投影下不具有不变性，这在11.2.5节还会讨论。

11.2.2 Bézier曲线

三次多项式参数曲线段的Bézier形式[BEZI70；BEZI74]，以Pierre Bézier命名。它通过给定两个不在曲线上的中间点来间接地确定端点切向量，如图11-19所示。向量P_1P_2和P_3P_4决定了始点和终点处的切向量R_1和R_2，有关系式

$$R_1 = Q'(0) = 3(P_2 - P_1),\ R_4 = Q'(1) = 3(P_4 - P_3) \tag{11-24}$$

Bézier曲线通过（插值）两个控制端点并接近另外两个控制顶点。通过习题11.12，可以知道为什么等式(11-24)中的常数为3。

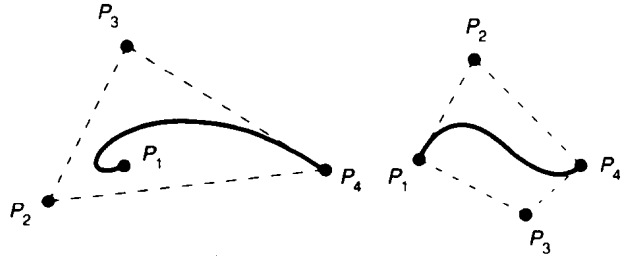


图11-19 两条Bézier曲线及其控制点。注意，虚线表示的控制顶点的凸包并不一定要经过所有四个控制点

Bézier几何向量G_B由四个点构成，即

$$G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} \tag{11-25}$$

下面的等式是式(11-24)的矩阵形式，其中的4 × 4矩阵就是定义Hermite几何向量与Bézier几何向量之间关系式$G_H = M_{HB} \cdot G_B$的矩阵M_{HB}：

$$G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{HB} \cdot G_B \tag{11-26}$$

为了求出Bézier基矩阵M_B，我们将$G_H = M_{HB} \cdot G_B$代入式(11-20)，并定义$M_B = M_H \cdot M_{HB}$得：

$$Q(t) = T \cdot M_H \cdot G_H = T \cdot M_H \cdot (M_{HB} \cdot G_B) = T \cdot (M_H \cdot M_{HB}) \cdot G_B = T \cdot M_B \cdot G_B \tag{11-27}$$

计算乘法$M_B = M_H \cdot M_{HB}$，得

$$M_B = M_H \cdot M_{HB} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{11-28}$$

则乘积$Q(t) = T \cdot M_B \cdot G_B$为

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 \tag{11-29}$$

式(11-29)中作为权因子的四个系数多项式$B_B = T \cdot M_B$称为Bernstein多项式，如图11-20所示。

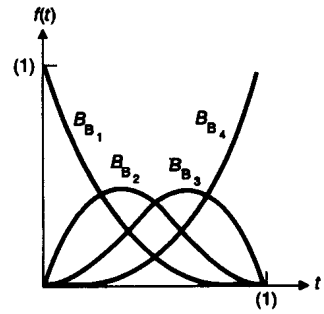


图11-20 Bézier曲线的权函数，Bernstein多项式。当$t=0$时，只有B_{B_1}非零，因而曲线插值P_1；类似地，$t=1$时，只有B_{B_4}非零，所以曲线插值P_4

图11-21给出两条有公共端点的Bézier曲线段。当$P_3 - P_4 = k(P_4 - P_5)$，$k>0$时，端点处是G^1连续的。也就是说，P_3，P_4和P_5三点相异且共线。更严格的情况是，当$k=1$，G^1连续变成了C^1连续。

如果把两段曲线的多项式表示为x^l（左段）和x^r（右段），就可以写出拼接点处具有C^0和C^1的连续性的条件：

$$x^l(1) = x^r(0), \quad \frac{d}{dt}x^l(1) = \frac{d}{dt}x^r(0) \tag{11-30}$$

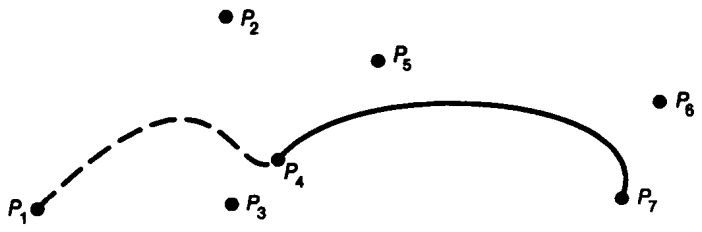


图11-21 两条Bézier曲线的拼接点是P_4。顶点P_3，P_4，P_5共线。这两条曲线与图11-17中的曲线相同

只考虑等式(11-29)中的x分量，得到

$$x^l(1) = x^r(0) = P_{4x}, \frac{d}{dt}x^l(1) = 3(P_{4x} - P_{3x}), \frac{d}{dt}x^r(0) = 3(P_{5x} - P_{4x}) \tag{11-31}$$

同样，相同的条件对y和z也成立。这样，正如预料的那样，当$P_4 - P_3 = P_5 - P_4$时，曲线具有C^0和C^1连续性。

考察式(11-29)中的四个多项式B_B，可知，在$0 \leqslant t \leqslant 1$内每一点，它们的和总为1，而且每一个多项式值都为非负。因此，$Q(t)$就是四个控制顶点的加权平均。这说明每条曲线段都是四个控制顶点分别乘上多项式权值后的和，从而完全落在四个控制顶点的凸包之内。平面曲线的凸包就是四个控制顶点组成的凸多边形：将这个多边形想像成围绕在四个点上的橡皮筋（图11-19)。对于三维曲线，凸包即为控制顶点组成的多面体。将它想像为紧紧包住四个点的一张橡皮膜。

488 ~ 490

如果调配函数非负且和为1，则对所有通过控制顶点做加权和定义的三次曲线，凸包性都成立。一般来说，n个点的加权平均落在这n个点的凸包内；这对$n = 2$或$n = 3$显然成立，对更大的n可归纳证明。由四个多项式和为1这一性质，可得另一个推论：对任意的t，第四个多项式的值等于1减去前三个多项式的值。

凸包性对曲线裁剪也十分有用。我们不用对曲线中每一条直线段进行裁剪以决定其是否可见。首先对曲线凸包或它的包围区域应用多边形裁剪算法。若凸包（包围区域）完全在裁剪区域内，则整个曲线段也在区域内；若凸包（包围区域）完全在裁剪区域外，则曲线段也在区域外；只有当凸包（包围区域）裁剪区相交时才要对曲线本身进行裁剪。

11.2.3 均匀非有理B样条曲线

样条这个词可追溯到绘图人员在设计飞机、汽车或轮船的表面时使用的有弹性的金属长条。通过移动压在样条上的“压铁”，可以沿任意方向推动样条。除非受到太大的压力，否则这些金属样条具有二阶连续性。这些金属样条的数学描述称为自然三次样条，它具有C^0，C^1和C^2连续性并插值控制点。这种曲线的连续性比Hermite曲线和Bézier曲线高一次，从而看起来也更加光滑。

但是，自然三次样条的多项式系数由所有的n个控制点决定，并需要对一个$(n+1) \times (n+1)$的矩阵求逆[BART87]。这就带来了两个缺点：一个控制点的改变将影响整条曲线，对矩阵求逆的计算时间会影响曲线交互调整的速度。

本节讨论的B样条由若干条曲线段构成，每个曲线段的多项式系数仅由少数几个控制点决定，称之为*局部控制*。这样，某个控制顶点的改变只影响曲线的一小部分。同时，计算系数的时间也大大缩短了。B样条具有和自然样条同样的连续性，但不插值控制顶点。

在后面的讨论中，术语会有一点变化，因为我们要讨论的是由几个曲线段构成的一整条曲线，而不是单独的一条曲线段。一条曲线段不需要经过控制点，而且它的两个连续性条件要从相邻的曲线段获得。为此曲线段之间要共用控制点，所以最好把这些曲线段看成一个整体来描述。

三次B样条曲线由$m-2$条三次多项式曲线段$Q_3, Q_4, \cdots, Q_m$构成的曲线来逼近$m+1$个控制点$P_0, P_1, \cdots, P_m$，$m \geqslant 3$。每一条三次曲线段的参数域是$0 \leqslant t < 1$，但通过调整参数域（用$t = t + k$代入）使不同曲线段的参数域相接。这样，Q_i的参数域为$t_i \leqslant t < t_{i+1}$，$3 \leqslant i \leqslant m$，特殊情况$m = 3$时，曲线段$Q_3$的四个控制点是$P_0$到$P_3$，参数域为区间$t_3 \leqslant t < t_4$。

491

对每一个$i \geqslant 4$，在参数值t_i处曲线段Q_{i-1}和Q_i之间有一个拼接点，或叫*结点*；这种点的参数值称为*结点值*。因为t_3与t_{m+1}所定义的始点与终点也是结点，所以总共有$m-1$个结点。图11-22给出了标明结点的平面B样条曲线。容易构造一条闭合的B样条曲线：控制点P_0，P_1和P_2在控制点序列末尾重复一次，即$P_0, P_1, \cdots, P_m, P_0, P_1, P_2$。

这里“均匀”的意思是结点参数t呈相同间隔分布。不失一般性，可以假设$t_3 = 0$且区间$t_{i+1} - t_i = 1$。在11.2.4节将讨论非均匀非有理B样条，它的结点分布是不均匀的。（实际上，本节介绍的结点的概念主要是为非均匀样条做辅垫的。）“非有理”是为了区别从三次有理多项式曲线中得到的样条曲线，其中的$x(t)$，$y(t)$和$z(t)$都定义为两个三次多项式之比，在11.2.5节将讨论这种曲线。“B”代表basis，因为这种样条曲线可以表示成多项式基函数加权和的形式，而自然样条则不可以。

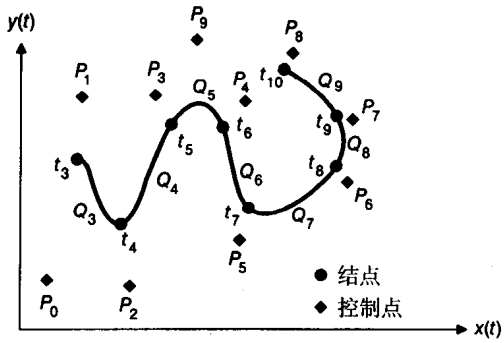


图11-22 由曲线段Q_3到Q_9构成的B样条。此图以及本章中其他许多图的生成程序是由Carles Castellsaquè所写

B样条曲线的$m + 1$个控制点定义了$m - 2$条曲线段，其中的每一条曲线段都由四个控制点定义。特别是，曲线段Q_i由点P_{i-3}，P_{i-2}，P_{i-1}和P_i定义。因此，曲线段Q_i的B样条几何向量G_{Bs_i}为

$$G_{Bs_i} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}, \quad 3 \leqslant i \leqslant m \tag{11-32}$$

第一条曲线段Q_3由P_0到P_3定义，其参数域从$t_3 = 0$到$t_4 = 1$；Q_4由P_1到P_4定义，参数域从$t_4 = 1$到$t_5 = 2$；而最后一个曲线段Q_m由P_{m-3}，P_{m-2}，P_{m-1}和P_m定义，参数域从$t_m = m - 3$到$t_{m+1} = m - 2$。一般来说，曲线段Q_i从点P_{i-2}附近为起点，在点P_{i-1}附近终止。可以证明B样条的调配函数处处非负而且和为1，所以曲线段Q_i限制在它的四个控制点所构成的凸包之内。

正如每个曲线段由四个控制点定义一样，每一个控制点（除控制点序列P_0，P_1，…，P_m的起始点和终止点外）能影响到四条曲线段。沿给定方向移动一个控制点，则受此点影响的四条曲线段都会沿相同方向移动；而其余曲线段则完全不受影响（如图11-23所示）。这就是B样条的局部控制性，本章所介绍的所有样条曲线也都有这样的性质。 492

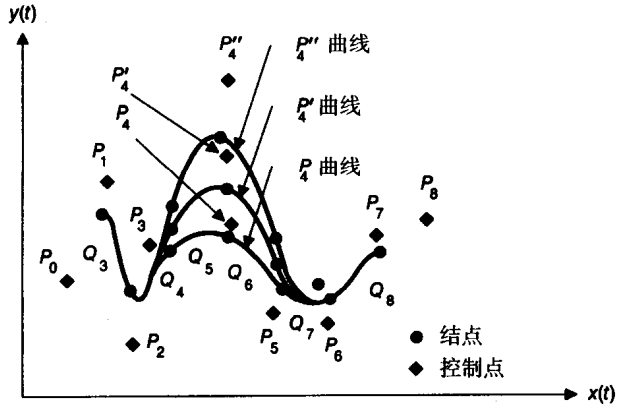


图11-23 当控制顶点P_4在不同位置时B样条曲线的形状

若将T_i定义为行向量$[(t - t_i)^3 \ (t - t_i)^2 \ (t - t_i) 1]$，则第$i$条曲线段的B样条公式为

$$Q_i(t) = T_i \cdot M_{Bs} \cdot G_{Bs_i}, \quad t_i \leqslant t < t_{i+1} \tag{11-33}$$

对$3 \leqslant i \leqslant m$，应用式(11-33)就可得到整条曲线。

B样条基矩阵M_{Bs}建立了几何约束G_{Bs}与调配函数和多项式系数之间的关系：

$$M_{Bs} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{11-34}$$

上述矩阵的推导见[BART87]。

与前面Bézier曲线和Hermite曲线类似，B样条的调配函数B_{Bs}可以通过乘积$T_i \cdot M_{Bs}$得到。注意，因为每条曲线段i的参数值$t - t_i$从$t = t_i$时为0变化到$t = t_{i+1}$时为1，所以每条曲线段的调配函数完全相同。若用$t - t_i$代替t，而且将区间$[t_i, t_{i+1}]$换为$[0, 1]$，就得到

$$\begin{aligned} B_{Bs} = T \cdot M_{Bs} &= [B_{Bs_{-3}} \quad B_{Bs_{-2}} \quad B_{Bs_{-1}} \quad B_{Bs_0}] \\ &= \frac{1}{6}[-t^3 + 3t^2 - 3t + 1 \quad 3t^3 - 6t^2 + 4 \quad -3t^3 + 3t^2 + 3t + 1 \quad t^3] \\ &= \frac{1}{6}[(1-t)^3 \quad 3t^3 - 6t^2 + 4 \quad -3t^3 + 3t^2 + 3t + 1 \quad t^3], \ 0 \leqslant t < 1 \end{aligned} \tag{11-35}$$

493

图11-24是B样条的调配函数B_{Bs}。因为四个函数都非负且总和为1，凸包性对每个曲线段都成立。由[BART87]可知这些调配函数与Bernstein多项式基函数之间的关系。

展开式(11-33)，在第二个等号后将$t - t_i$换成t，得到

$$\begin{aligned} Q_i(t - t_i) &= T_i \cdot M_{Bs} \cdot G_{Bs_i} = T \cdot M_{Bs} \cdot G_{Bs_i} \\ &= B_{Bs} \cdot G_{Bs_i} = B_{Bs_{-3}} \cdot P_{i-3} + B_{Bs_{-2}} \cdot P_{i-2} + B_{Bs_{-1}} \cdot P_{i-1} + B_{Bs_0} \cdot P_i \\ &= \frac{(1-t)^3}{6}P_{i-3} + \frac{3t^3 - 6t^2 + 4}{6}P_{i-2} + \frac{-3t^3 + 3t^2 + 3t + 1}{6}P_{i-1} \\ &\quad + \frac{t^3}{6}P_i, \ 0 \leqslant t < 1 \end{aligned} \tag{11-36}$$

容易证明Q_i和Q_{i+1}之间在拼接点处具有C^0，C^1和C^2连续性。事实上，考察邻接曲线段的x分量$x_i(t - t_i)$和$x_{i+1}(t - t_{i+1})$(y和z分量可类似考虑)，只要证明在结点t_{i+1}处有

$$x_i(t_{i+1}) = x_{i+1}(t_{i+1}), \quad \frac{d}{dt}x_i(t_{i+1}) = \frac{d}{dt}x_{i+1}(t_{i+1}), \quad \text{and} \quad \frac{d^2}{dt^2}x_i(t_{i+1}) = \frac{d^2}{dt^2}x_{i+1}(t_{i+1}) \tag{11-37}$$

同样用t代替$t - t_i$，式(11-37)等价于证明

$$\begin{aligned} x_i|_{t-t_i=1} &= x_{i+1}|_{t-t_{i+1}=0} \\ \frac{d}{dt}x_i|_{t-t_i=1} &= 1\ \frac{d}{dt}x_{i+1}|_{t-t_{i+1}=0} \\ \frac{d^2}{dt^2}x_i|_{t-t_i=1} &= \frac{d^2}{dt^2}\ x_{i+1}|_{t-t_{i+1}=0} \end{aligned} \tag{11-38}$$

为了证明其等价性，考察式(11-36)的x分量和它的一阶及二阶导数，得到

$$x_i|_{t-t_i=1} = x_{i+1}|_{t-t_i=1} = \frac{P_{i-2_x} + 4P_{i-1_x} + P_{i_x}}{6} \tag{11-39}$$

$$\frac{d}{dt}x_i|_{t-t_i=1} = \frac{d}{dt}x_{i+1}|_{t-t_{i+1}=0} = \frac{-P_{i-2_x} + P_{i_x}}{2} \tag{11-40}$$

$$\frac{d^2}{dt^2}x_i|_{t-t_i=1} = \frac{d^2}{dt^2}x_{i+1}|_{t-t_{i+1}=0} = P_{i-2_x} - 2P_{i-1_x} + P_{i_x} \tag{11-41}$$

B样条曲线可以有更强的连续性，但这是以降低控制的灵活性为代价的。通过相重的控制点，曲线可以插值某个控制点；这对曲线的端点和中间点都有用。比如，若$P_{i-2}=P_{i-1}$，曲线就会被这个控制点拉近，因为曲线段Q_i只由三个不同点定义，而$P_{i-2}=P_{i-1}$在式(11-36)中就被权因子乘了两次：一次是B_{Bs-2}，另一次是B_{Bs-1}。

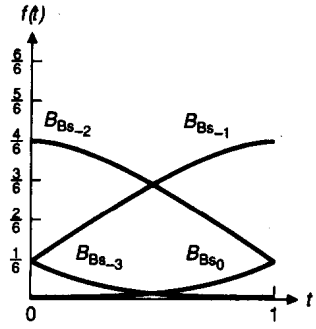


图11-24 式(11-35)的四个B样条调配函数。当$t=0$或$t=1$时，恰有三个函数非零

若一个控制点被使用三次，比如，$P_{i-2}=P_{i-1}=P_i$，则式(11-36)就变成了

$$Q_i(t)=B_{Bs-3}\cdot P_{i-3}+(B_{Bs-2}+B_{Bs-1}+B_{Bs0})\cdot P_i \tag{11-42}$$

Q_i显然是一条直线。而且，在$t=1$时直线插值P_{i-2}，此时P_i的三个系数和为1，但一般地，$t=0$时曲线不插值P_{i-3}。也可以这样想，Q_i的凸包现在只定义在两个不同点上，所以Q_i只能是一条直线。图11-25显示了多重控制点对B样条曲线内部的影响。在图11-25c中曲线通过所有控制点，此时曲线就失去了G^1连续性，虽然式(11-40)表明C^1连续性依然保持（值为零）。正如11.2节所讨论过的，这是一个C^1连续性不蕴含G^1连续性的例子。

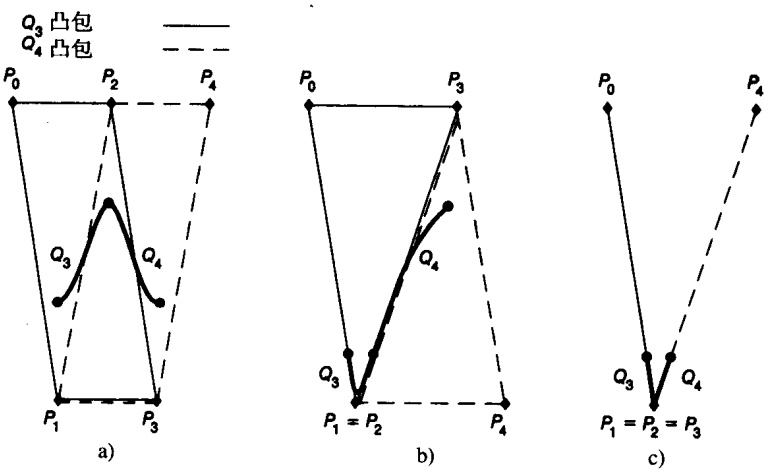


图11-25 均匀B样条曲线上多重控制点的影响。a)没有多重顶点，两曲线段的凸包有重叠；曲线段Q_3和Q_4之间的拼接点在两凸包的公共区域内。b)中有一个二重控制点，两凸包共享一条边P_2P_3，曲线段间的拼接点落在共享边上。c)有一个三重控制点，两凸包是直线段，在三重控制点处相接，因此，两曲线段之间的拼接点也是此三重控制点。因两凸包是直线段，所以，两曲线段也必须是直线段。在拼接点处是C^2参数连续的，但只是G^0几何连续

在[BARS83；BART87]中介绍了另一种插值端点（虚顶点）的方法。下一节，用非均匀B样条可以比用均匀B样条更自然地使曲线插值端点或中间点。

11.2.4 非均匀非有理B样条曲线

非均匀非有理B样条曲线与前面介绍的均匀非有理B样条曲线的区别在于相邻结点值所取的参数间隔不一定是均匀的。不均匀的结点值序列意味着每一段曲线上的调配函数不再相同。

这样的曲线比均匀B样条有几个优点。首先，选定拼接点上的连续性可以从C^2减为C^1或C^0，甚至没有连续性。若连续性减为C^0，则曲线可以插值一个控制点而不会产生像均匀B样条那样

控制点两边的曲线都退化成直线的现象。其次，曲线可以恰好插值给定的始点与终点而不会出现变成直线段的曲线段。另外，对非均匀B样条还可以添加结点和控制点，这样很容易改变一条曲线的形状，而这对均匀B样条是做不到的。我们在11.2.7节还将深入讨论。

由于非均匀B样条比均匀B样条具有更一般的特性，需要把均匀B样条中用到的表示方法稍微变化一下。与前面一样，非均匀B样条也是由三次多项式曲线构成的分段连续曲线，逼近控制点P_0到P_m。结点值序列是一个从t_0到t_{m+4}的单调不减序列（也就是说，结点数要比控制点数多四个）。因为控制点至少为四个，所以最小的结点序列也至少有八个结点值，而且曲线定义在从t_3到t_4的参数区间。

结点序列的惟一限制是单调不减，也就是允许相邻结点值相等。若相等，则参数值称为重结点，而相等的参数值的数目称为结点的重数（单独一个结点的重数为1）。举个例子，在结点序列（0, 0, 0, 0, 1, 1, 2, 3, 4, 4, 5, 5, 5, 5）中，结点值0重数为4；结点值1的重数为2；值2和3的重数都为1；值4的重数为2；值5的重数为4。

494
l
496

曲线段Q_i由控制点P_{i-3}，P_{i-2}，P_{i-1}，P_i和调配函数$B_{i-3,4}(t)$,$B_{i-2,4}(t)$,$B_{i-1,4}(t)$,$B_{i,4}(t)$确定，可表示成加权和的形式

$$Q_i(t) = P_{i-3} \cdot B_{i-3,4}(t) + P_{i-2} \cdot B_{i-2,4}(t) + P_{i-1} \cdot B_{i-1,4}(t) + P_i \cdot B_{i,4}(t)$$
$$3 \leqslant i \leqslant m, \quad t_i \leqslant t < t_{i+1} \tag{11-43}$$

在从t_3到t_{m+1}的区间以外，曲线没有定义。当$t_i = t_{i+1}$时（即重结点情形），曲线段Q_i是一个点。正是这种允许曲线段退化为点的表示方法，使非均匀B样条具有更多的灵活性。

与其他类型的样条曲线不同，非均匀B样条的调配函数不惟一。它们依赖于结点值之间的区间，由低阶的调配函数递归定义。$B_{i,j}(t)$是第j阶调配函数，作为控制点P_i的权因子。因为我们讨论的是四阶（三次）B样条，递归定义终止于$B_{i,4}(t)$，并且很容易表示成嵌套形式。三次B样条的递归形式如下：

$$B_{i,1}(t) = \begin{cases} 1, & t_i \leqslant t < t_{i+1} \\ 0, & \text{其他状态} \end{cases}$$
$$B_{i,2}(t) = \frac{t - t_i}{t_{i+1} - t_i} B_{i,1}(t) + \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} B_{i+1,1}(t)$$
$$B_{i,3}(t) = \frac{t - t_i}{t_{i+2} - t_i} B_{i,2}(t) + \frac{t_{i+3} - t}{t_{i+3} - t_{i+1}} B_{i+1,2}(t) \tag{11-44}$$
$$B_{i,4}(t) = \frac{t - t_i}{t_{i+3} - t_i} B_{i,3}(t) + \frac{t_{i+4} - t}{t_{i+4} - t_{i+1}} B_{i+1,3}(t)$$

图11-26以结点向量（0, 0, 0, 0, 1, 1, 1, 1）为例说明了怎样用式(11-44)来求出调配函数。图中还表明了为什么计算四个调配函数时要用八个结点向量。$B_{3,1}(t)$在区间$0 \leqslant t < 1$上等于1，其余的$B_{i,1}(t)$等于0。$B_{2,2}(t)$和$B_{3,2}(t)$是锯齿形的，是两点之间作线性插值时的调配函数。类似的，$B_{1,3}(t)$，$B_{2,3}(t)$和$B_{3,3}(t)$是二次的，是进行二次曲线插值时的调配函数。对于这个结点向量，$B_{i,4}(t)$正好是Bernstein多项式，也就是说，是Bézier曲线的调配函数，将其与图11-20比较。另外，对于这个结点向量，曲线插值控制点P_0和P_3，实际上就是一条Bézier曲线，它在端点处的切向量由向量P_0P_1和P_2P_3决定。

调配函数的计算很费时。通过限制B样条结点序列的间隔为0或者为1，则可以存储少数几个对应于式(11-44)的矩阵，使它们覆盖所有的可能结点取值。这样就不用对每一个曲线段都计算一遍式(11-44)。

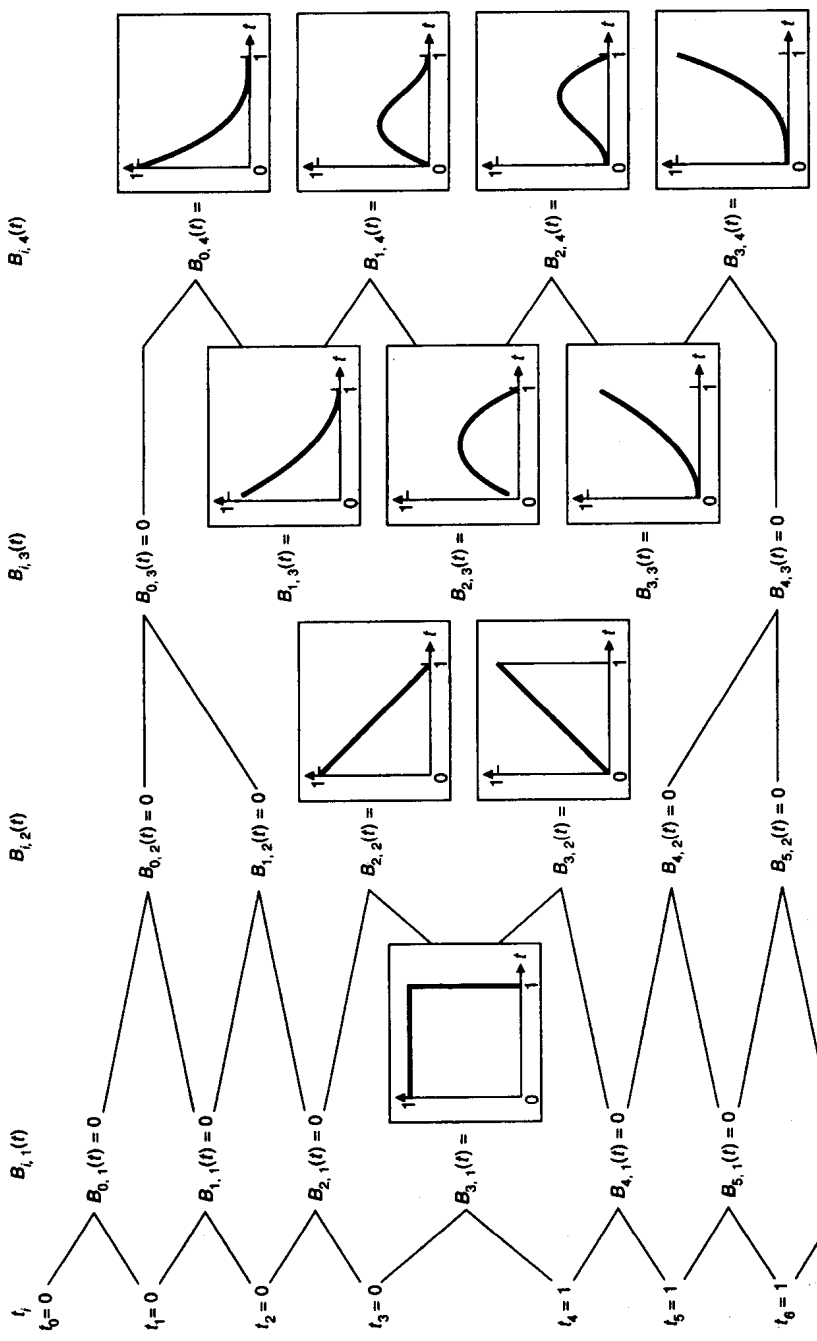


图11-26 由式(11-44)定义的结点矢量 (0,0,0,0,1,1,1,1) 与调配函数$B_{i,1}(t)$，$B_{i,2}(t)$，$B_{i,3}(t)$，$B_{i,4}(t)$之间的关系

可以证明调配函数是非负且和为1的，所以非均匀B样条曲线完全在它们的四个控制点的凸包之内。对于重数大于一的结点，由于相邻结点值相等，分母有可能为零：因此规定除以零的结果为零。

增加结点的重数有两种效果。首先，每个结点值t_i自动处于点P_{i-3}，P_{i-2}和P_{i-1}的凸包之内。若t_i和t_{i+1}相等，则它们必须同时处于P_{i-3},P_{i-2},P_{i-1}凸包和P_{i-2},P_{i-1},P_i凸包中。这就是说它们必须恰好在线段P_{i-2}, P_{i-1}上。同样，若$t_i = t_{i+1} = t_{i+2}$，则这个结点必须在点P_{i-1}上。若$t_i = t_{i+1} = t_{i+2} = t_{i+3}$，则结点必须在$P_{i-1}$和$P_i$上——曲线就断开了。第二，重结点会降低参数连续性：增加一个结点（重数为2），连续性从C^2降为C^1；增加两个结点（重数为3），连续性从C^1降为C^0；增加3个结点（重数为4），则曲线从C^0连续变为不连续。

图11-27深入分析了一个特例。图11-27a表示的是所有结点的重数为1的情况。每一个曲线段由四个控制点和四个调配函数定义，相邻的曲线段共用三个控制点。比如，曲线段Q_3由点P_0，P_1，P_2，P_3定义；曲线段Q_4由点P_1，P_2，P_3，P_4定义；曲线段Q_5由点P_2，P_3，P_4，P_5定义。图11-27b表示的是重结点的情况，$t_4 = t_5$，曲线段Q_4长度为零。这样Q_3和Q_5相邻但只共用两个控制点P_2和P_3；因此这两段曲线段缺少了一些“共性”，连续性降低一次。图11-27c是三重结点情形，只有控制点P_3被共用，由两段曲线同时插值。因为只有一个控制点相同，所以拼接点只有一个约束条件，具有C^0连续性。图11-27d是重数为4的结点，曲线断开，没有连续性。因此，几条分离的样条可以用一个统一的结点序列和控制点序列表示。图11-28可以增加对结点、曲线段和控制点之间关系的理解。表11-1总结了重控制点和重结点的影响。

表11-1 重控制点和重结点的影响对比

重 数	重控制点	重 结 点
1	C^2G^2*	C^2G^2*
2	C^2G^1	C^1G^1
	结点限制到小凸包内	结点限制到较少控制点的凸包内
3	C^2G^0	C^0G^0
	曲线插值三重控制点	曲线插值控制点
	拼接点两边的曲线段连续	能控制拼接点两边曲线段的形状
4	C^2G^0	
	曲线插值四重控制点	有曲线的不连续性
	拼接点两边的曲线段是线性的并插值拼接点两边的控制点	曲线停在一个控制点上，并从下一个继续
		能控制不连续处两边曲线段的形状

497 ~ 499

图11-29描述了用重结点方法所能表示的形状的复杂度。注意到图11-29a中结点序列为（0，0，0，0，1，1，1，1）：曲线插值两个端点但不插值两个中间点，它是一条Bézier曲线。另两条曲线在起始点和终止点都是三重结点。这样端点处的切向量由向量P_0P_1和$P_{m-1}P_m$确定，该曲线在两端点处有Bézier曲线一样的控制。

典型的交互生成非均匀样条曲线方法，要求给定控制点的位置，只要连续选中同一个点就能指定一个重控制点。图11-30给出了一种交互指定结点值的方法。另一种方法是用一个多键鼠标直接点曲线：双击一个键表示一个双重控制点；双击另一个键表示一个双重结点。

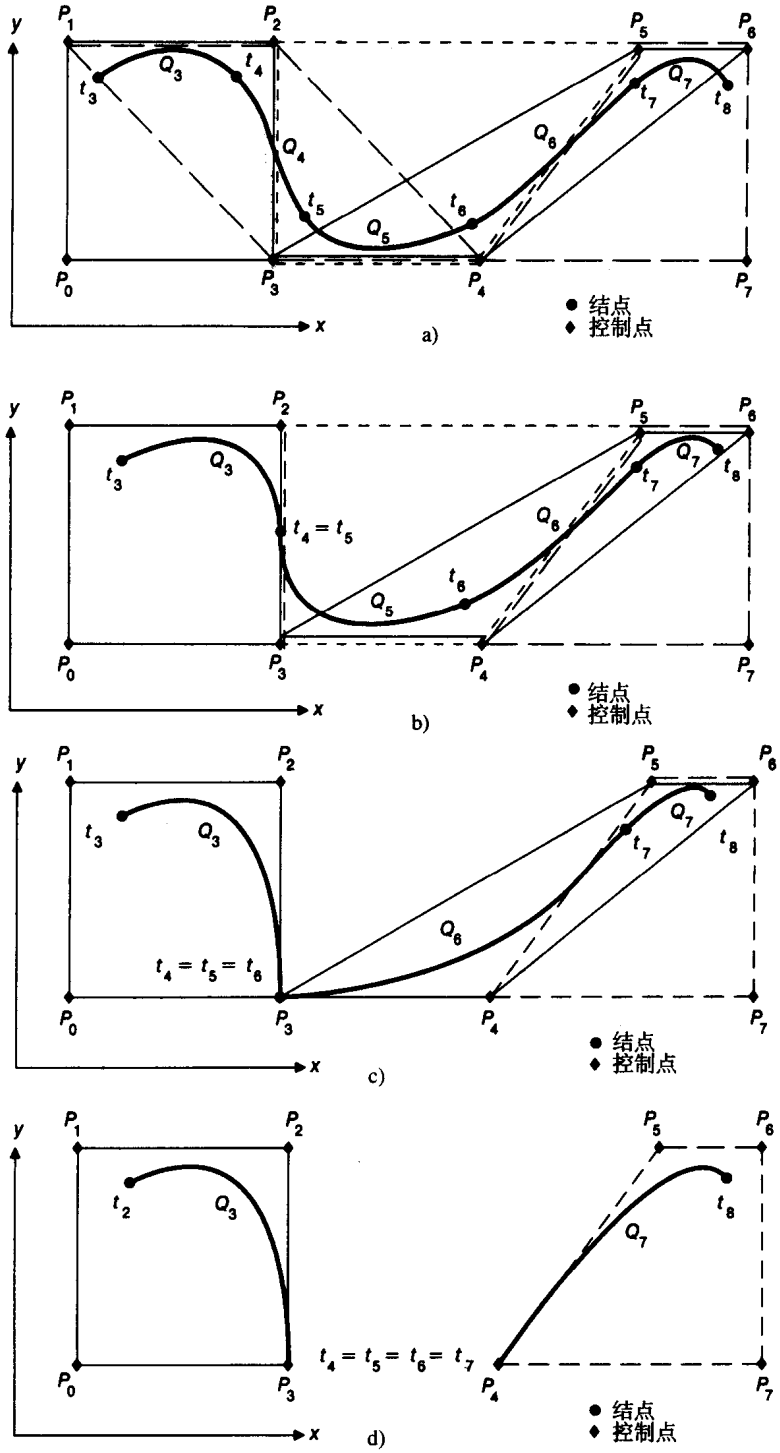


图11-27 重结点的影响。a)结点序列为（0，1，2，3，4，5），没有重结点；曲线段之间的拼接为C^2和G^2连续。图中也显示了包含每条曲线的凸包。b)结点序列（0，1，1，2，3，4），有一个重结点，所以曲线段Q_4退化为一个点。包含Q_3和Q_5的凸包在边P_2P_3处相接，连曲线的拼接点也落在此边上。拼接点处只有C^1和C^2连续性。c)结点序列（0，1，1，1，2，3），有一个三重结点，所以Q_4和Q_5退化成点。包含Q_3和Q_6的凸包仅在控制点P_3处相接，拼接点也落在P_3上。两个曲线段仅有公共控制点P_3，具有C^0连续性。d)结点序列（0，1，1，1，1，2），有一个四重结点，因为曲线段Q_3和Q_7没有公共控制点，曲线不连续

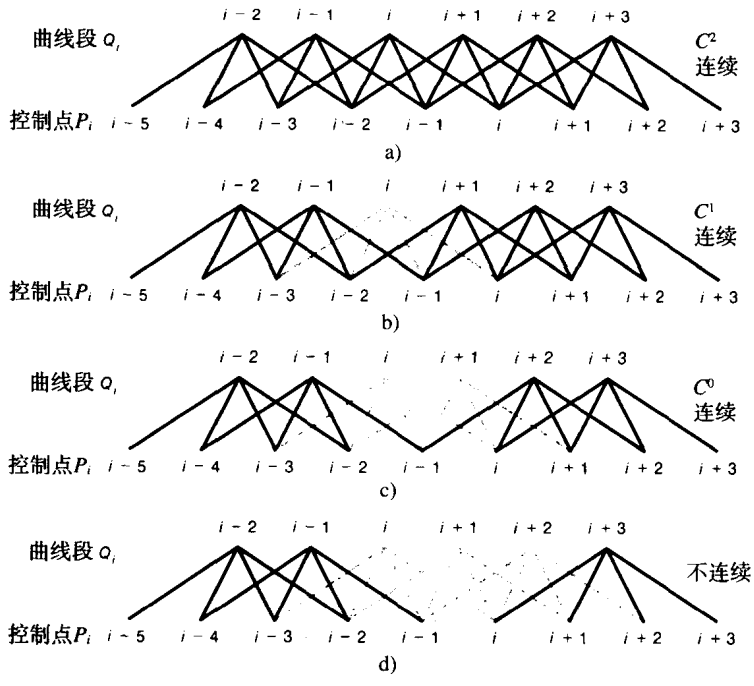


图11-28 非均匀B样条的曲线段、控制点和重结点之间的关系。直线段连接的是曲线段和它的控制点；灰线表示的是由于结点间隔零（结点的重数大于1）而退化了的曲线段，在显示时没有这一曲线段。a)所有结点都是单重结点。b)有一个双重结点，所以曲线段i没画出。c)有一个三重结点，所以有两个曲线段没画出；于是控制点$i-1$被相邻曲线段$i-1$和$i-2$共有。d)有一个四重结点，曲线段$i-1$和$i+3$没有公共的控制点，使曲线在控制点$i-1$和i之间断开

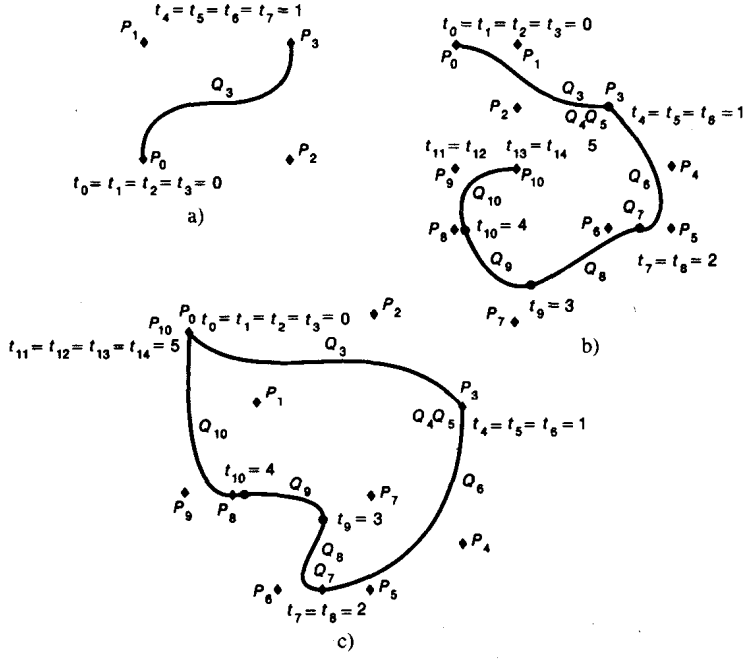


图11-29 用非有理B样条和重结点定义的曲线的例子。a)是一段Bézier曲线，结点序列为（0, 0, 0, 0, 1, 1, 1, 1），因此只有一个曲线段Q_3。b)和c)的结点序列相同，为（0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 4, 5, 5, 5, 5），但控制点不同。每条曲线都由曲线段从Q_3到Q_{10}组成。曲线段Q_4、Q_5和Q_7被定位在重结点，长度为零

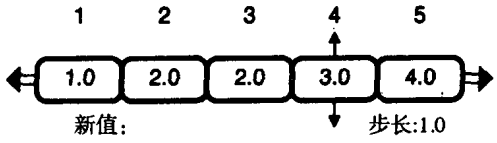


图11-30 由Carles Castellsaguè提出的指定结点值的交互方法。结点序列的一部分显示出来，并且可以用水平箭头左右滚动。光标选中的一个结点值可以用垂直箭头增大或减小，改变量由值的步长设定。选中的结点值也可以由新输入的值替换

11.2.5 非均匀有理三次多项式曲线段

一般的有理三次曲线段是多项式的比：

$$x(t) = \frac{X(t)}{W(t)},\quad y(t) = \frac{Y(t)}{W(t)},\quad z(t) = \frac{Z(t)}{W(t)} \tag{11-45}$$

其中$X(t)$，$Y(t)$，$Z(t)$和$W(t)$都是三次多项式曲线，它们的控制点定义在齐次坐标上。还可以将曲线想像成定义在齐次空间$Q(t) = [X(t), Y(t), Z(t), W(t)]$上。通常，从齐次空间变到三维空间要除以$W(t)$。任一个非有理曲线可以通过增加第四个元素$W(t) = 1$变成有理曲线。一般，有理曲线中的多项式可以是Bézier、Hermite或其他形式的。如果是B样条，就得到非均匀有理B样条曲线，有时称为NURBS曲线[FORR80]。

有理曲线有两个优点。首先，也是最重要的是，它在控制点的旋转、缩放、平移以及透视变换下具有不变性（非有理曲线只在旋转、缩放、平移下有不变性）。也就是说，只要对控制点运用透视变换就能对原曲线进行透视变换。如果在透视变换前不想把非有理曲线先转化为有理曲线，就得对曲线的每个点做透视变换，这样效率就要低得多。类似地可以观察到，对一个球体做透视变换与只对原来球体的球心和半径做变换所得的球是不同的。

500 ~ 503

有理曲线的第二个优点是可以精确定义圆锥曲线。这一点与非有理曲线不同，圆锥曲线可以用非有理曲线逼近，但需要用许多接近圆锥曲线的控制点。这个优点在一些应用领域，特别是CAD领域，非常有用。在那里，同时需要一般的曲线、曲面以及圆锥曲线，两种实体类型都能用NURBS曲线表示。

定义圆锥曲线只要用二次多项式就够了，而不需要三次多项式。因此，调配函数$B_{i,3}(t)$可以从递归式(11-44)中推出，曲线的形式是

$$Q_i(t) = P_{i-2}B_{i-2,3}(t) + P_{i-1}B_{i-1,3}(t) + P_iB_{i,3}(t)$$
$$2 \leqslant i \leqslant m,\quad t_i \leqslant t < t_{i+1} \tag{11-46}$$

图11-31给出了两种以原点为圆心生成单位圆的方法。注意到用二次B样条曲线时，一个二重结点使得曲线插值一个控制点，而三重结点使曲线始点和终点插值控制点。

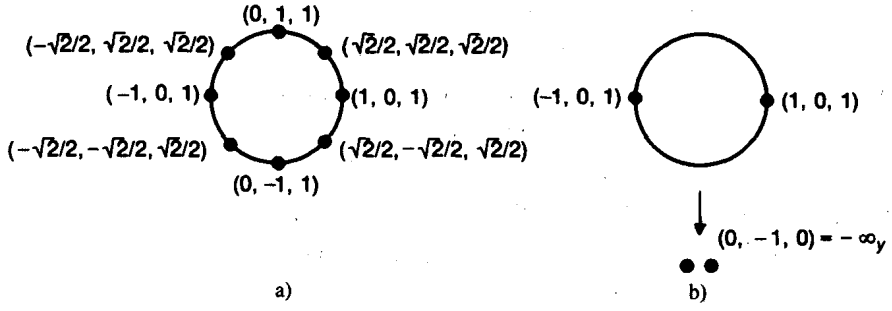


图11-31 用平面有理样条曲线的控制点定义一个圆。坐标系是（X, Y, W）。结点向量是（0，0，0，1，1，2，2，3，3，4，4），初始控制点和末端控制点重合。P_0可以任意选择

圆锥曲线和NURBS曲线的进一步讨论，可以参阅[FAUX79；BöHM84；TILL83]。

11.2.6 其他样条曲线

我们经常用一条光滑曲线插值（通过）一个给定的点列。点列可以是从一个数据表中读入或用鼠标生成，或者是一条曲线或摄影机路径所经过的一个三维点列。对这些情况，插值或逼
504 近样条的Catmull-Rom样条族[CATM74a]，也叫Overhauser样条[BREW77]，非常有用。这个样条族中的其中一种能够插值点列从P_0到P_m中从P_1到P_{m-1}的点。另外，P_i点的切向量与线段$P_{i-1}P_{i+1}$平行，如图11-32所示。但这些样条曲线不具有凸包性。自然（插值）样条曲线也可以插值给定点，但没有Catmull-Rom样条那样的局部控制。

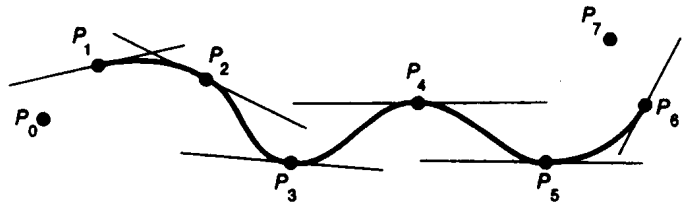


图11-32 Catmull-Rom样条曲线。曲线插值给定的点，其中每一点的方向与连接两个相邻点的直线平行。图中的直线段表示了这些方向

令M_{CR}为Catmull-Rom基矩阵，与B样条曲线一样，利用式(11-32)中的几何矩阵G_{Bsi}，表达式为

$$Q^i(t) = T \cdot M_{CR} \cdot G_{Bsi} = \frac{1}{2} \cdot T \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix} \tag{11-47}$$

[BARR88b]中给出了一种快速显示Catmull-Rom曲线的方法。

另一种样条曲线是形状均一β样条[BARS88；BART87]，它有另外两个参数β_1和β_2，对曲线形状做进一步控制。所使用的几何矩阵G_{Bsi}与B样条相同，同时具有凸包性。β样条基矩阵M_β为

$$M_\beta = \frac{1}{\delta} \begin{bmatrix} -2\beta_1^3 & 2(\beta_2 + \beta_1^3 + \beta_1^2 + \beta_1) & -2(\beta_2 + \beta_1^2 + \beta_1 + 1) & 2 \\ 6\beta_1^3 & -3(\beta_2 + 2\beta_1^3 + 2\beta_1^2) & 3(\beta_2 + 2\beta_1^2) & 0 \\ -6\beta_1^3 & 6(\beta_1^3 - \beta_1) & 6\beta_1 & 0 \\ 2\beta_1^3 & \beta_2 + 4(\beta_1^2 + \beta_1) & 2 & 0 \end{bmatrix} \tag{11-48}$$

$$\delta = \beta_2 + 2\beta_1^3 + 4\beta_1^2 + 4\beta_1 + 2$$

第一个参数β_1称为偏移参数，第二个参数β_2称为紧绷参数。若$\beta_1 = 1$且$\beta_2 = 0$，则M_β退化为B样条基矩阵M_{Bs}（式(11-34)）。当β_1超过1时，在点的参数增加一侧，沿切向量方向，曲线有“偏移”，或者说影响较大，如图11-33所示。当β_1小于1时，偏移方向相反。当紧绷参数β_2增大
505 时，曲线向控制点的连线靠近，如图11-34所示。

这种β样条曲线称为形状均一的，是因为对所有曲线段，β_1和β_2的取值相同。β_1和β_2的全局影响有悖于样条的局部控制性质。连续形状β样条和离散形状β样条对每个控制点取不同的β_1和β_2值，对曲线的影响是局部的，而非全局的[BARS83；BART87]。

虽然β样条比B样条有更一般的形状表示，它在拼接点处具有G^2连续，但只有C^0连续。这在几何造形中并不算缺点，但在动画运动路径中会引起速度不连续。在特殊情况$\beta_1 = 1$，β样条

具有C^1连续性。

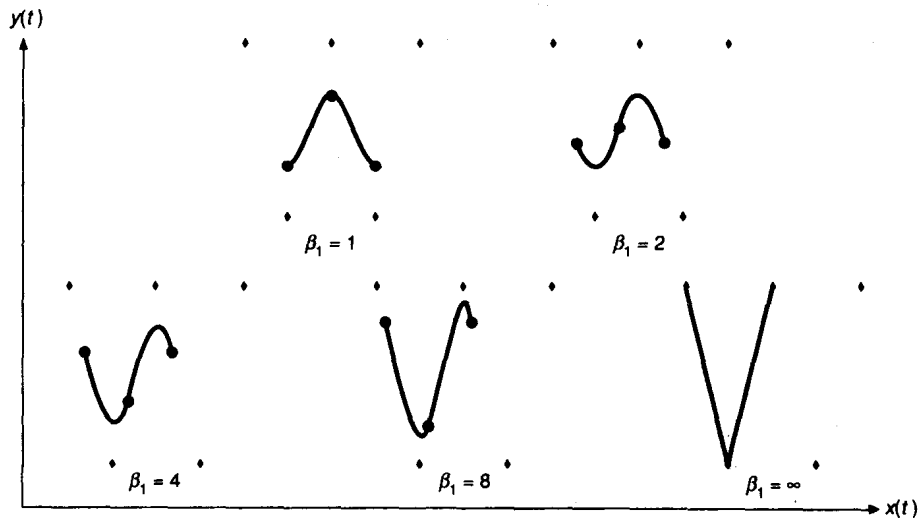


图11-33 偏移参数β_1从1到无穷大变化对形状均一β样条的影响

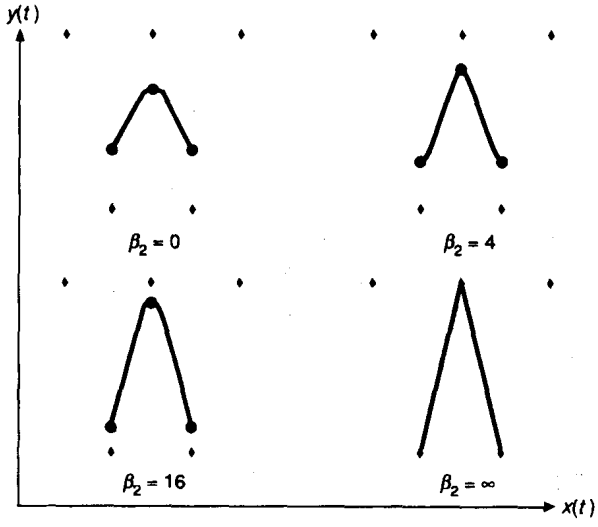


图11-34 紧绷参数β_2从0到无穷大变化对形状均一β样条曲线的影响

Kochanek和Bartels提出了一种Hermite曲线的变化形式，在动画中对运动路径的控制很有用[KOCH84]。点P_i处的切向量R_i为

$$R_i = \frac{(1 - a_i)(1 + b_i)(1 + c_i)}{2}(P_i - P_{i-1}) + \frac{(1 - a_i)(1 - b_i)(1 - c_i)}{2}(P_{i+1} - P_i) \quad (11\text{-}49)$$

P_{i+1}处的切向量R_{i+1}为

$$\begin{aligned} R_{i+1} &+ \frac{(1 - a_{i+1})(1 + b_{i+1})(1 - c_{i+1})}{2}(P_i - P_{i-1}) \\ &+ \frac{(1 - a_{i+1})(1 - b_{i+1})(1 + c_{i+1})}{2}(P_{i+1} - P_i) \end{aligned} \quad (11\text{-}50)$$

如何计算基矩阵M_{KB}，可参见习题11.16。

参数a_i、b_i、c_i在点P_i附近以不同方面控制曲线走向：a_i控制曲线的弯曲程度；b_i相当于β样条的偏移参数β_1，c_i控制P_i的连续性。其中最后一个参数用来模拟一个物体在从另一个物体弹开方向上的快速变化。

11.2.7 曲线分割

假设你已经创建了一列相连的曲线段来逼近你正在设计的形状。可以操纵控制点来调整曲线形状，但仍可能得不到所要的曲线形状。也许是由于没有足够多的控制点来达到预期的效果。有两种方法可以增加控制点的数目。第一种是升次：样条曲线的次数从3次升到4次或更高。这种调整有时很必要，尤其是需要更高的连续性时。不过这样会使曲线中互相影响的点增多，而且增加了额外的计算时间，通常不采用。这部分内容已超出了我们讨论的范围，要了解更多细节可以参阅[FARI86]。

第二种更有用的方法是将一条或几条曲线段再分割为两段。比如，一条四个控制点的Bézier曲线可以划分为总共有七个控制点的两段（两段新曲线有一个公共控制点）。这两条新曲线段与原曲线段完全吻合，除非移动某一个控制点，这样一条或两条新曲线段就不再与原曲线段吻合。

506
~
507

对于非均匀B样条曲线，更一般的方法称为细化，用于向曲线加入任意数量的控制点。进行分割的另一个原因是为了曲线或曲面的显示。在11.2.9节讨论分割的终止条件时还要详细介绍。

给定一条由控制点P_1、P_2、P_3、P_4定义的Bézier曲线$Q(t)$，我们想要找到其左边的曲线，由点L_1、L_2、L_3、L_4定义，以及右边的曲线，由点R_1、R_2、R_3、R_4定义，使得左曲线与Q在区间$0 \leqslant t<1/2$内一致，同时右曲线与Q在区间$1/2 \leqslant t<1$内一致。Casteljau[DECA59]提出了一种几何构造方法，可以在任意参数t处分割Bézier曲线。对于参数值t，做构造线L_2H，将P_1P_2和P_2P_3按比率$t:(1-t)$分成两段，类似地，HR_3也将P_2P_3和P_3P_4按比率$t:(1-t)$分成两段，再做L_3R_2，同样地分割L_2H和HR_3。这样，点L_4（就是点R_1）按上述同样比值将L_3R_2分为两段，它就是曲线上的点$Q(t)$。图11-35给出了当$t=1/2$时的构造过程。

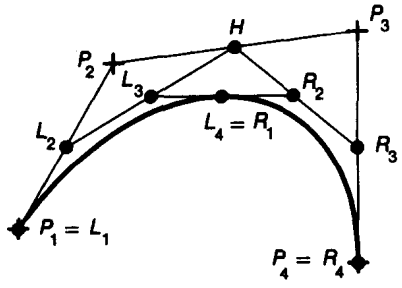


图11-35 由控制点P_i定义的Bézier曲线，在$t=1/2$处被分为两段，左曲线由点L_i定义，右曲线由点R_i定义

通过加法和移位（被2除）就能很容易地计算出所有的点：

$$L_2=(P_1+P_2)/2,\ H=(P_2+P_3)/2,\ L_3=(L_2+H)/2,\ R_3=(P_3+P_4)/2$$
$$R_2=(H+R_3)/2,\ L_4=R_1=(L_3+R_2)/2 \tag{11-51}$$

以上结果可以重新写成矩阵形式，这样就可以求出Bézier左分矩阵D_B^L和Bézier右分矩阵D_B^R，通过它们可以分别求出左（右）Bézier曲线的几何矩阵$G_B^L(G_B^R)$：

$$G_B^L=D_B^L\cdot G_B=\frac{1}{8}\begin{bmatrix}8&0&0&0\\4&4&0&0\\2&4&2&0\\1&3&3&1\end{bmatrix}\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix}$$
$$G_B^R=D_B^R\cdot G_B=\frac{1}{8}\begin{bmatrix}1&3&3&1\\0&2&4&2\\0&0&4&4\\0&0&0&8\end{bmatrix}\begin{bmatrix}P_1\\P_2\\P_3\\P_4\end{bmatrix} \tag{11-52}$$

注意到每一个新控制点L_i和R_i都是点P_i的加权和，并且系数都是正的且总和为1。因此，每一个新控制点都是在原控制点所构成的凸包内。所以这些新控制点不会比原控制点离曲线$Q(t)$更远，而一般会离得更近。这种变差缩减性质对所有具有凸包性的样条曲线都成立。还可以注意到D_B^L和D_B^R之间的对称性，以及几何构造中的一系列对称性。 508

在$t = 1/2$处分割Bézier曲线通常可以满足用户交互的需要，但如果能让用户任意指定一个分割点就更好了。给定曲线上的一个点，很容易估算出相应的t值。这样就可以像前面所描述的那样进行分割了，只是构造线应该按$t:(1-t)$的比例分割（见习题11.22）。

分割B样条曲线时，相应的矩阵D_{Bs}^L和D_{Bs}^R是

$$G_{Bs_i}^L = D_{Bs}^L \cdot G_{Bs_i} = \frac{1}{8}\begin{bmatrix}4&4&0&0\\1&6&1&0\\0&4&4&0\\0&1&6&1\end{bmatrix}\begin{bmatrix}P_{i-3}\\P_{i-2}\\P_{i-1}\\P_i\end{bmatrix}$$

$$G_{Bs_i}^R = D_{Bs}^R \cdot G_{Bs_i} = \frac{1}{8}\begin{bmatrix}1&6&1&0\\0&4&4&0\\0&1&6&1\\0&0&4&4\end{bmatrix}\begin{bmatrix}P_{i-3}\\P_{i-2}\\P_{i-1}\\P_i\end{bmatrix} \tag{11-53}$$

仔细观察这两个等式可以发现，G_{Bs_i}中的四个控制点被总数为五个的新控制点所代替，这五个新的控制点由$G_{Bs_i}^L$和$G_{Bs_i}^R$共用。但是，分割后两侧的样条曲线段仍然由G_{Bs_i}的部分控制点定义。因此，改变五个新控制点中的任一个或四个原控制点都会使B样条曲线断开。用非均匀B样条曲线可以避免这一问题。

分割非均匀B样条不像只要指定两个分割矩阵那么简单，因为它们没有一个统一的基矩阵，它们的基矩阵是递归定义的。一个基本方法是插入结点。已知一个由控制点P_0，…，P_n定义的非均匀B样条曲线以及要加入的结点值$t = t'$（从而加入一个控制点），我们要确定定义同一条曲线的新的控制点Q_0，…，Q_{n+1}。（t'值可以由用户交互决定，见习题11.21。）若t'值满足$t_j < t' \leqslant t_{j+1}$。Böhm[BÖHM80]已证明新控制点可以表示为

$$\begin{aligned} Q_0 &= P_0 \\ Q_i &= (1-a_i)P_{i-1} + a_iP_i, \quad 1 \leqslant i \leqslant n \\ Q_{n+1} &= P_n \end{aligned} \tag{11-54}$$

其中a_i为

$$\begin{aligned} a_i &= 1, && 1 \leqslant i \leqslant j-3 \\ a_i &= \frac{t'-t_i}{t_{i+3}-t_i}, && j-2 \leqslant i \leqslant j \quad \text{（被0除是0）} \\ a_i &= 0, && j+1 \leqslant i \leqslant n \end{aligned} \tag{11-55}$$

509

这个算法是更一般的Oslo算法[COHE80]的一种特殊的情况，Oslo算法可以在B样条曲线的一次计算过程中插入任意多个结点。若一次插入多个结点，Oslo算法要比Böhm算法更有效。

举一个Böhm分割的例子，结点序列（0, 0, 0, 0, 1, 1, 1, 1），四个控制点的x坐标为（5.0, 8.0, 9.0, 6.0），考虑新结点$t = 0.5$（设$n = 3$，$j = 3$）。由式(11-55)定义的a_i值为（0.5, 0.5, 0.5）。代入式(11-54)得到新控制点Q_i的x坐标（5.0, 6.5, 8.5, 7.5, 6.0）。

注意，增加一个结点，原来两个控制点就会被三个新控制点代替。而且，曲线段分割后，

它的相邻曲线段只由新控制点定义。相比之下，均匀B样条分割时五个新控制点替换四个原控制点，同时相邻段仍由原来的控制点定义，两个新曲线段已不使用这些旧控制点了，所以，非均匀B样条要好用得多。

*层次式B样条细分*是另一种对曲线形状更有效的控制方法[FORS89]。在曲线局部区域加入额外的控制点，建立一个层次式数据结构来定义新控制点与原控制点的关系。层次的应用可以实现更多的加细。这种方法不是用更多的新控制点代替原控制点，而是将层次存储起来，这样既可以用原控制点定义粗略的曲线形状，也可以用新控制点进行细节的控制。

11.2.8 各种表示法之间的转换

我们经常要把一种表示转化为另一种表示。也就是说，已知一条由几何矩阵G_1和基矩阵M_1表示的曲线，要求出对应于另一个基矩阵M_2的等价几何矩阵G_2，使两条曲线相同：$T \cdot M_2 \cdot G_2 = T \cdot M_1 \cdot G_1$。上式还可以写为$M_2 \cdot G_2 = M_1 \cdot G_1$。解出未知几何矩阵$G_2$，得到

$$M_2^{-1} \cdot M_2 \cdot G_2 = M_2^{-1} \cdot M_1 \cdot G_1, \text{ or } G_2 = M_2^{-1} \cdot M_1 \cdot G_1 = M_{1,2} \cdot G_1 \tag{11-56}$$

也就是说，将对应于表示1的已知几何向量转化为对应于表示2的几何向量的矩阵就是$M_{1,2} = M_2^{-1} \cdot M_1$。

举个例子，将B样条曲线转化成Bézier形式，矩阵$M_{Bs,B}$为

$$M_{Bs,B} = M_B^{-1} \cdot M_{Bs} = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{bmatrix} \tag{11-57}$$

它的逆矩阵为

$$M_{B,Bs} = M_{Bs}^{-1} \cdot M_B = \begin{bmatrix} 6 & -7 & 2 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 2 & -7 & 6 \end{bmatrix} \tag{11-58}$$

510

非均匀B样条没有一个确定的基矩阵。注意到由四个控制点且结点序列为（0，0，0，0，1，1，1，1）的非均匀B样条曲线就是Bézier曲线。因此，将非均匀B样条转化为其他任一种形式的一个办法是，用11.2.7节提到的Böhm算法或Oslo算法用插入重结点法把所有的结点重数变为4，使其先转化为Bézier曲线，然后再把Bézier形式转化为其他任意一种有基矩阵的形式。

11.2.9 曲线绘制

画三次参数曲线有两种基本方法。第一种是对不断增加的t值迭代求出$x(t)$，$y(t)$和$z(t)$，然后把相邻的点用直线段相连，第二种是对曲线进行递归分割直到控制顶点足够接近曲线本身。

在11.2节的引言中曾经介绍过一种简单的重复求值显示法，用Horner规则对连续的t值计算出多项式的值。每个三维点的开销是9个乘法和10个加法。但是，用*向前差分*方法重复计算三次多项式，则要有效得多。函数$f(t)$的向前差分$\Delta f(t)$是

$$\Delta f(t) = f(t + \delta) - f(t), \delta > 0 \tag{11-59}$$

即

$$f(t + \delta) = f(t) + \Delta f(t) \tag{11-60}$$

将式(11-60)写成迭代形式，得到

$$f_{n+1} = f_n + \Delta f_n \tag{11-61}$$

其中f以步长为δ作等间隔计算，使得$t_n = n\delta$和$f_n = f(t_n)$。

对于三次多项式

$$f(t) = at^3 + bt^2 + ct + d = T \cdot C \tag{11-62}$$

向前差分为

$$\begin{aligned}\Delta f(t) &= a(t+\delta)^3 + b(t+\delta)^2 + c(t+\delta) + d - (at^3 + bt^2 + ct + d) \\ &= 3at^2\delta + t(3a\delta^2 + 2b\delta) + a\delta^3 + b\delta^2 + c\delta\end{aligned} \tag{11-63}$$

这样，$\Delta f(t)$是一个二次多项式。但是，计算式(11-61)仍然需要先计算出$\Delta f(t)$，外加一次加法。不过，可以用向前差分来简化$\Delta f(t)$的计算。从式(11-61)，可得

$$\Delta^2 f(t) = \Delta(\Delta f(t)) = \Delta f(t+\delta) - \Delta f(t) \tag{11-64}$$

代入式(11-63)可得

$$\Delta^2 f(t) = 6a\delta^2 t + 6a\delta^3 + 2b\delta^2 \tag{11-65}$$

这是一个关于t的一次式。用下标n改写式(11-64)，得到

$$\Delta^2 f_n = \Delta f_{n+1} - \Delta f_n \tag{11-66}$$

511

将n换为$n-1$并整理得到

$$\Delta f_n = \Delta f_{n-1} + \Delta^2 f_{n-1} \tag{11-67}$$

现在，为求出式(11-61)中的Δf_n，只要求出$\Delta^2 f_{n-1}$再加上Δf_{n-1}即可。因为$\Delta^2 f_{n-1}$关于t是线性的，这要比直接从二次多项式(11-63)计算Δf_n简单得多。

重复上述步骤，以避免直接从式(11-65)求$\Delta^2 f(t)$:

$$\Delta^3 f(t) = \Delta(\Delta^2 f(t)) = \Delta^2 f(t+\delta) - \Delta^2 f(t) = 6a\delta^3 \tag{11-68}$$

三阶向前差分是常数，所以不必再做向前差分了。考虑指标n以及$\Delta^3 f_n$为常数，把式(11-68)写成

$$\Delta^2 f_{n+1} = \Delta^2 f_n + \Delta^3 f_n = \Delta^2 f_n + 6a\delta^3 \tag{11-69}$$

再将n换为$n-2$，得

$$\Delta^2 f_{n-1} = \Delta^2 f_{n-2} + 6a\delta^3 \tag{11-70}$$

上述结果代入式(11-67)求出Δf_n，然后再将Δf_n代入式(11-61)求出f_{n+1}。

为了从$n=0$到$n\delta=1$迭代求出向前差分，先用式(11-62)、(11-63)、(11-65)和(11-68)计算出$t=0$时的初始条件。它们是

$$f_0 = d,\ \Delta f_0 = a\delta^3 + b\delta^2 + c\delta,\ \Delta^2 f_0 = 6a\delta^3 + 2b\delta^2,\ \Delta^3 f_0 = 6a\delta^3 \tag{11-71}$$

直接对四个等式求值得到这些初始条件。但若令初始差分向量为D，则

$$D = \begin{bmatrix} f_0 \\ \Delta f_0 \\ \Delta^2 f_0 \\ \Delta^3 f_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ \delta^3 & \delta^2 & \delta & 0 \\ 6\delta^3 & 2\delta^2 & 0 & 0 \\ 6\delta^3 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{11-72}$$

用$E(\delta)$表示其中的4×4矩阵，上式可写成$D=E(\delta)C$。因为要处理三个函数$x(t)$, $y(t)$和$z(t)$，所以有三个相应的初始条件，$D_x = E(\delta)C_x$, $D_y = E(\delta)C_y$, $D_z = E(\delta)C_z$。

基于以上的推导，图11-36给出了一个显示三次参数曲线的算法。这个过程对每一个三维点只需要九个加法而无须乘法，而且初始化时也只需要很少的几个加法和乘法!这比起用Horner规则实现的需要10个加法9个乘法的简单迭代求值要好多了。但是，要注意的是这个算法会有一个误差累积问题，因此必须留有足够的小数精度来避免这一问题。比如，当$n=64$并使用整型算法时，需要16位的小数精度；当$n=256$时，则需要22位[BART87]。

```
void DrawCurveFwdDif (
        int n,                          /* 用于画曲线的步数 */
        double x, double Δx, double Δ²x, double Δ³x,
        /* x(t)多项式在t = 0时的初值，计算为D_x = E(δ)C_x */
        double y, double Δy, double Δ²y, double Δ³y,
        /* y(t)多项式在t = 0时的初值，计算为Dy = E(δ)C_y */
        double z, double Δz, double Δ²z, double Δ³z
        /* z(t)多项式在t = 0时的初值，计算为D_z = E(δ)C_z */
        /* 用于计算D_x, D_y和D_z的步长δ为1/n */
)

{
    int i;

    MoveAbs3 (x, y, z);         /* 移到曲线起点 */
    for (i = 0; i < n; i++) {
        x += Δx;   Δx += Δ²x;   Δ²x += Δ³x;
        y += Δy;   Δy += Δ²y;   Δ²y += Δ³y;
        z += Δz;   Δz += Δ²z;   Δ²z += Δ³z;
        LineAbs3 (x, y, z);     /* 画一条短线段 */
    }
} /*  DrawCurveFwdDif */
```

图11-36 用向前差分显示三次参数曲线的程序

显示曲线的第二种方法是递归分割。递归终止条件是自适应的，当分割后的曲线段已足够平坦而可以用直线段逼近时，就可以终止了。其中的细节依不同类型的曲线而不同，因为每一种曲线的分割过程和平坦程度的判断都会有细微的差别。图11-37给出了一个一般的算法。

```
void DrawCurveRecSub (curve, ε)
{
    if (Straight (curve, ε))      /* 测试控制点在线的ε内 */
        DrawLine (curve);
    else {
        SubdivideCurve (curve, leftCurve, rightCurve);
        DrawCurveRecSub (leftCurve, ε);
        DrawCurveRecSub (rightCurve, ε);
    }
} /* DrawCurveRecSub */
```

图11-37 用递归分割显示曲线的程序。函数Straight当曲线足够平坦时返回true

Bézier表示尤其适合使用递归分割进行显示。它的分割很快，只需要式(11-51)中的六个移位和六个加法。Bézier曲线段的平坦程度判断也比较简单，只要基于四个控制点构成的凸包即可，见图11-38。若距离d_2和d_3中的大者小于某个阈值ε，则曲线段可以用连接两端点的直线段来近似代替。Bézier曲线的端点就是第一个和最后一个控制点P_1与P_4。如果使用其他类

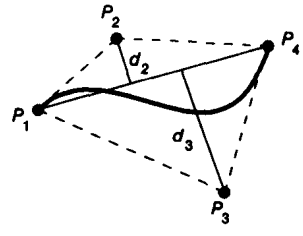


图11-38 曲线段平坦程度的判断。若d_2和d_3都小于ε，则认为曲线段是足够平坦的，可以用线段P_1P_4来近似代替

型的曲线，平坦测试会复杂得多，而且还需要求出端点。因此，用递归分割显示前常常先将曲线转化Bézier形式。

关于递归分割的更多细节请参阅[BARS85；BARS87；LANE80]。如果用整型算法来实现递归分割，所需要的小数精度要比用向前差分方法的小。若最多要8层递归（一个合乎情理的预计），则只需要3位的小数精度就够了；若16层，只需4位。

递归分割避免了许多不必要的计算，而向前差分法用的是固定步长的分割。向前差分的步长δ必须足够小，使曲线中曲率半径最小的部分也能达到比较好的逼近效果。而对曲线中很直的部分，用一个比较大的步长就行了。[LANE80a]给出了一种由给定分割曲线的最大偏差来计算出分割步长δ的方法。另一方面，递归分割需要花时间进行曲线的平坦程度判断。一种替代的方法是递归分割到一个固定的深度，即多做一些分割步骤而不用平坦测试（见习题11.30）。

一种混合的方法，自适应向前差分法[LIEN87；SHAN87；SHAN89]，综合了向前差分法和递归分割法的优点。基本策略是向前差分，但使用的是自适应步长。采用有效的计算方法将步长加倍或减半，以保证逼近误差总接近于1个像素。也就是说实质上在计算点之间不用直线段逼近。

11.2.10 三次曲线的比较

不同类型的三次参数曲线可以从几个不同方面进行比较，比如交互控制的灵活性、在拼接点处的连续程度、表示的一般性和计算速度等。当然，既然所有表示都可以相互转化(在11.2.8节曾讨论过)，没有必要只选择单独一种表示法。举个例子，可以用非均匀有理B样条作为内部表示，而用户可以交互控制Bézier控制点或者Hermite控制点和切向量。一些交互的图形编辑系统为用户提供Hermite曲线，同时在内部表示时为PostScript支持的Bézier形式[ADOB85a]。一般来说，一个交互CAD系统的用户可能有几种选择，如Hermite、Bézier、均匀B样条和非均匀B样条等。因为非均匀有理B样条最具一般性，它经常用于系统的内部表示。

512 ~ 514

表11-2对本节提到过的大多数曲线进行了比较。表中并没有直接给出交互控制灵活性的比较，因为它更依赖于具体的应用领域。“控制曲线段的参数数量”是指四个几何条件以及其他参数，比如非均匀样条曲线的结点分布，β样条的β_1和β_2，或者Kochanek-Bartels模型中的a，b，c。“容易达到的连续性”指一些约束条件，比如使控制点共线以达到G^1连续性等。因为C^n连续性要比G^n强，任何一种能达到C^n连续的曲线理论上也能达到G^n。

表11-2 七种三次参数曲线的比较

	Hermite	Bézier	均匀B样条	形状均一β样条	非均匀B样条	Catmull-Rom	Kochanek-Bartels
控制点定义的凸包	N/A	Yes	Yes	Yes	Yes	No	No
插值一些控制点	Yes	Yes	No	No	No	Yes	Yes
插值所有控制点	Yes	No	No	No	No	Yes	Yes
易于分割	Cood	Best	Avg	Avg	High	Avg	Avg
表示法固有的连续性	C^0	C^0	C^2	C^0	C^2	C^1	C^1
	G^0	G^0	G^2	G^2	G^2	G^1	G^1
容易达到的连续性	C^1	C^1	C^2	C^1	C^2	C^1	C^1
	G^1	G^1	$G^2$①	$G^2$①	$G^2$①	G^1	G^1
控制一曲线段的参数个数	4	4	4	6②	5	4	7

① 除了在11.2节讨论的特殊情况外。

② 其中4个参数针对每个段，是局部参数，2个参数是整个曲线的全局参数。

CAD系统经常只要求几何连续性，此时，选择范围就缩小到几种样条曲线上，它们都能达到G^1和G^2连续。在表中的三种样条曲线中，均匀B样条曲线限制最多。非均匀B样条所允许的多重结点，使得用户对曲线形状的控制更灵活。同样，有β样条曲线中使用的形状参数β_1和β_2。当然，重要的是要有一个好的用户界面，使用户能充分利用这些功能。

若要使曲线插值照相机路径或形状轮廓线上的数据点，Catmull-Rom样条曲线或Kochanek-Bartels样条曲线十分合适。当生成曲线既需要插值又需要切向量控制时，Bézier曲线或Hermite曲线最合适。

传统上用户应该能交互地拖动控制点或切向量，并及时更新曲线显示。图11-23显示了这样一个B样条序列。在一些应用中，B样条曲线的一个缺点是控制点不在曲线上。但也可以不显示控制点，而让用户交互控制结点（做上标记以便选取）。当用户选取一个结点并将它移动了$(\Delta x, \Delta y)$时，受结点影响最大并确定拼接点位置的控制点也移动了$(\Delta x, \Delta y)$。但拼接点并没有移动$(\Delta x, \Delta y)$，因为它是好几个控制点的加权和，而只移动了其中一个控制点。因此，把鼠标重新定位在拼接点上。这个过程不断迭代直到用户停止拖动。

11.3 双三次参数曲面

双三次参数曲面是三次参数曲线的推广。回忆一下三次参数曲线的一般形式$Q(t) = T \cdot M \cdot G$，其中几何向量G是一个常量。首先，为了定义方便，我们将t换成s，得到$Q(s) = S \cdot M \cdot G$。如果允许G中的点可以在三维空间中沿某条以t为参数的路径变化，则得到

$$Q(s, t) = S \cdot M \cdot G(t) = S \cdot M \cdot \begin{bmatrix} G_1(t) \\ G_2(t) \\ G_3(t) \\ G_4(t) \end{bmatrix} \tag{11-73}$$

现在，对一个固定的t_1，$Q(s, t_1)$是一条曲线，因为$G(t_1)$是常量。让t变到另一个值，比如，t_2，其中$t_2 - t_1$很小，则$Q(s, t_2)$略有不同。令t_2在0和1之间重复取值，就定义了整个曲线网格，相邻两条曲线都可任意接近。所有这些曲线的集合就定义了一个曲面。若$G_i(t)$本身是三次曲线，则这样的曲面称为*双三次参数曲面*。

515
~
516

考虑$G_i(t)$为三次曲线的情形，每一个都可以表示为$G_i(t) = T \cdot M \cdot \boldsymbol{G}_i$，其中$\boldsymbol{G}_i = [g_{i1} g_{i2} g_{i3} g_{i4}]^T$（$G$和$g$用来区别曲线中用到的$G$）。因此，$g_{i1}$就是曲线$G_i(t)$的几何向量中的第一个元素，以此类推。

利用恒等式$(A \cdot B \cdot C)^T = C^T \cdot B^T \cdot A^T$，对式$G_i(t) = T \cdot M \cdot \boldsymbol{G}_i$做转置，得到$G_i(t) = \boldsymbol{G}_i^T \cdot M^T \cdot T^T = [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}] \cdot M^T \cdot T^T$。将这个结果代入式(11-73)，得到

$$Q(s, t) = S \cdot M \cdot \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \cdot M^T \cdot T^T \tag{11-74}$$

或者

$$Q(s, t) = S \cdot M \cdot \boldsymbol{G} \cdot M^T \cdot T^T, 0 \leqslant s, t \leqslant 1 \tag{11-75}$$

把x，y，z分开表示，则形式为

$$
\begin{aligned}
x(s,t) &= S \cdot M \cdot G_x \cdot M^{\mathrm{T}} \cdot T^{\mathrm{T}} \\
y(s,t) &= S \cdot M \cdot G_y \cdot M^{\mathrm{T}} \cdot T^{\mathrm{T}} \\
z(s,t) &= S \cdot M \cdot G_z \cdot M^{\mathrm{T}} \cdot T^{\mathrm{T}}
\end{aligned}
\tag{11-76}
$$

给定这样的一般形式，现在可以继续探讨用不同的几何矩阵描述特定曲面的特定方法了。

11.3.1 Hermite曲面

Hermite曲面完全由一个4×4的几何矩阵G_H定义。使用与式(11-75)同样的推导方法，可以求出G_H。这里，再把推导过程深入展开一些，代入$x(s,t)$。首先，把式(11-13)中的t换成s，得到$x(s) = S \cdot M_H \cdot G_{Hx}$。重新改写这个等式，使$G_{Hx}$不是常量，而是关于$t$的函数，得到

$$
x(s,t) = S \cdot M_H \cdot G_{H_x}(t) = S \cdot M_H \begin{bmatrix} P_1(t) \\ P_4(t) \\ R_1(t) \\ R_4(t) \end{bmatrix}_x
\tag{11-77}
$$

函数$P_{1_x}(t)$和$P_{4_x}(t)$定义了关于s的参数曲线的初始点和终止点的x分量。类似地，$R_{1_x}(t)$和$R_{4_x}(t)$是这两点上的切向量。对于任一给定的t值，可随之确定两端点和两切向量。图11-39给出了$P_1(t)$，$P_4(t)$以及当t等于0.0，0.2，0.4，0.6，0.8和1.0时关于s的三次曲线。实际上也可以把这块曲面看成$P_1(t) = Q(0,t)$和$P_4(t) = Q(1,t)$之间的三次插值，或者是$Q(s,0)$和$Q(s,1)$之间的三次插值。

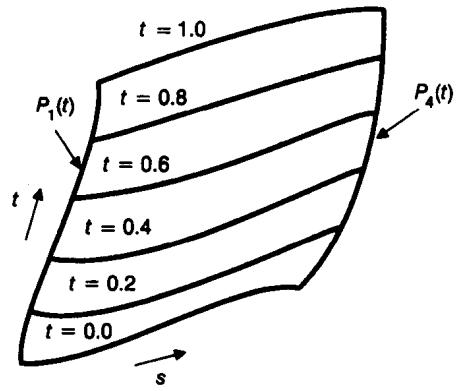


图11-39 双三次曲面上的等参线：$P_1(t)$取$s=0$，$P_4(t)$取$s=1$

特殊情况是，四条插值曲线$Q(0,t)$，$Q(1,t)$，$Q(s,0)$和$Q(s,1)$都是直线，得到的是一个直纹面。若插值曲线共面，则曲面就成了一个平面四边形。

继续推导，把$P_1(t)$，$P_4(t)$，$R_1(t)$和$R_4(t)$都表示为Hermite形式

$$
\begin{aligned}
P_{1_x}(t) &= T \cdot M_H \begin{bmatrix} g_{11} \\ g_{12} \\ g_{13} \\ g_{14} \end{bmatrix}_x, \qquad P_{4_x}(t) = T \cdot M_H \begin{bmatrix} g_{21} \\ g_{22} \\ g_{23} \\ g_{24} \end{bmatrix}_x \\
R_{1_x}(t) &= T \cdot M_H \begin{bmatrix} g_{31} \\ g_{32} \\ g_{33} \\ g_{34} \end{bmatrix}_x, \qquad R_{4_x}(t) = T \cdot M_H \begin{bmatrix} g_{41} \\ g_{42} \\ g_{43} \\ g_{44} \end{bmatrix}_x
\end{aligned}
\tag{11-78}
$$

这四条三次曲线可以合为一个等式：

$$
[P_1(t) \quad P_4(t) \quad R_1(t) \quad R_4(t)]_x = T \cdot M_H \cdot G_{H_x}^{\mathrm{T}}
\tag{11-79}
$$

其中

$$\boldsymbol{G}_{\mathrm{H}_x}=\begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}_x \tag{11-80}$$

将等式(11-79)两边一起作转置，得

$$\begin{bmatrix} P_1(t) \\ P_4(t) \\ R_1(t) \\ R_4(t) \end{bmatrix}_x = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}_x \boldsymbol{M}_{\mathrm{H}}^{\mathrm{T}} \cdot \boldsymbol{T}^{\mathrm{T}} = \boldsymbol{G}_{\mathrm{H}_x} \cdot \boldsymbol{M}_{\mathrm{H}}^{\mathrm{T}} \cdot \boldsymbol{T}^{\mathrm{T}} \tag{11-81}$$

517
↓
518

将等式(11-81)代入等式(11-77)得

$$x(s,t) = \boldsymbol{S} \cdot \boldsymbol{M}_{\mathrm{H}} \cdot \boldsymbol{G}_{\mathrm{H}_x} \cdot \boldsymbol{M}_{\mathrm{H}}^{\mathrm{T}} \cdot \boldsymbol{T}^{\mathrm{T}} \tag{11-82}$$

类似地，

$$y(s,t) = \boldsymbol{S} \cdot \boldsymbol{M}_{\mathrm{H}} \cdot \boldsymbol{G}_{\mathrm{H}_y} \cdot \boldsymbol{M}_{\mathrm{H}}^{\mathrm{T}} \cdot \boldsymbol{T}^{\mathrm{T}},\ z(s,t) = \boldsymbol{S} \cdot \boldsymbol{M}_{\mathrm{H}} \cdot \boldsymbol{G}_{\mathrm{H}_z} \cdot \boldsymbol{M}_{\mathrm{H}}^{\mathrm{T}} \cdot \boldsymbol{T}^{\mathrm{T}} \tag{11-83}$$

三个4×4矩阵$\boldsymbol{G}_{\mathrm{H}_x}$，$\boldsymbol{G}_{\mathrm{H}_y}$和$\boldsymbol{G}_{\mathrm{H}_z}$在Hermite曲面中的作用与$\boldsymbol{G}_{\mathrm{H}}$在Hermite曲线中的作用一样。通过式(11-77)和式(11-78)可以理解$\boldsymbol{G}_{\mathrm{H}}$的16个元素的含义，元素$g_{11_x}$是$x(0,0)$，因为它是$P_{1_x}(t)$的起始点，也就是$x(s,0)$的起始点。类似地，$g_{12_x}$是$P_{1_x}(t)$的终点，也就是$x(s,1)$的起始点，从而是$x(0,1)$。$g_{13_x}$是$\alpha x/\alpha t(0,0)$，因为它是$P_{1_x}(t)$的起始点上的切向量；$g_{33_x}$是$\alpha^2 x/\alpha s \alpha t(0,0)$，因为它是$R_{1_x}(t)$的起始点上的切向量，也就是$x(s,0)$的起始点的斜率。

采用这些表示，可以把$\boldsymbol{G}_{\mathrm{H}_x}$写成

$$\boldsymbol{G}_{H_x}=\begin{bmatrix} x(0,0) & x(0,1) & \dfrac{\partial}{\partial t}x(0,0) & \dfrac{\partial}{\partial t}x(0,1) \\ x(1,0) & x(1,1) & \dfrac{\partial}{\partial t}x(1,0) & \dfrac{\partial}{\partial t}x(1,1) \\ \dfrac{\partial}{\partial s}x(0,0) & \dfrac{\partial}{\partial s}x(0,1) & \dfrac{\partial^2}{\partial s \partial t}x(0,0) & \dfrac{\partial^2}{\partial s \partial t}x(0,1) \\ \dfrac{\partial}{\partial s}x(1,0) & \dfrac{\partial}{\partial s}x(1,1) & \dfrac{\partial^2}{\partial s \partial t}x(1,0) & \dfrac{\partial^2}{\partial s \partial t}x(1,1) \end{bmatrix} \tag{11-84}$$

$\boldsymbol{G}_{\mathrm{H}_x}$的左上角的2×2块包含了曲面片四个角的$x$坐标。右上角和左下角的2×2块分别是曲面片沿两个参数方向的切向量。右下角的2×2块是曲面四个角点处关于s和t的偏导数。这些偏导数通常称为扭矢，因为它们越大，曲面在四角处卷曲得越厉害。图11-40中的曲面片的四个角点都标出了这些参数。

这种双三次曲面片的Hermite形式是Coons曲面片[COON67]的另一种严格表达形式。更一般的曲面可以允许边界曲线和斜率是任意曲线。(Coons曲面片是由Steven A.Coons后期提出来的[HERZ80]，他是CAD和计算机图形学的先驱，SIGGRAPH以他的名字命名了计算机图形学突出贡献Steven A.Coons奖。)当Hermite曲面的四个扭矢都为零时，曲面也称为Ferguson曲面，这是以另一位早期曲面表示的研究者的名字命名的[FERG64；FAUX79]。

正如三次Hermite曲线在两个曲线段之间具有C^1和G^1连续性一样，Hermite双三次曲面在两个曲面片之间同样具有C^1和G^1连续性。首先，要在边界处C^0连续，两块曲面的边界上的控制点

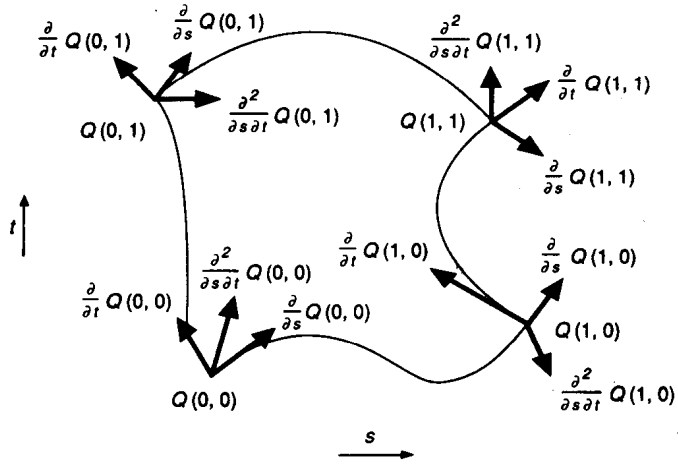


图11-40 Hermite曲面的几何矩阵的各分量。每个向量都是三元组，式(11-84)求出的是x分量

必须重合。C^1连续性的必要条件是两块曲面上的控制点沿边界线相等，曲面沿边界线的跨界切向量和扭矢都分别相等。对于G^1连续性，切向量这一条件可以放宽为在同一方向上，而不一定要有相同的大小。若公共边在面片1是$s=1$，在面片2是$s=0$，如图11-41所示，则两曲面的几何矩阵中某几行的值必须体现G^1连续条件，如下所示：

曲面片1　　曲面片2

$$\begin{bmatrix} - & - & - & - \\ g_{21} & g_{22} & g_{23} & g_{24} \\ - & - & - & - \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \quad \begin{bmatrix} g_{21} & g_{22} & g_{23} & g_{24} \\ - & - & - & - \\ kg_{41} & kg_{42} & kg_{43} & kg_{44} \\ - & - & - & - \end{bmatrix} \quad k>0 \tag{11-85}$$

标为短线的项可以是任意值。如果四个曲面片在同一个公共角点相拼接，且曲面片之间沿这个角点出发的边界要具有G^1连续性，则曲面片之间的关系要更复杂，见习题11.25。

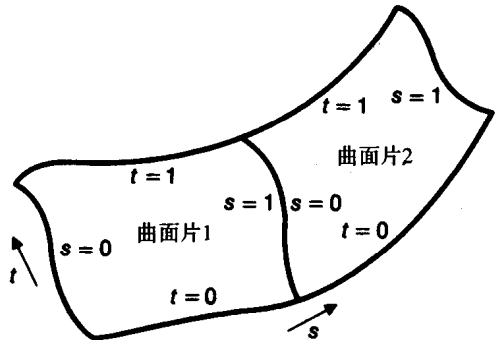


图11-41 两块相接曲面

11.3.2 Bézier曲面

Bézier双三次曲面的公式推导过程与Hermite三次曲线的完全一样。其结果是：

$$\begin{aligned} x(s,t) &= S\cdot M_B\cdot G_{B_x}\cdot M_B^T\cdot T^T \\ y(s,t) &= S\cdot M_B\cdot G_{B_y}\cdot M_B^T\cdot T^T \\ z(s,t) &= S\cdot M_B\cdot G_{B_z}\cdot M_B^T\cdot T^T \end{aligned} \tag{11-86}$$

Bézier几何矩阵G由16个控制点构成，如图11-42所示。和Bézier曲线一样，Bézier曲面在交互设计时十分常用。一部分控制点落在曲面上，提供了精确的控制，同时还可以直接控制切向量。当Bézier曲面用于内部表示时，常用到它的凸包性和易分割性。

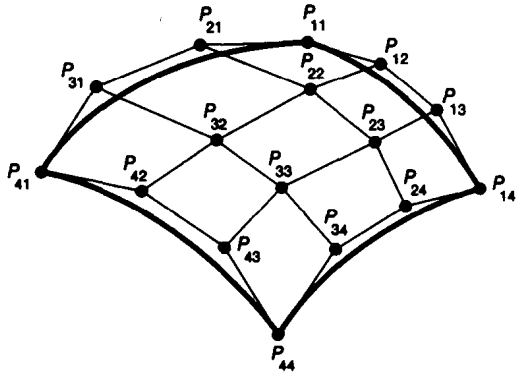


图11-42 Bézier双三次曲面片的十六个控制点

使四个公共控制点重合就能保证曲面片拼接边的C^0和G^0连续性。当拼接边两侧相对应的四个控制点与边界上的四个控制点的连线共线时，曲面具有G^1连续性。在图11-43中，以下几组控制点连线平共线，而且定义了4条线段，其长度有相同比率k：(P_{13}，P_{14}，P_{15})，(P_{23}，P_{24}，P_{25})，(P_{33}，P_{34}，P_{35})和(P_{43}，P_{44}，P_{45})。

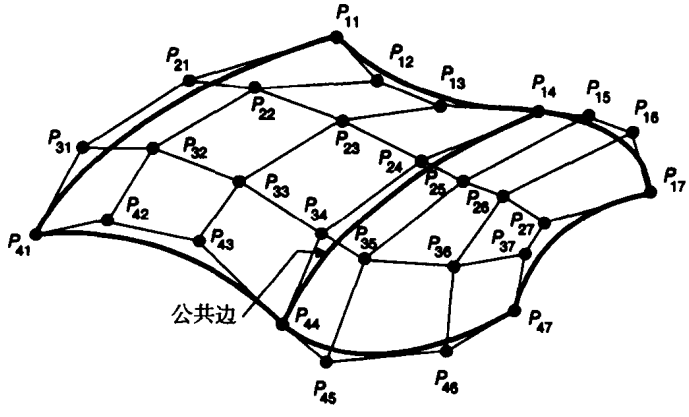


图11-43 两块在边$P_{14}, P_{24}, P_{34}, P_{44}$相拼接的Bézier曲面片

还有一种保持曲面间连续性的方法，在[FAUX79；BEZI70]中有详细论述。这种方法要求拼接边的两角点、控制点与相邻的控制点共面。

11.3.3 B样条曲面

B样条曲面片表示为

$$\begin{aligned} x(s,t) &= S \cdot M_{Bs} \cdot G_{Bs_x} \cdot M_{Bs}{}^{T} \cdot T^{T} \\ y(s,t) &= S \cdot M_{Bs} \cdot G_{Bs_y} \cdot M_{Bs}{}^{T} \cdot T^{T} \\ z(s,t) &= S \cdot M_{Bs} \cdot G_{Bs_z} \cdot M_{Bs}{}^{T} \cdot T^{T} \end{aligned} \tag{11-87}$$

B样条曲面在边界处自动具有C^2连续性；对控制点没有什么特殊要求，只是要避免重控制点。因为重控制点会造成不连续。

双三次非均匀有理B样条曲面以及其他有理曲面与它们的三次曲线形式类似。相应的分割和显示方法都可以直接用在双三次曲面中。

11.3.4 曲面的法线

在曲面明暗处理（第16章）、机器人碰撞检测、数控加工中的等距线计算以及其他一些计算中都需要用到双三次曲面的法线，求双三次曲面的法线较为容易。由式(11-75)可得曲面$Q(s, t)$的s切向量

$$\frac{\partial}{\partial s}Q(s,t)=\frac{\partial}{\partial s}(S\cdot M\cdot G\cdot M^{T}\cdot T^{T})=\frac{\partial}{\partial s}(S)\cdot M\cdot G\cdot M^{T}\cdot T^{T}$$
$$=[3s^2\quad 2s\quad 1\quad 0]\cdot M\cdot G\cdot M^{T}\cdot T^{T} \tag{11-88}$$

519 ~ 522

以及t的切向量

$$\frac{\partial}{\partial t}Q(s,t)=\frac{\partial}{\partial t}(S\cdot M\cdot G\cdot M^{T}\cdot T^{T})=S\cdot M\cdot G\cdot M^{T}\cdot\frac{\partial}{\partial t}(T^{T})$$
$$=S\cdot M\cdot G\cdot M^{T}\cdot[3t^2\quad 2t\quad 1\quad 0]^{T} \tag{11-89}$$

这两个切向量在点(s,t)处与曲面平行，因此，它们的叉乘垂直于曲面。当两个向量都为零时，叉乘积也为零，这时平面法线没有意义。我们曾经说过当切向量为零时拼接点处C^1连续但不是G^1连续。

因为式(11-75)表示双三次曲面的x，y和z分量，所以每个切向量都是三元组。用x_s表示s切向量的x分量，y_s表示y分量，z_s表示z分量，则法线为

$$\frac{\partial}{\partial s}Q(s,t)\times\frac{\partial}{\partial t}Q(s,t)=[y_sz_t-y_tz_s\quad z_sx_t-z_tx_s\quad x_sy_t-x_ty_s] \tag{11-90}$$

曲面的法线是一个双五次（二元，五次）多项式，因此计算起来十分麻烦。[SCHW82]中给出了一种当曲面本身相对平滑时较为满意的双三次逼近法。

11.3.5 双三次曲面的显示

与曲线类似，曲面也可以用双三次多项式迭代求值或者分割两种方法显示，其中分割方法实质上是一种双三次多项式的自适应求值。首先考察迭代求值法，然后讨论分割法。

迭代求值法最适合显示图11-44的那种双三次曲面片类型。曲面上关于参数s和t的每一条等参数曲线都是三次曲线，于是可以直接显示这些曲线，如图11-45所示。

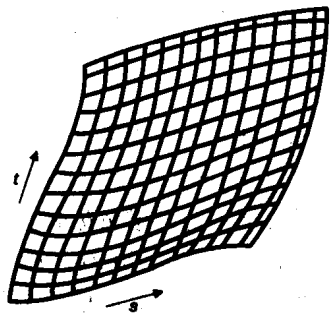


图11-44 用s和t的等参数曲线显示曲面片 523

```
typedef double Coeffs[4][4][3];
void DrawSurface (
        Coeffs coefficients,        /* Q(s,t)的系数 */
        int n_s,                    /* s 为常数的等参数曲线的曲线次数，一般取5~10 */
        int n_t,                    /* t 为常数的等参数曲线的曲线次数，一般取5~10 */
        int n)                      /* 每条曲线的画线步数，一般取20~100 */
{
    double δ = 1.0 / n;             /* 画每条曲线时使用的步长 */
    double δ_s = 1.0 / (n_s-1);     /* t 为常数的等参数曲线，在s方向上的步长的增量 */
    double δ_t = 1.0 / (n_t-1);     /* s 为常数的等参数曲线，在t方向上的步长的增量 */
    int i, j; double s, t;
    /* 画s为常数的等参数曲线n_s，  s = 0, δ_s, 2δ_s,…1 */
    for (i = 0, s = 0.0; i < n_s; i++, s += δ_s) {
        /* 画s为常数的等参数曲线，t 取值为0~1 */
        /* X, Y, Z为双三次曲线的求值函数 */
        MoveAbs3 (X (s, 0.0), Y (s, 0.0), Z (s, 0.0));
```

图11-45 双三次曲面片的网格显示过程。X(s, t), Y(s, t)和Z(s, t)过程分别用系数矩阵*coefficients*计算曲面

```
        for (j = 1, t = δ; j < n; j++, t += δ) {
            /* 对于每条曲线，t从δ到1之间，取n-1步 */
            LineAbs3 (X (s, t), Y (s, t), Z (s, t));
        }
    }
    /* 画t为常数的等参数曲线n_t，t = 0, δ_t, 2δ_t, ···1 */
    for (i = 0, t = 0.0; i < n_t; i++, t += δ_t) {
        /* 画t为常数的等参数曲线 s 取值为0~1 */
        MoveAbs3 (X (0.0, t), Y (0.0, t), Z (0.0, t));
        for (j = 1, s = δ; j < n; j++, s += δ) {
            /* 对于每条曲线，s从δ到1之间，取n-1步 */
            LineAbs3 (X (s, t), Y (s, t), Z (s, t));
        }
    }
} /* DrawSurface */
```

图11-45（续）

对曲面直接迭代求值通常比曲线情形开销大，因为曲面方程大约要计算$2/\delta^2$次。当$\delta = 0.1$时，次数为200；当$\delta = 0.01$时，次数为20 000。从这些数字可以看出，这里用向前差分法更合适。11.2.9节给出了基本向前差分法。剩下的步骤是要理解怎样从曲线i的向前差分值求出曲线$i+1$的初始向前差分值。

11.2.9节中的用于寻找曲线的$D = E(\delta) \cdot C$的推导，同样地可以用于寻找

$$DD_x = E(\delta_s) \cdot A_x \cdot E(\delta_t)^T \tag{11-91}$$

524 其中δ_s是关于s的步长，δ_t是关于t的步长，A_x是$x(s, t)$的4×4系数矩阵。4×4矩阵DD_x的第一行的值为$x(0, 0)$，$\Delta_t x(0, 0)$，$\Delta_t^2 x(0, 0)$和$\Delta_t^3 x(0, 0)$（记号Δ_t表示关于t的向前差分，对参数s也类似）。这样，就可以用第一行的增量δ_t求出$x(0, t)$。

求出$x(0, t)$后，怎样求出$x(\delta_s, t)$以便画出关于参数s的等参数曲线呢？（这一步是从曲线i的向前差分求曲线$i+1$的初始向前差分。）DD_x中的其他行分别是第一行的向前差分的关于s的一阶、二阶、三阶向前差分。因此，可以将下列等式应用于DD_x的行

$$\begin{aligned} &\text{row 1} := \text{row 1} + \text{row 2} \\ &\text{row 2} := \text{row 2} + \text{row 3} \\ &\text{row 3} := \text{row 3} + \text{row 4} \end{aligned} \tag{11-92}$$

计算出DD_x的第一行$x(\delta_s, 0)$，$\Delta_t x(\delta_s, 0)$，$\Delta_t^2 x(\delta_s, 0)$和$\Delta_t^3 x(\delta_s, 0)$。与前面一样，用这些值和向前差分法可以求出$x(\delta_s, t)$。然后重复等式(11-92)中的步骤，用$D$的第一行求出$x(2\delta_s, t)$，以此类推。

关于t的等参数曲线可以类似画出。为了计算$x(s, 0)$，只要利用式(11-92)，把DD_x中的行换成列。也可以用DD_x^T代替DD_x，这样仍然可用矩阵的行来计算$x(s, 0)$，$x(s, \delta_t)$，$x(s, 2\delta_t)$，等等。

图11-46给出了一个利用向前差分法用s和t的等参数曲线显示曲面的算法。

用递归分割法显示曲面也是11.2.9节中程序DrawCurveRecSub的一个简单推广，如图11-47所示。把曲面不断分割到接近平面的四边形，然后调用程序DrawQuadrilateral，用第15章和第16章要介绍的方法，显示消隐后的平面四边形。若采用Bézier形式，则这个过程更简单。

平坦程度是通过计算由曲面的四个角点中的三个所确定的平面与其余13个控制点的距离来确定的；其中的最大距离必须小于ε。它仅是曲线平坦测试的一个推广。[LANE79]中讨论了另一种更有效的平坦测试方法。当然，同样也可以让递归进行到一个固定深度，用一些额外的分割步骤来代替平坦测试。

```
typedef double Coeffs[4][4][3];

void DrawSurfaceFwdDif (
        Coeffs A,   /* Q(s,t)的系数 */
        int n_s,    /* s 为常数的等参数曲线的画线数目，一般取5~10 */
        int n_t,    /* t 为常数的等参数曲线的画线数目，一般取5~10 */
        int n)      /* 每条曲线的画线步数，一般取20~100 */
{
    /* 初始化 */
    double δ_s = 1.0 / (n_s − 1.0); double δ_t = 1.0 / (n_t − 1.0);
    /* "*" 表示矩阵乘法 */
    DD_x = E(δ_s) * A_x * E(δ_t)^T;
    DD_y = E(δ_s) * A_y * E(δ_t)^T;
    DD_z = E(δ_s) * A_z * E(δ_t)^T;

    /* 画s为常数的等参数曲线n_s，s = 0, δ_s, 2δ_s,…1 */
    for (i = 0; i < n_s; i++) {
        /* 根据11.2.9节中的过程来画曲线 */
        DrawCurveFwdDif (n, First row of DD_x, First row of DD_y, First row of DD_z);
        /* 下一次重复画线 */
        使用公式11.92计算DD_x, DD_y和DD_z;
    }
    /* 对矩阵DD_x, DD_y, DD_z进行转置，以便按行t继续画线 */
    DD_x = DD_x^T; DD_y = DD_y^T; DD_z = DD_z^T;
    /* 画t为常数的等参数曲线n_t，t = 0, δ_t, 2δ_t, 3δ_t,…1 */
    for (i = 0; i < n_t; i++) {
        DrawCurveFwdDif (n, First row of DD_x, First row of DD_y, First row of DD_z);
        /* 下一次重复画线 */
        将式(11-92)应用于DD_x, DD_y和DD_z;
    }
}   /* DrawSurfaceFwdDif */
```

图11-46 用 s和 t的等参数曲线显示曲面的程序

```
void DrawSurfaceRecSub (surface, ε)
{
    /* 测试控制点是否在误差厚度为ε的平面内 */
    if (Flat (surface, ε))
        DrawQuadrilateral (surface);      /* 如果在误差范围内，把曲面画成四边形 */

    else {                                /* 如果曲面不够平坦，把曲面分成四个子曲面 */
        SubdivideSurface (surface, &surfaceLL, &surfaceLR, &surfaceRL, &surfaceRR);
        DrawSurfaceRecSub (surfaceLL, ε);
        DrawSurfaceRecSub (surfaceLR, ε);
        DrawSurfaceRecSub (surfaceRL, ε);
        DrawSurfaceRecSub (surfaceRR, ε);
    }
}   /* DrawSurfaceRecSub */
```

图11-47 参数曲面的消隐显示程序。DrawQuadrilateral绘制较为平坦的单个四边形。当曲面平坦到与平面的误差在ε范围内时，Flat返回true

分割曲面首先是将曲面沿一个参数，例如s，分成两半，然后再分别对每块分割曲面沿参数t分割成两块。应用11.2.7节所介绍的曲线分割方法，对每条由四个控制点定义的等参数曲线做分割。

这一思想如图11-48所示，最后得到了标有*ll*,*lr*,*rl*和*rr*的四个曲面，这些记号是图11-35的拓广。另外，当曲面在一个参数方向上相对很平坦时，也可以只沿另一个参数方向分割。[LANE80a]中对分割过程做了深刻的理论分析，也介绍了如何求解曲线与曲线或曲面与曲面之间的交。

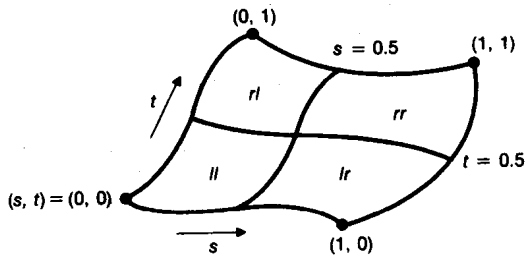


图11-48 曲面被分割成四块，即*ll*, *lr*, *rl* 和*rr*，每一块都比原曲面更“平坦”

自适应分割的一个关键问题是用四边形逼近曲面时可能会产生裂缝。如图11-49所示。这是由相邻曲面片之间所做的分割层次不同引起的。做固定深度的分割或者设置平坦阈值ε非常小，都可以避免产生分割裂缝，但是这两种解决办法都会带来不必要的分割操作。另一种方法是调整相邻的逼近四边形，如图11-50所示。Clark[CLAR79]和Barsky、DeRose及Dippé[BARS87]曾采用了这种基本策略。

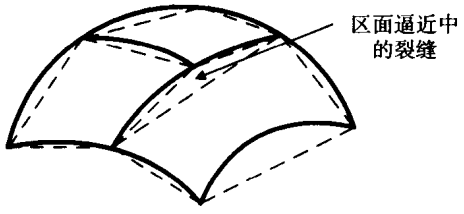


图11-49 分割曲面的三个逼近四边形在逼近过程中出现裂缝

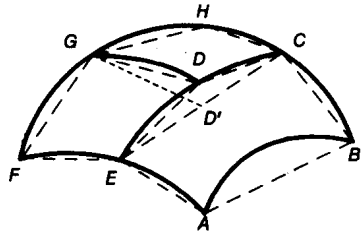


图11-50 递归分割中的裂缝消除。最简单的算法是显示四边形*ABCE*、*EFGD*和*GDCH*。复杂一些的算法是显示四边形*ABCE*、*EFGD'* 和*GD'CH*。用顶点*D'* 代替*D*

过程DrawQuadrilateral要显示的是一个接近平面的四边形。将四边形显示成消隐曲面的最好方法是进一步将其分成四个三角形，如图11-51所示。这种策略避免了人为视差。

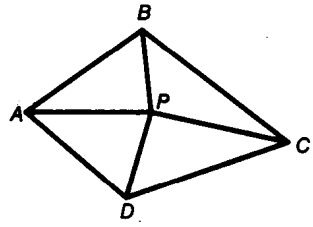


图11-51 要显示的四边形*ABCD*首先分割成四个三角形。新点*P*是对点*A*, *B*, *C*和*D*取平均

有时只需要显示双三次曲面的一部分。比如，一条穿过曲面的管道使曲面上有一个洞。可以采用修剪曲线（trimming curve）部分显示曲面，它是双三次曲面上的一条定义在(s, t)参数空间而非(x, y, z)空间上的样条曲线。当用DrawSurfaceFwdDif（见图11-46）显示一个被修剪曲线限制的曲面时，修剪曲线上的(s,t)值用来限定迭代步骤的开始和停止。

[FORR79]中介绍了其他显示双三次曲面的方法。

11.4 二次曲面

形如

$$f(x, y, z) = ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hy + 2jz + k = 0 \quad (11\text{-}93)$$

的隐式曲面方程定义了所有的二次曲面。比如，当$a = b = c = -k = 1$且其他系数为零时，方程

定义了一个以原点为中心的单位球体。当a到f都是零时，方程定义的是一个平面。在一些特定应用领域，比如分子造型[PORT79；MAX79]，二次曲面十分有用。同时，实体造型系统也集成了二次曲面。回忆一下，用三次有理曲线可以表示圆锥曲线；同样，用有理双三次曲面也可以表示二次曲面。因而，当只能用二次曲面表示时，可以用隐式二次方程表示有理曲面。使用二次曲面还有以下原因：

- 容易计算曲面的法线。
- 容易判断一个点是否在曲面上（只要将点坐标代入式(11-93)，然后检测其结果是否在零的某个ε领域内）。
- 给定x和y就很容易求出z（这在隐藏面算法中非常重要，见第15章）。
- 易于计算两个曲面的交。

式(11-93)也可以表示成

$$P^{\mathrm{T}} \cdot Q \cdot P = 0 \tag{11-94}$$

其中

$$Q = \begin{bmatrix} a & d & f & g \\ d & b & e & h \\ f & e & c & j \\ g & h & j & k \end{bmatrix} \qquad P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{11-95}$$

用Q表示的曲面做平移和放缩变换时也很方便。给定一个4×4的变换矩阵M（如第5章中介绍过的形式），变换后的二次曲面Q'由下式给出：

$$Q' = (M^{-1})^{\mathrm{T}} \cdot Q \cdot M^{-1} \tag{11-96}$$

由隐式方程 $f(x,y,z) = 0$所定义的曲面的法向量为$[\mathrm{d}f/\mathrm{d}x, \mathrm{d}f/\mathrm{d}y, \mathrm{d}f/\mathrm{d}z]$。这要比11.3.4节讨论的双三次曲面的法向量更容易求解。

11.5 小结

本章涉及了关于曲线与曲面的表示的一些重要概念，运用这些表示实现交互系统，这些内容已经足够了。这些内容在理论方面的处理可参阅[BART87；DEBO78；FARI88；FAUX79；MORT85]。

多边形网格是分段线性的，适合表示多面体，但对曲面体就不太适合。分段连续的三次参数曲线和双三次曲面广泛应用于计算机图形学和CAD中，用来表示曲面物体，主要是基于以下考虑：

- 单个x，y值允许有多个值与之对应。
- 可以表示无穷大斜率。
- 具有局部控制性，以便改变一个控制点仅仅影响到曲线周围一部分。
- 根据具体应用，可以插值或逼近控制点。
- 计算的有效性。
- 可以通过分割和插入结点等方法进行细分处理，加快了显示和交互控制。
- 曲线或曲面的变换，可通过控制点的变换获得。

虽然这里只讨论了三次曲面，还可以使用高于三次或低于三次的曲面。前面提到的内容对一般次数n的参数曲线和曲面都适用。

习题

11.1 对平面方程的系数A和B推导出类似于式(11-2)的等式。假设从平面正侧看，多边形顶点

是按逆时针的方向排列的。用A，B，C给出的平面法向指向平面正侧（这是11.1.3节要把计算B的面积取负的原因）。

11.2 写一个程序计算平面方程系数，已知一个近似平面的n个顶点多边形。顶点次序按逆时针方向排列，与习题11.1定义一样。当$n=3$时用几个已知平面进行测试；然后再用更大的n测试。

11.3 求出式(11-11)中定义的参数表示的直线的几何矩阵和基矩阵。

11.4 实现图11-18所给出的过程DrawCurve。用它显示几个系数cx, cy和cz各不相同的曲线。试着使曲线与本章图中的曲线段相对应。想想为什么很难做到这一点？

11.5 证明，若平面曲线$[x(t)\ \ y(t)]$具有G^1连续性，则曲线段拼接点两侧的斜率dy/dx相等。

11.6 令$\gamma(t)=(t, t^2)$，其中$0\leqslant t\leqslant 1$，并令$\eta(t)=(2t+1, t^3+4t+1)$，其中$0\leqslant t\leqslant 1$，注意$\gamma(1)=(1,1)=\eta(0)$，所以γ和η在拼接点处C^0连续。

a. 画出曲线$\eta(t)$和$\gamma(t)$，$0\leqslant t\leqslant 1$。

b. $\eta(t)$和$\gamma(t)$在拼接点处具有C^1连续性吗？（计算向量$d\gamma/dt(1)$和$d\eta/dt(0)$来验证这一点。）

c. $\eta(t)$和$\gamma(t)$在拼接点处具有G^1连续性吗？（计算b中两个向量的比值来验证这一点。）

11.7 曲线$\gamma(t)=(t^2-2t+1, t^3-2t^2+t)$和$\eta(t)=(t^2+1, t^3)$，定义在区间$0\leqslant t\leqslant 1$内。由$\gamma(1)=(1,0)=\eta(0)$知两曲线相接。证明两曲线在拼接点处具有$C^1$连续性，但没有$G^1$连续性。以$t$为变量画出两条曲线以验证为什么这个行为发生。

11.8 证明两条曲线$\gamma(t)=(t^2-2t, t)$和$\eta(t)=(t^2+1, t+1)$在$\gamma(1)=\eta(0)$处都是C^1和G^1连续的。

11.9 分析四个相邻的控制点共线对B样条曲线的影响。

11.10 写一个程序，可以输入任意一个几何矩阵、基矩阵和控制点列表，并画出相应的曲线。

11.11 求两条相接的Hermite曲线具有C^1连续性的条件。

11.12 假设关于Hermite几何条件和Bézier几何条件的等式为$R_1=\beta(P_2-P_1)$，$R_4=\beta(P_4-P_3)$。给定四个均匀分布的Bézier控制点$P_1=(0,0)$，$P_2=(1,0)$，$P_3=(2,0)$，$P_4=(3,0)$。证明：若要使参数曲线$Q(t)$从P_1到P_4具有恒定的速率，则系数β必须等于3。

11.13 写一个交互程序，允许用户生成或分割分段连续的三次曲线。用B样条作为曲线的内部表示。允许用户指定曲线的交互控制方式——用Hermite曲线、Bézier曲线或者B样条曲线。

530 11.14 证明B样条曲线内部的重控制点并不影响拼接点处的C^2连续性。可以这样证明，写出由控制点P_{i-1}，P_i，P_{i+1}，$P_{i+2}=P_{i+1}$，P_{i+3}构成的两条曲线段的表达式，然后求出第一段曲线在$t=1$以及第二段曲线在$t=0$的二阶导数，两者应该相等。

11.15 求出式(11-47)定义的Catmull-Rom样条曲线的调配函数。它们的和是否总为1，是否都不是零？若不是，则此样条曲线没有包含在点的凸包中。

11.16 根据式(11-49)、式(11-50)和式(11-19)，并用式(11-32)中的几何矩阵G_{Bs_i}，求出Kochanek-Bartels样条曲线的基矩阵M_{KB}。

11.17 写一个交互程序，允许用户生成、交互控制和细分分段连续的β样条曲线。试改变β_1和β_2观察曲线的形状效果。

11.18 写一个交互程序，允许用户生成、交互控制和细分分段连续的Kochanek-Bartels曲线。试着改变a, b, c观察曲线的形状效果。

11.19 分别用向前插分法和递归分割法实现曲线显示的程序。以同样的光滑效果显示不同类型的曲线并比较它们之间的执行时间。

11.20 为什么均匀B样条曲线的式(11-36)要写成$Q_i(t-t_i)$，而非均匀B样条曲线的式(11-43)写成了$Q_i(t)$?

11.21 已知平面非均匀B样条曲线以及曲线上的一点(x, y)，写一个程序求出相应的t值。必须考虑到一种情况，对一个给定的x值（或y值），可能有多个对应的y值（或x值）。

11.22 已知Bézier曲线要在t值处分成两段，用图11-35中的Casteljau构造方法求出分割矩阵$D_B^L(t)$和$D_B^R(t)$。

11.23 用推导Hermite曲面表示式(11-82)的方法推导出Bézier曲面的表示式(11-86)。

11.24 分别写一个程序用向前差分法和递归分割法显示三次参数曲线。改变步长δ和误差度量ε的值并考察这些参数对曲线形状的影响。

11.25 已知有四个相接的Hermite曲面片交于同一个点以及从这点出发的四条边界线，求出四个几何矩阵以及它们之间应该满足的关系。

11.26 令$t_0=0, t_1=1, t_2=3, t_3=4, t_4=5$。用这些值求出$B_{0,4}$和其定义中的每一个权函数。然后再在区间$-3\leqslant t\leqslant 8$上画出这些函数。

11.27 将式(11-44)中的$B_{i,4}(t)$递推关系展开成一个显式表达式。可以参见图11-26。

11.28 写一个程序，输入一个结点序列和一列控制点，显示非均匀、非有理B样条曲线。并提供一个用户选项，决定用哪种方法计算$B_{i,4}(t)$：(a)用式(11-44)中的递推关系；(b)用习题11.27中求出的显示表达式。测出两种方法所需的时间。更快的方法一定是更好的方法吗?

11.29 扩展习题11.28中的程序，使其允许用户可以对B样条曲线做交互的输入和修改。

11.30 写一个程序，分别用两种方法对曲线进行递归分割：自适应的并使用平坦测试；使用统一的、固定的分割深度。先用自适应的分割法显示曲线，注意它所需要的最大分割深度。再用这个深度作为固定深度显示同一条曲线。对不同的曲线比较一下两个程序的执行时间。

531

第12章 实体造型

第11章所讨论的表示方法可以描述二维和三维的曲线和曲面。正像2D直线和曲线的集合不足以描述一个闭区域的边界一样，一个3D平面和曲面的集合也不能够围成一个封闭的体。但在许多应用中，重要的是能够区分一个3D形体的内部、外部及其表面，这样才能导出形体的许多性质。例如，在CAD/CAM领域，一个实体如果能充分利用它的几何特征来建模，那么在制造它前，就可以对它施行许多操作。我们可能希望能判定两个形体彼此是否相交，例如，机器人的手臂是否会碰到其他物体，或者，切削刀具是否只切削预定的材料。在机械仿真中，重要的是计算物体（如齿轮）的特性，如体积和质心等。在实体造型中应用有限元分析，就是通过有限元造型来确定诸如应力和温度等因素。实体的一种合适的表示甚至可能在实体的数控加工制造中为刀具自动生成加工指令代码。另外，像产生折射透明等一些图形学技术，与光线从何处进入和穿出实体的判断有关。这些应用都是实体造型的例子。构造实体的需求促进了表示实体的各种方法的发展。本章将简述这些表示方法。

12.1 实体表示

把形体表示成看起来像实体，这并不表明，这种表示能力对实体的表示是合适的。考察一下到目前为止，我们是怎样表示形体的，把实体看成是直线段、曲线、多边形和曲面等的罗列。如图12-1a所示，其中的线段定义了一个立方体了吗？如果把形体每一侧的四条线段看成是构成四边形面的边界，那么这个图所表示的就是立方体。但是在这种表示方式中没有任何理由要求对直线段做这样的解释。例如，把任一这样的面去掉，同样的这些直线段可用于画这个图。在画图中，如果我们把连接直线段的每一个平面环定义成一个多边形面，又会怎样？在图12-1b中，由图12-1a中所有的面，再加上一个额外的悬挂面，会产生一个不能界定其体积的形体。在12.5节我们将看到，如果想保证所表示的造型就是实体，那么对实体的定义需要一些额外的约束。

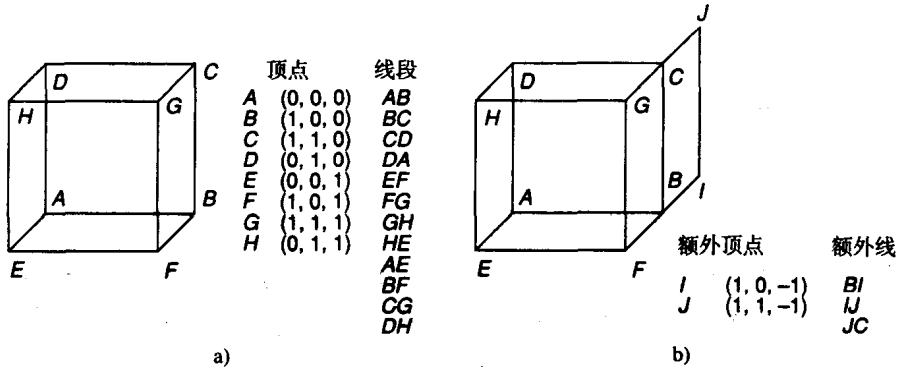


图12-1 a) 由12条直线组成的线框立方体，b)有一个额外面的线框立方体

Requicha[REQU80]给出了实体的表示形式应具有的性质。表示的域应足够大，以允许表示所需的实际形体。理想的表示方式应该是无歧义的，即不像图12-1a所表示的形体那样，会有“想要表示什么样的形体”这样的疑问，并且给定的表示应该对应一个且仅一个实体。无歧

义的表示也称为是完备的。如果一种表示可以仅用一种方式把给定的实体编码表达出来，那么这样表示是唯一的。如果一种表示可以保证惟一性，则像“判别两个形体是否相同”这样的操作就容易了。精确表示是不用逼近方式来表示形体。正像许多只能画直线段的图形系统用逼近方法表示光滑曲线一样，一些实体造型表示方式也是用逼近方式表示形体。一种理想的表示方法不应产生无效的表示（所谓无效表示是指它没有与相应的实体相对应），如图12-1b所示。另一方面，借助于交互实体造型系统，它应该容易生成有效表示。我们希望一个形体在旋转、平
534 移和其他操作下保持封闭性，即在有效形体上进行这些操作，其结果应仍然是有效的。表示方式还应该是紧凑的，便于节省存储空间，同时在分布式系统中可以减少通信时间。最后，表示方式应允许使用高效的算法以计算指定的物理性质，对我们而言最重要的是生成图像。

事实上，很难找到这样的表示方法，能满足上述所有这些性质，通常要有所折衷。由于这里只讨论目前正在使用的几种主要的表示方式，重点是提供足够的细节，以便能够理解这些表示方式是如何在图形软件中体现出来的。实体造型方面的更多的细节可参见[REQU80；MORT85；MANT88]。

12.2 正则布尔集合运算

不管形体是如何表示的，我们希望能把它们拼合起来，以生成新的形体。最直观和最常用的方法是利用集合论中的布尔集合运算并、差、交等，如图12-2所示。三维情形与二维情形类似。但是，对两个实体进行通常的集合运算，不一定能得到一个实体。例如，两个立方体进行通常的集合运算如图12-3a到图12-3e所示，图中依次得到的是实体、平面、线段、点以及空集。

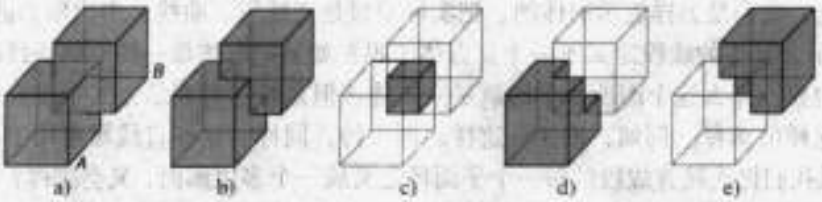


图12-2 布尔运算：a)实体A和B，b)$A\cup B$，c)$A\cap B$，d)$A-B$，e)$B-A$

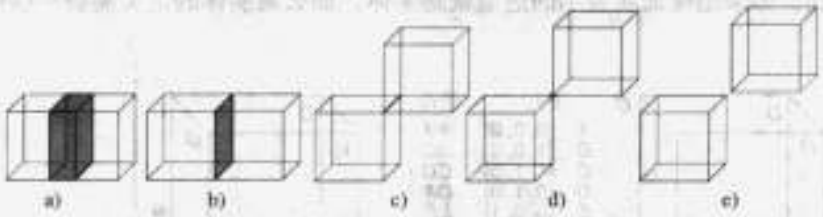


图12-3 两个立方体的普通集合运算可能生成a)实体、b)平面、c)直线段、d)点或者e)空集

我们用正则布尔集合运算[REQU77]来代替通常的布尔运算，用$\cup^*$、$\cap^*$和$-^*$来表示，这样，两个实体进行布尔运算后总能生成实体。例如，图12-3a和图12-3e的情形，正则布尔交运算的结果与通常的布尔交运算一样，但对于图12-3b到图12-3d的情形，正则求交运算结果是空集。

为了探讨两种布尔运算之间的差别，我们把形体看成是一个由内部点和边界点构成的点集，如图12-4a。边界点集由一个形体到它的补集之间的距离为0的点集构成。边界点不一定是形体上的点。一个形体的闭集包含它的所有边界点，而开集则不包含。一个点集与它的边界点集的并集称为这一集合的闭包，如图12-4b所示。闭包本身也是闭集。一个闭集的边界是其边界点的集合，

而它的内部则是由这个集合除边界以外的所有其他点构成，如图12-4c所示，因此，相对于这一形体而言，内部是边界的补集。集合的正则化定义为这个集合的内部点的闭包。图12-4d所示就是图12-4c中形体的闭包，因此，就是图12-4a所表示的形体的正则化。若一个集合与它的正则化相同，则这个集合被称为正则集。注意正则集不包含那些与点集内部点不相连的边界点；因此它没有悬挂的边界点、线或者面。与普通的集合运算相对应，正则的集合运算符可以定义如下：

$$A\ op^*\ B = \text{closure}(\text{interior}(A\ op\ B)) \tag{12-1}$$

式中op表示∪、∩或者－。对正则集应用正则集合运算符时，只生成正则集。

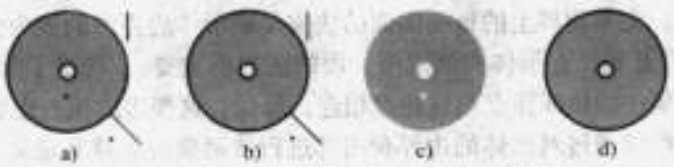


图12-4 形体的正则化。a)形体由内部点和边界点组成。内部点由浅灰色表示，边界点的一部分由黑色表示，其余部分由深灰色表示。该形体含有悬挂的和分离的点、线，并且在形体的内部还有一个边界点，这一点不是形体中的点。b)形体的闭包。形体的所有边界点都属于该形体。a)中在形体内部的一个边界点已属于形体的内部点。c)形体的内部。悬挂的或分离的点和线段都已删除。d)形体的正则化集是形体内部的闭包

现在我们来比较对正则集进行普通集合运算和正则集合运算时，两者之间的差别。考虑两个形体（如图12-5a所示），两形体放置如图12-5b所示。两形体的普通布尔交集包括每个形体的内部和边界与另一个形体的内部和边界的交集，如图12-5c所示。相反，两形体的正则布尔交集（如图12-5d所示）则包括两者的内部的交集以及一个形体的内部与另一个形体的边界之间的交集，后者仅仅是两形体的边界交集的一个子集。用于确定这个子集的准则决定了为何正则布尔运算与普通布尔运算的求交运算之间有差别，后者包括了两集合边界的交集的所有部分。

535
↓
536

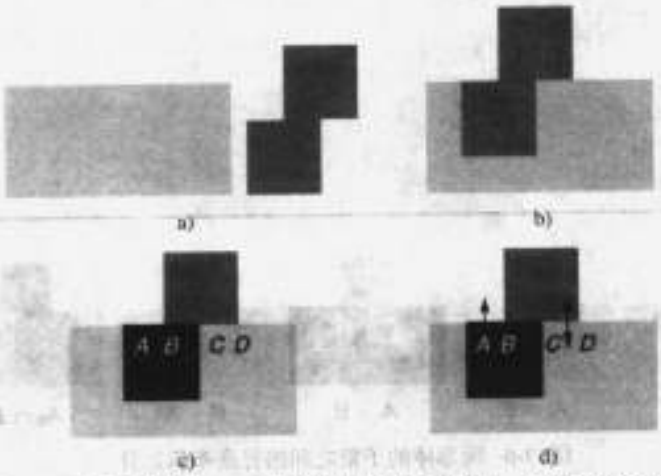


图12-5 布尔交集。a)两个形体的截面；b)两形体求交前的位置；c)普通布尔求交运算导致了一个悬挂面，如图截面中的线段CD；d)如果两形体位于它的同一侧，正则布尔交集包含了一部分共享边界(AB)；如果形体位于边界的异侧，不包含一部分共享边界(CD)。边界与内部的交集(BC)总是属于正则交集内的

直观上看，当且仅当两形体的内部位于这段共享边界的同一侧，两形体边界与边界的一段交集才属于正则集合运算的交。因为两形体与那段共享边界直接相连的那些内部点集是两形体

的交集，这一段边界也肯定属于这些内部点集的闭包内。考察两多面体的一侧表面共面时的共享边界情形。如果定义了两形体表面的向内(或向外)的法向，那么，判断两形体的内部点集是否位于它们的共享边的同一侧就简单了。若它们的法向是同向的，则内部点集在同侧。因此图12-5d中的AB段就属于正则交集。记住，如果一个形体的边界与另一个形体的内部相交，如图12-5d中的BC线段，则这部分边界总是属于正则交集的。

再来考察一下，当两形体的内部点集分别位于它们的共享边的异侧时，会怎样？如图12-5d中CD段情形，与这条边相邻的内部点集都不属于两形体的交集。因此，这段共享边与两形体的交集的内部不相连，从而不属于两形体的正则交集。对共享边属于交集进行这一额外的限制，保证了交集是正则集。交集边界上的每一侧面的法向是原形体边界中的成为交集边界部分的面
537 法向。(在第16章将会看到，在形体的消隐中，面的法向很重要。)确定了哪些面在边界上后，如果边界与边界的交集中的边或顶点与这些面相连，那么，这些边或顶点也属于交集的边界。

每一种正则运算可以通过对形体的边界和内部进行普通集合运算来定义。表12-1表示如何对形体A、B作正则集合运算。图12-6表示运算结果。A_b和A_i分别表示A的边界和A的内部。$A_b \cap B_b$同侧表示A和B的共享边界的一部分，其中A_i和B_i在它的同一侧。也就是说，对于共享边界上的某一点b，如果至少有一点内部点i与之相近，则点i同时属于A和B。$A_b \cap B_b$异侧表示A和B共享边界的一部分，其中A_i和B_i位于它的两侧。此时，对于共享边界上的某一点b，找不到这样的内部点i与之相近，使得i同时属于A_i和B_i。每一种正则运算定义为由表12-1中所在列中带·号项所对应的普通集合运算的并集。

表12-1 正则布尔运算

集 合	$A \cup^* B$	$A \cap^* B$	$A -^* B$
$A_i \cap B_i$	·	·	
$A_i - B$	·		·
$B_i - A$	·		
$A_b \cap B_i$		·	
$B_b \cap A_i$		·	·
$A_b - B$	·		·
$B_b - A$	·		
$A_b \cap B_b$同侧	·	·	
$A_b \cap B_b$异侧			·

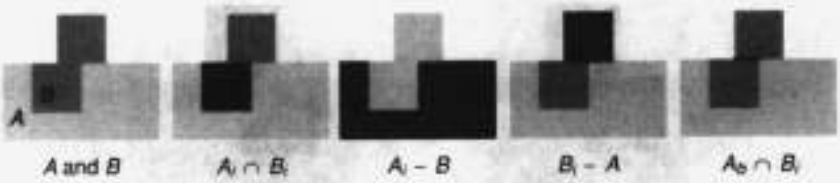


图12-6 两形体的子集之间的普通布尔运算

注意，对于每一种情形，正则运算所生成的边界中的每一部分，或者在原一个形体的边界上，或者在原来两个形体的边界上。在计算$A \cup^* B$或者$A \cap^* B$时，计算得到的形体的表面的法向
538 是原来一个形体或者两个形体所相应的表面的法向。但在$A -^* B$情形中，运算的结果通常是A被B挖去一部分，这一部分的每一表面的法向一定是与B在这一表面上的法向方向相反。这一点与$A_b \cap B_b$异侧和$B_b \cap A_i$的边界部分得到的结果相同。或者，$A -^* B$也可写成$A \cap^* \bar{B}$。$\bar{B}$(B的补

集)可以通过对B的内部取补以及对B的边界处的法向取反得到。

作为一种用户界面技术，正则集合运算已用于大多数从简单形体生成复杂形体的造型表示方法中，这些造型方法将在后面讨论。构造实体几何这一造型表示方式显然也包含这一操作手段。在以下几节中，我们将详细叙述几种表示实体的方法。

12.3 基本实体举例法

在基本实体举例（primitive instance）中，造型系统定义一个基本的三维实体形状的集合，它们与应用领域有关。这些基本实体用典型的参数形式表示，而不用第7章所讨论的变换，但也有其他性质。例如，一个基本实体可能是一个正规的棱锥体，具有与顶点相连的面数，面数由用户指定。基本实体举例类似于参数化对象，如第2章的菜单，只是这里的对象是实体。参数化基本实体可看成定义一个零件类，这个类的数目随各种参数而变化，这是一个重要的CAD概念，称为*群组技术*。基本实体举例法常用于一些相对复杂的对象，如齿轮或者螺栓等。如果它们用更简单的实体通过布尔并运算来生成，那就太繁锁了，所以用一些简单的参数来刻画。例如，齿轮可以用它的半径或者齿数来参数化表示，如图12-7所示。

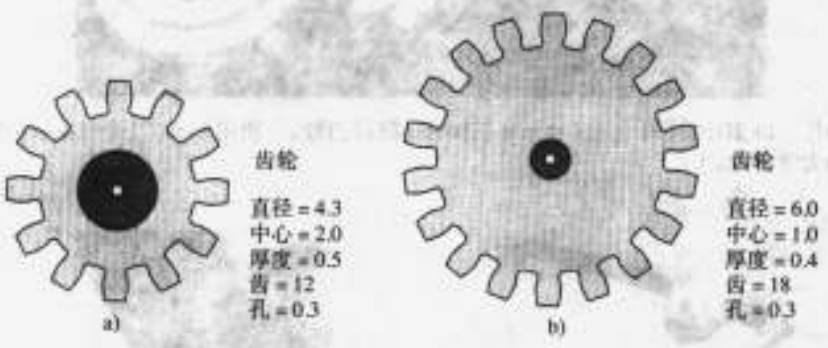


图12-7 由基本实体举例法定义的两个齿轮

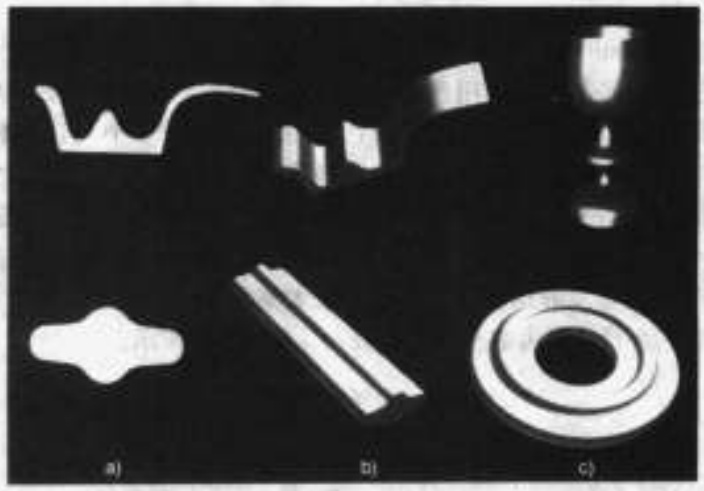
虽然可以建立基本实体举例的层次结构，但每一叶子节点仍然是一个单独定义的对象。在基本实体举例法中，不能通过基本实体的组合（例如利用正则的集合运算）来生成更高层次的基本实体。因此，生成新的基本实体的惟一方法是通过编写代码来定义。类似地，绘制实体图或者是确定其质量等性质的例程必须对每一个基本实体单独编写。 539

12.4 扫掠表示法

一个形体沿空间路径做扫掠运动，其运动轨迹定义了一个新的形体，这一过程称为*扫掠*（sweep）。最简单的扫掠是由一个2D区域沿区域所在平面垂直的线性路径进行扫掠生成一个体来定义的。这种体被称为*平移扫掠体*或者*拉伸*。它可以自然表示一类形体，可以通过给定的截面形状沿模板拉伸金属或塑料制品而生成的形体。在这些简单的情形中，每个扫掠体的体积就是扫掠体的截面积与扫掠长度的简单乘积。简单的推广涉及截面在拉伸过程中可以缩放，生成带斜度的形体，或者截面沿着一条与截面所在平面不垂直的线性路径扫掠。*旋转扫掠*是一个区域绕轴旋转生成的形体。图12-8表示两个形体以及由此生成的简单的平移和回转扫掠体。

被扫掠的形体不一定是二维的。实体的扫掠在机床刀具移动或者机器人沿路径运动所生成的区域构造中很有用，如图12-9所示。扫掠体在扫掠过程中，它的面积或者体积在大小、形状

或者扫掠方向等方面都可改变，扫掠路径也可以是任意的曲线，这种扫掠称为一般扫。2D区域的一般扫在计算机视觉中[BINF71]称为广义柱面，它的构造是参数化的2D截面沿任意曲线运动生成，且截面线始终与路径垂直。高效的一般扫生成方法是十分困难的。例如，路径和形体形状选择可能使得扫掠体自交，从而使体积计算很复杂。同样，一般扫也不总能生成三维实体。例如，一个2D区域沿它所在平面做扫掠，生成的是另一个2D区域。



540 图12.8 扫掠。a) 2D区域用于定义b)平移扫掠和c)旋转扫掠。（使用Alpha_1造型系统创建，经犹他大学许可。）



图12.9 a) 生成实体扫掠的刀具的轨迹用于定义飞机零件的模型b)，c)为实际零件原型，由自动生成指令加工得到。（利用Alpha_1造型系统生成。由犹他大学许可。）

一般地，如果事先不转换成其他某一种表示，对扫掠体施加正则集合运算是很困难的。在正则集合运算中，即使是最简单的扫掠体也是不封闭的。例如，两个简单的扫掠体的并，

一般不是一个简单的扫掠体，如图12-10所示。虽然扫掠存在着封闭性和计算等问题，但它仍然是一种自然的和直观的构造各种实体的方法。因此，许多实体造型系统允许用户构造扫掠体，但存储时却是我们要讨论的另外一种表示形式。

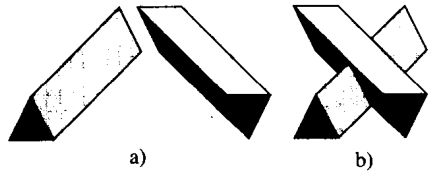


图12-10 a)两个由2D形体（三角形）构成的简单扫掠体，b)为a)中的两个扫掠体的并集，它本身不是由2D形体的简单扫掠体

541

12.5 边界表示法

边界表示（也称为b-reps）有点像我们在12.1节讨论的那种朴素的表示方法，它是用实体表面边界的顶点、边、面来刻划实体。某些边界表示限制在平面多边形边界，甚至要求面是凸多边形或者是三角形。如果允许用曲面来表示，那么要确定表面的构成就很困难了，如图12-11所示。曲面常用多边形来逼近。实体也可以用曲面片来表示，如果处理这些表示的算法能处理交线，而这样的交线一般会比原曲面的次数要高。边界表示由前面几章介绍的简单的向量表示发展而来，并且已用于目前许多造型系统中。由于它在图形学中被广泛采用，现已开发了大量的高效的技术来生成多面体实体的光滑消隐图，其中一些技术将在第16章讨论。

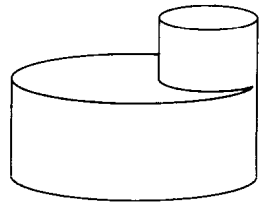


图12-11 这一实体共有多少个面

许多边界表示系统只支持边界是用二维流形表示的实体。根据定义，二维流形上的每一点存在一个任意小的邻域，使得这个小邻域与欧氏平面上的一个圆盘是拓扑等价的。也就是存在一个从小邻域与圆盘之间的连续的一一对应关系，如图12-12a和图12-12b所示。例如，如果有多于两个面共享一条边，如图12-12c所示，则这条边上一个点的任一邻域包含这些面中任一个面上的点。显然没有从这个邻域到平面圆盘之间的一一对应。但这一点从数学上证明并那么直接。因此图12-12c中的表面不是二维流形。虽然现在有些系统并没有这样的限制，但我们仍然把讨论范围限定在边界是二维流形上，除非在有的地方特别声明。

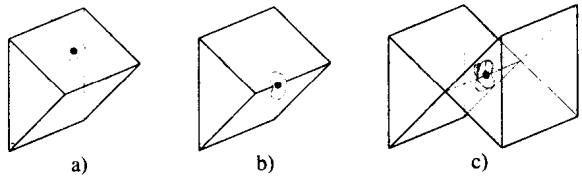


图12-12 在二维流形上，每个点（显示为黑点）都有一个周围的点的邻域（它是一个拓扑圆盘），如a)和b)中灰色部分所示。c)如果物体不是一个二维流形，那么它具有没有邻域（它是一个拓扑圆盘）的点

12.5.1 多面体和欧拉公式

多面体是由一组多边形作为边界构成的实体，它的每一条边属于偶数个多边形（在二维流形情形中只有两个多边形），它也要满足其他的一些约束（这一点将在后面讨论）。简单多面体是这样一个实体：它可以通过形变变成一个球形，也就是说，与环状体不同，这样的多面体没有洞。简单多面体的边界表示满足欧拉公式，它表示简单多面体的顶点数、边数和面数之间的一种不变关系：

$$V - E + F = 2 \tag{12-2}$$

式中V是顶点数，E是边数，F是面数。图12-13表示一些简单多面体以及它们的顶点数、边数

和面数。注意，如果有曲线边和非平面面，欧拉公式仍成立。欧拉公式只说明了一个实体是简单多面体的一个必要条件而非充分条件。可以构造这样的形体，它满足欧拉公式，但没有围成一个体，这可以在一个有效的实体上附加一个或多个悬挂面或悬挂边来构造，如图12-1b所示。因此，有必要加上约束以保证形体是实体：每一条边必须与两个顶点相连，每一条边只有两个面共享，每一个顶点至少有三条边，面之间不能相互贯穿。

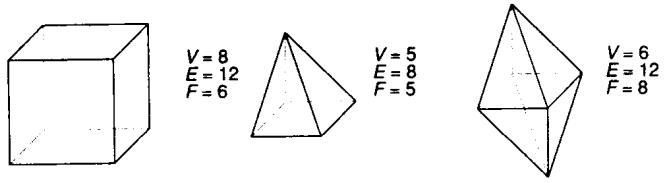


图12-13 多面体以及它们V，E和F的值，每一种情形，满足$V-E+F=2$

对于面上带洞的二维流形，有广义欧拉公式：

$$V - E + F - H = 2(C - G) \tag{12-3}$$

式中H是面上洞的数目，G是贯穿形体的洞的数目，C是形体相离部件的数目，如图12-14所示。如果实物是单部件的，那么其中的G称为亏格；如果是多部件的，那么G是每一部件上亏格的和。与前面一样，附加的约束条件也必须保证形体是实体。

542
~
543

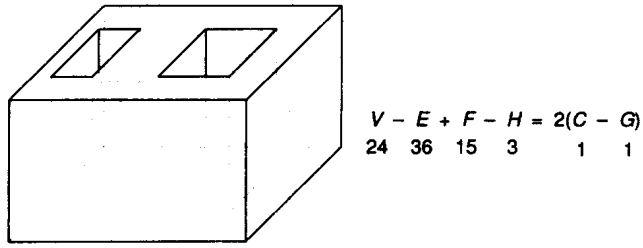


图12-14 多面体按式(12-3)分类，它的顶面有两个洞，底面有一个洞

Baumgart引入了欧拉算子的概念，它对满足欧拉公式的实体做运算，将实体转换成新的实体，也满足欧拉公式，方法是增加和删除顶点、边和面[BAUM74]。Braid、Hillyard和Stroud[BRAI78]说明了如何用少量的欧拉操作组合来生成新的实体，并指出，中间结果可以不必是有效实体。同时Mäntylä[MANT88]证明了所有有效的边界表示都可以通过有限步的欧拉操作来构成。另外也可以定义一些不影响实体的顶点、边、面数目的操作，它们通过移动现有的顶点、边和面来提拉形体，如图12-15所示。

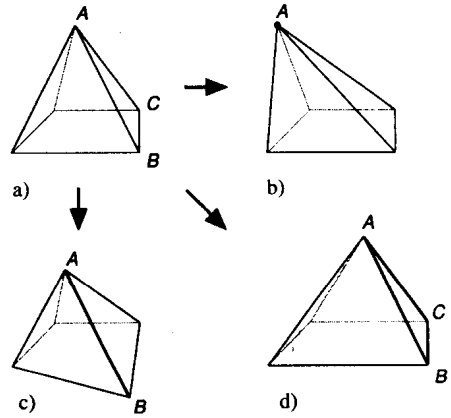


图12-15 a)形体做提拉操作，以移动b)顶点A，c)边AB，d)面ABC

最简单的边界表示可能是列出所有的多边形面，每一个面由顶点坐标列表表示。为了表示每个多边形面的方向，多边形顶点从实体外部看按顺时针方向排列。为避免被面共享的点的重复表示，在坐标列表中引入索引来表示面的顶点。在多边形顶点列表中，这种表示把边隐式地表示成两相邻的顶点。对于每个面，边不是显式地表示成顶点对，而是在边表中定义成索引列表。这些表示方法已在11.1.1节中详细讨论了。

544

12.5.2 翼边表示法

简单表示可能使得某些计算相当费时。例如，寻找某边相邻的两个面（例如，为帮助说明某一种表示方式给定的是有效实体）需要遍历所有面的边表。为了降低这些计算的复杂性，出现了更为复杂的边界表示。最有名的一个是由Baumgart[BAUM72；BAUM75]提出的翼边数据结构。如图12-16所示，翼边数据结构中每条边用指针来表示，以指向它的两个顶点，指向共享这条边的两个面，也指向另外与它的两个顶点相连的所有边中的四条边。每一个顶点有一个向后指针，指向与它相连的边中的一条，同时每个面指向它的边中的一条。注意，当两个相邻面中的顶点按顺时针方向排列时，共享边的顶点次序在两个相邻面上是反向的。我们把边的两个顶点标记为n和p，当边的方向是从点n到点p时，与该边相邻的右边的面记为这条边的p面，当边的方向是从点p到点n时，与边相邻的右边的面记为它的n面。对于图12-16中的边$E1$，如果n是$V1$，p是$V2$，则$F1$是$E1$的p面，$F2$是$E1$的n面。与每条边相连的四条边，可进行如下分类：与边的顶点n相连的两条边$E3$和$E2$分别是n面的下一边（顺时针方向）和p面的上一边（逆时针方向）；与顶点p相连的两条边$E4$和$E5$分别是p面的下一条边（顺时针方向）和n面的上一条边（逆时针方向）。这四条边就像两翼，翼边数据结构由此得名。

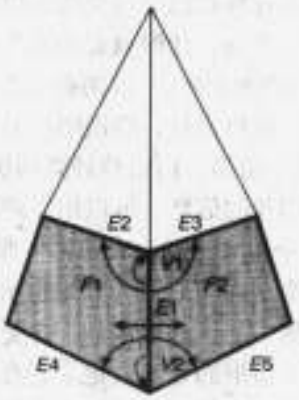


图12-16 $E1$的翼边数据结构。$V1$，$V2$，$F1$和$F2$的每一个都有一个向后指针，指向它们的一条边（未显示）

注意，这里描述的数据结构要求面上没有洞。如果每一个面用边环来表示，则这个限制可以去掉。面上的外环是顺时针的，如果有洞，则洞作为内环是逆时针的。这些将在19.1节中讨论。或者，可以引入辅助边，把各个洞的边界和外侧边界相连，当遍历面上的所有边时，每条辅助边要沿反向通过两次。由于辅助边的两侧是同一个面，它两侧的面指针指向同一个面，因此辅助边容易识别。 545

边界表示可以用来查找与每一面、边或顶点相邻的面、边或顶点。这些查找对应于九种邻接关系。翼边数据结构可以同时确定哪些顶点或面与一条边相关。但需要花较长的时间来计算另外一些邻接关系。翼边的一个吸引人的特性是实体的边、面和顶点的每一个数据结构都是固定的而且较小。只是每一数据结构的实例数随形体不同而不同。Weiler[WEIL85]和Woo[WOO85]讨论了翼边数据结构的时间-空间效率以及许多改进的边界表示数据结构。

12.5.3 布尔集合运算

利用正则的布尔集合运算，边界表示可以组合生成新的边界表示[REQU85]。Sarraga[SARR83]和Miller[MILL87]讨论了确定二次曲面间的求交算法。[TURN84；REQU85；PUTN86；LAID86]中给出了多面体组合的算法，Thibault和Naylor[THIB87]描述了一种基于空间划分二叉树的实体表示方法，这种表示方法将在12.6.4节中讨论。

[LAID86]中的一种方法是检查两形体的多边形，如果需要，则可对它们进行分割，以保证确定两形体之间的点、边、面的交是它们的点、边或者是面。然后，一个形体的多边形将依据另一形体分类，以确定它们是在另一形体的内部、外部或边界上。回顾一下表12-1，注意到，由于它是边界表示，我们只要考虑表中最后的六行，其中每一行表示原来两形体边界A_b和B_b之一或者两者中的一部分。分割后，每一形体的每个多边形或者全部在另一形体的内部（A_b

$\cap B_i$或$B_b \cap A_i$），或者全部在另一形体的外部（$A_b - B$或$B_b - A$），或者是共享边界的一部分（$A_b \cap B_b$同侧或$A_b \cap B_b$异侧）。

多边形也可以根据光线投射技术分类，这一技术将在15.10.1节讨论。从多边形的内部一点沿多边形的表面法向构造一个向量，然后找出另一形体中与这一向量有交，且相距最近的多边形。如果没有与此向量相交的多边形，则原多边形在另一形体的外面。如果最近的相交多边形与原多边形共面，此时就是边界与边界的交，比较两者的法向可以知道是哪一类的交（$A_b \cap B_b$同侧或者$A_b \cap B_b$异侧）。否则要检查两个多边形法向的点积。点积为正，说明原多边形在另一形体的内部；点积为负，则说明在另一形体的外部。如果向量在相交多边形的平面上，则点积为0；对于这一情形，向量被轻微的扰动，再与另一形体求交。

546 顶点相邻的信息，可以用来避免用这种方式对每一多边形分类这样的开销。如果一个多边形与一个分过类的多边形相邻（即与之同享顶点），并且与另一形体的表面没有交，则它可以归入同一类。在多边形分割的初始阶段，两形体的共同边界上的所有顶点可以做上标记。一个多边形与另一形体的表示是否有交可以通过检查它是否有边界顶点来确定。

每一多边形的分类决定了它在生成复合形体的操作过程中是否被保留，如12.2节所述。例如，在构成集合的并时，一个形体的任一多边形如果在另一形体的内部，则不予考虑。两形体之一的任一多边形如果不在另一形体内部，则这一多边形予以保留，除非是多边形间的共面情形。如果两个共面多边形的表面法向是反向的，则两者不予考虑；如果表面法向是相同的，则只保留两个多边形中的一个。如果形体是由不同材料构成的，则确定哪个多边形予以保留这样的判断就很重要。虽然$A \cup^* B$与$B \cup^* A$有相同的几何意义，但两者看起来可能会有差别，所以在多边形共面情形，两者可根据自己的喜好，选择其中之一作为运算的定义。

12.5.4 非多边形的边界表示法

如果形体本身不是多面体，则只能通过多面体表示来逼近含有曲面的形体，并且在可接受的精度下，逼近的数据量会很大。如图12-17所示，考察这样的问题，用多面体边界表示来表示一个圆柱形洞中的一个圆柱形形体。如果实际形体的边界是接触的，那么，即使两逼近的多面体的边界在初始时是一致的，不管逼近时用多少个多边形、当两逼近体之一被慢慢旋转时，两者就会相交。

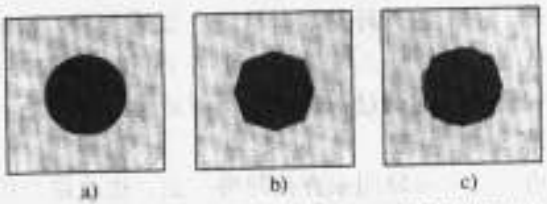


图12-17 a)圆柱在圆洞中的截面；b)圆洞和圆柱的多边形逼近；c)若逼近的圆柱相对于圆孔转动，则产生干涉

一种有前途的精确的边界表示方法，是用曲线定义的雕塑曲面。Alpha_1[COHE83]和Geomod[TILL83]造型系统可以构造像NURBS这样的自由曲面（见11.2.5节）。因为单个曲面自身可能不是封闭的，Thomas[THOM84]为Alpha_1系统开发一个算法，这一算法可以对只有部分边界的形体做正则的布尔集合运算，如图12-18所示。彩图I-31中

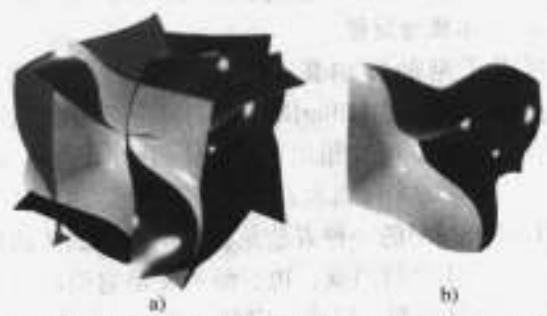


图12-18 部分有界的形体上的布尔集合运算。a)6个部分有界的形体，b)集合的交集定义一个波状形的立方体。（经犹他大学W.Thomas许可使用。）

的形体就是用Alpha_1系统构造的。

因为边界表示是表示实体表面的各个面元，因此表示是不惟一的。另外，正像前面提到的那样，许多基于边界表示的系统只处理表面是二维流形的形体。二维流形在正则布尔集合运算下不一定是封闭的。例如，两个边界表示的立方体，放置的位置是使得两个立方体只有一条公共边，两立方体做正则并运算，则这条公共边就由四个面共享。但是这样的配置在某些系统中是不允许的，例如基于翼边表示的系统。Weiler[WEIL88]讨论了一个非流形的、基于边界的造型系统，它可以处理线框形体，以及曲面和实体，如彩图I-32所示。

12.6 空间划分表示法

在空间划分表示中，实体被分解成相连的但不相交的实体组合，它比原来的实体有更多的基本实体，尽管不必与原来实体类型相同。基本实体可以在类型、大小、位置、参数化或方向等方面各不相同，很像小孩子搭积木时用的不同形状的积木。对形体分解到何种程度依赖于实体需要怎样的基本实体以便于准备完成感兴趣的运算。

12.6.1 单元分解法

空间划分中最一般的类型之一是*单元分解*。每一个单元分解系统定义了一个基本实体单元集，这些基本实体单元已经参数化，并且常常含有曲线边。单元分解法与基本实体举例法不同，这里，我们可以用简单的、基本的单元自底向上把它们“胶合”在一起复合成更为复杂的形体，胶合操作可理解为严格的并，其中的形体不能有交。对胶合单元的更进一步的限制是，两个单元共享一个点、一条边或一个面。虽然一个形体的单元分解表示是明确的，但不一定是惟一的，如图12-19所示。单元分解的有效性证明也较困难，因为每一对单元都必须进行可能的求交测试。不过，在有限元分析中，单元分解法是一种重要的表示方法。

547 ~ 548

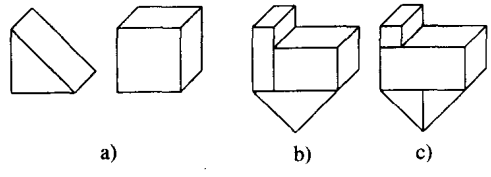
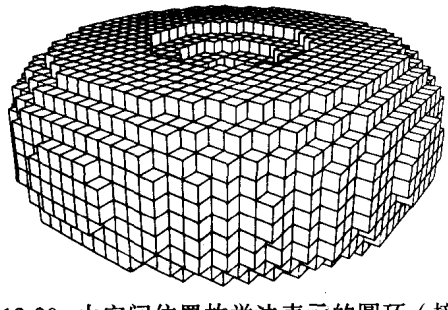


图12-19 a) 中的图可以组合成b)和c)不同的图形

12.6.2 空间位置枚举法

空间位置枚举（spatial-occupancy enumeration）是单元分解的一种特殊情形，这里实体被分解成相同的单元，并以固定的、正则的网格排列。这些单元常称为*体素*，类似于像素。图12-20表示了一个被空间位置枚举法表示的形体。最常见的单元类型是立方体，以正规的立方体排列的空间表示方式，被称为cuberille。用空间位置枚举法来表示形体时，只要控制网格上每一位置单元是否为空。在表示一个形体时，我们只须判断形体包含哪些单元，不包含哪些单元即可。因此，形体可以用惟一的且明确的所占单元列表方式表示出来。判断一个单元在实体的内部或外部是很容易的，要判断两个实体是否邻接也同样简单。空间位置枚举法常应用于生物医学领域，用来表示体数据，这些数据来自计算机X射线轴向分层造影（CAT）扫描等途径。



549

图12-20 由空间位置枚举法表示的圆环（摘自Siggraph '80 Conference Proceedings, *Computer Graphics* (14)3,July,1980，作者AHJ Christensen，由ACM公司许可使用。）

尽管空间位置枚举方法有很多优点，但它也有许多明显的缺陷，这些缺陷类似于二维形

状表示法，后者只用一位深度的位图表示。它没有“部分”充满的概念。因此，许多实体只能近似表示，图12-20就是一例。如果单元是立方体，则只有那些表面与单元立方体侧面平行并且顶点严格落在网格点上的实体，才能被精确表示出来。类似于位图中的像素，原则上，单元可以被分割成预定的大小，以增加表示精度。但是空间开销成了一个重要问题，因为在三维空间中，当体素分辨率为n时，表示一个形体就需要占用高达n^3个单元。

12.6.3 八叉树表示法

八叉树在表示层次上改进了空间位置枚举，以满足存储要求。八叉树实际上由四叉树而来，四叉树是一种二维表示格式，用于图像编码（参见17.7节）。正像Samet的深刻综述[SAME84]所说，这两种方法都是由许多研究人员独立发现的，四叉树在20世纪60年代后期至70年代早期出现[WARN69；KLIN71]，八叉树是在20世纪70年代后期至20世纪80年代早期出现[HUNT78，REDD78；JACK80，MEAG80；MEAG82a]。

四叉树和八叉树的基本思想都是二分法中的分而治之的方法。四叉树是对二维平面在两个方向上进行连续分割以形成象限的方式得到，如图12-21所示。当用四叉树来表示一平面区域时，根据象限与所分割的平面区域相交的程度，每一个象限是充满平面区域、部分充满或者为空（也分别称为黑色、灰色和白色），部分充满的象限再递归地分割成子象限，直到所有的象限类型相同（满或者空）或者分割到预先设定的中止深度。当四个同一层的相邻象限都是充满或者都是空时，删除这些象限并且让它们上一层从部分充满的父象限改成充满或者为空（用自底向上的方法，可以避免删除和合并操作[SAME90b]）。在图12-21中，在中止深度上所有部分充满的节点归入充满的节点类中。递归分割的过程可以用一棵树来表示，其中部分充满的象限作为中间节点，而充满和空的象限都为叶节点，如图12-22所示。这一思想可以与15.7.1节中讨论的Warnock区域细分算法进行比较。如果把节点归类的标准放宽，把在标准上下的节点都归类到或者是充满的或者是空的两类中，则区域表示将更加简洁，但表示精度却降低了。八叉树表示方式与四叉树类似，只是递归分割时是沿三个方向分割象限，如图12-23所示。

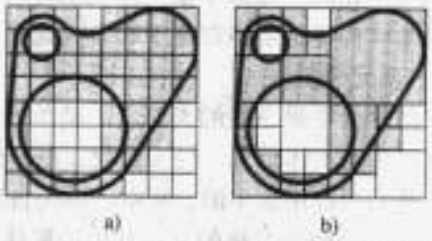


图12-21 形体表示，a)用空间位置枚举法表示，b)用四叉树表示

550

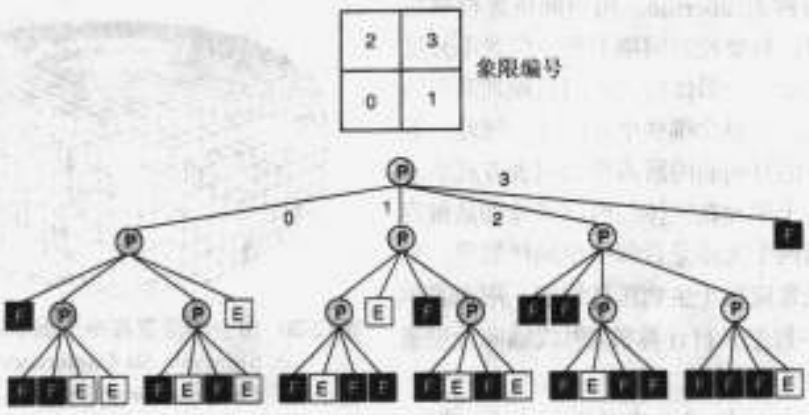


图12-22 图12-21中所示形体的四叉树数据结构。F＝充满，P＝部分充满，E＝空

四叉树的各个象限常用数字0到3来表示，而八叉树则用数字0到7来表示。由于没有标准的编号方式，也可以用一些助记符号。四叉树的象限是根据上一层父节点中心的方位走向来命名的：NW、NE、SW和SE。八叉树的象限的命名与四叉树类似，便于区分左(L)右(R)、上(U)下(D)以及前(F)后(B)：LUF、LUB、LDF、LDB、RUF、RUB、RDF以及RDB。

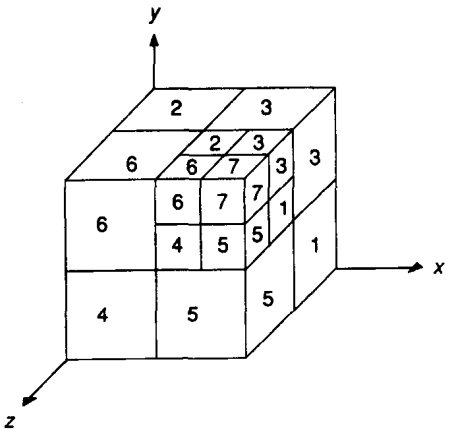


图12-23 八叉树枚举，其中刻度为0的象限不可见

除了一些最坏的情形外，可以证明，一个形体用四叉树或八叉树表示时，它的节点数分别与形体的周界或表面成正比[HUNT78；MEAG80]。这一关系是成立的，因为只是从形体的边界表示的需要出发，才增加节点分割。被分割的中间节点只是那些形体的部分边界穿过的节点。因此，在这些数据结构上的任何操作在执行时间上也与形体的周长或面积的大小成正比，数据结构在它包含的节点数目上是线性的。

虽然四叉树和八叉树的分而治之方法可以推广到任意维空间，但只用二叉树方法来表示也是可能的。二叉树在每个节点处沿单个轴向把空间等分，并在每层上依次沿一个新轴向做等分，而不是在树每个节点处沿所有轴向都做分割[TAMM84]。二叉树通常比与之类似的四叉树或八叉树有更多的节点。但叶子的数目并不比它们多，而且二叉树的计算处理算法可以表达得更为简洁。

1. 布尔集合运算和变换

在四叉树和八叉树的存储和处理的有效算法方面已做了许多工作[SAME84；SAME90a；SAME90b]。例如，四叉树和八叉树可以直接进行集合运算 [HUNT79]。为了计算两棵树S和T的交集或并集U，把两棵树自顶向下并列放置。图12-24表示二棵树的集合运算；这样的运算可直接推广到八叉树。检查每一对相匹配的节点。考虑集合并的情形。如果一对节点中的一个是黑的，相应的黑的节点加到U中去。如果节点对中有一个是白的，则利用节点对中的另一个节点的值，在U中生成相应节点。如果节点对中两个都是灰色的，则生成一灰色子节点并加到U中，再对这一对节点的子节点递归地运用上述算法。对于第三种情形，运用上述算法后，要检查新节点的孩子节点。如果所有子节点是黑的，则删除它们，并在U 中把它们的父节点由灰色变成黑色。集合的求交运算的算法与之类似，只是节点的黑色与白色的作用交换一下。如果节点对中任一个是白的，则在U 中相应节点为白的。如果节点对之一是黑色的，则U 中相应节点由节点对中的另一个节点表示。如果节点对中两个都是灰色的，则生成一灰色子节点并加到U中，再对这一子节点，递归地运用上述算法。与集合并情形一样，如果两个节点都是灰色，则对新生成的灰色节点的孩子节点作上述运算后，要对孩子节点作检查。此时，若孩子节点是白色的，则删除它们，并且在U 中把它们的父节点由灰变白。

在四叉树和八叉树上容易完成简单的变换。例如，绕轴旋转90°的倍数，则对每一层的子节点依次进行同样的旋转。也可以直接进行2次幂倍数的缩放和反射变换。平移变换要稍微复杂一些，对一般的变换也同样如此。另外，与一般的空间位置枚举法一样，在一般变换下，走样问题较为严重。

2. 查找邻节点

四叉树和八叉树的一个重要运算是查找邻节点，即查找与原节点在同一层或者在下层相邻

551
|
552
的节点（共享一个面、边或者顶点等）。一个四叉树节点在八个可能方向上有邻节点。它与东、南、西、北方向上的邻节点有共享边，与西北、东北、西南、东南方向上的邻节点有共享顶点。一个八叉树节点在26个可能方向上有邻节点，有共享面的邻节点有6个，共享边的有12个，共享顶点的有8个。

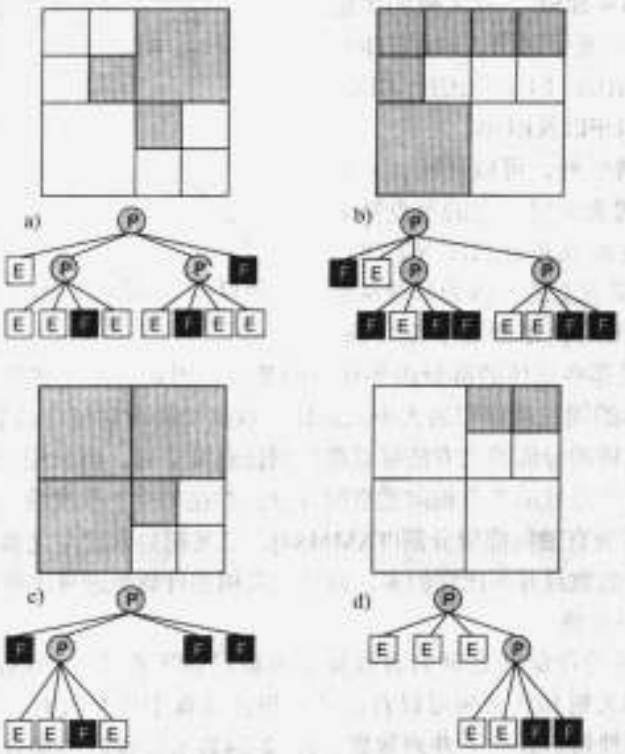


图12-24 实现四叉树的布尔集合运算。a)形体S及其四叉树，b)形体T及其四叉树，c)$S \cup T$，d)$S \cap T$

Samet[SAME89a]给出了一种查找指定方向上的邻节点的方法。从原节点开始，上溯四叉树或八叉树，直至找到原节点和邻节点的共同祖先，然后沿树向下找，直至找到所需要的邻节点。这里必须有效解决两个问题：查找共同的祖先，以及决定哪些子孙是它的邻节点。最简单的情形是沿八叉树节点的一个面（L、R、U、D、F或者B）的方向d查找邻节点。从原节点沿树上溯时，共同的祖先是第一个符合下列条件的节点：它不能从原节点d侧的孩子节点到达。例如，如果要查找它的一个L（左）邻居，则第一个共同祖先是第一个符合下列条件的节点：它不能从孩子节点LUF、LUB、LDF或者LDB到达。因为能从这些孩子节点到达的节点不能有
553
任何在原节点左侧的孩子节点（原节点的左邻居）。当共同祖先找到时，祖先的子树沿着从原节点到祖先的路径的镜像图方向遗传，镜像图是原节点到祖先的路径沿共同边界的反射得到。如果邻节点比原节点大，则只有部分反射路径是相通的。

类似地，可以沿着四叉树节点的一条边的方向来查找邻节点。例如，要查找图12-25中节点A的N邻节点，A是NW孩子节点，从A开始，沿图中黑线表示路径查找。从NW方向上溯到它的父亲，再沿NW方向上溯到它的祖父，最后从SW方向上溯到它的曾祖父，这是根节点，因为从S节点和N节点都可以上溯到这儿。然后沿着上述的镜像图路径（原路径沿N—S边界反射得到）向下找到根节点的NW孩子，最后到达这个子节点的SW孩子，这是叶节点。

Samet[SAME89a]更详细地描述了八叉树中查找邻边和邻顶点的算法，并给出了一个很好的递归实现方法，它在计算反射路径时用查找表实现。

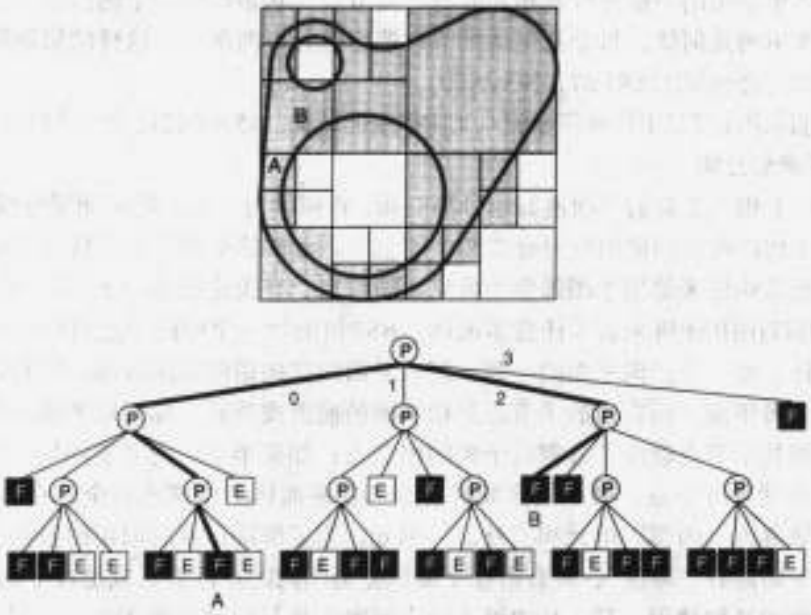


图12-25 查找四叉树节点的邻节点

3. 线性表示法

表示四叉树或八叉树的数据结构，虽然首先想到的是似乎要用指针来表示，但也可以不用指针。在线性四叉树或线性八叉树表示法中[GARG82]，每一个充满的节点表示成一个数字序列，它表示节点的所有限定的地址，数列的长度与节点的层次一致。只有黑色（即充满）的叶节点需要存储，以表示形体。不在最低层的节点在其数列后面加一个填充字符（如"X"）。例 554
如，一个线性八叉树可以利用9以内的数来进行简洁的编码（习惯上每个数字用4个二进制位来表示），从0到7表示象限，8表示附加的字符，线性八叉树的节点以已排序的顺序存储，它表示八叉树的后序遍历。例如，图12-21中的形体用线性四叉树表示为：00X，010，011，020，022，100，102，103，12X，130，132，20X，21X，220，222，223，230，231，232，3XX。

利用线性四叉树或线性八叉树表示，许多操作的实现效率高。例如，Atkinson、Gargantini和Ramanath[ATKI84]给出了一个算法，用于通过连续忽略八叉树节点序列来确定构成八叉树边界的体素。首先考虑最下层（最大体积）的充满节点。如果有这样的节点，它的六个侧面都与同一层次的充满节点相邻，则每一个这样的节点是在形体内部，而不是形体边界的一部分，因此在列表中删掉它。（每一个邻节点的代码可以通过原节点代码的简单算术运算得到。）在同一层次中的其他所有节点可能包含构成形体边界的体素；把每一个这样的节点分成八个子节点并在列表中用这八个子节点代替。按节点的层次从上到下依次重复这个算法，直到节点的大小为一个体素为止。留下来的节点就是构成形体边界的体素。

4. PM八叉树

许多学者研究了把八叉树和边界表示结合起来的混合表示方式，以保留导出形体的原边界表示的几何精度[HUNT81；QUIN82；AYAL85；CARL85；FUJI85]。PM八叉树（PM表示多边形映射）类似地可推广到四叉树情形[SAME90a]。八叉树递归分割成子节点，直到子节点只

有五种不同叶子类型。除了“充满”和“空”类型外，引入三种新的叶子类型，它们实际上是特殊的“部分充满”的节点：顶点节点，它包括形体的单个顶点以及与之相连的面和边；边节点，它包括形体单条边的一部分以及相邻的面；面节点，它是形体一个侧面的一部分。新的叶子类型限制在简单的几何体，每个几何体把节点严格地分成两部分，这样的限制简化了对形体操作的算法，如集合运算[CARL87；NAVA89]。

18.11.4节将讨论许多基于体素模型和八叉树模型的结构。15.8节将讨论八叉树可见表面算法。

12.6.4 二元空间划分树

八叉树用三个相互垂直的平面递归地分割空间，在树的每一层上把空间等分成八份。相反，二元空间划分(BSP)树递归地把空间分成两个子空间，分割的平面可以是任意朝向和任意位置的。二叉树数据结构原来是用于图形学中可见面的判定，有关论述见15.5.2节。后来，Thibault和Naylor[THIB87]用BSP树来表示任意多面体。BSP树的每一个中间节点对应一个平面，并有两个子节点指针，每一个指向平面的一侧。假如平面的法向指向形体外部，则左边的子节点是指平面的后面或者里面，而右边的子节点是指平面的前面或外面。如果对平面一侧的半空间做进一步分割，则其子节点就成了分割后子树的根节点；如果半空间内部是均匀一致的，则这一
555 半空间的子节点就是叶节点，表示的区域或者全部在多面体的内部或者全部在多面体的外部。这些均一的区域称为“内部”单元和“外部”单元。为了解决运算时限定的数值精度，每一个节点所对应的平面还有“厚度”，所有落在平面误差范围内的点都被看成是在平面上的。

与八叉树和四叉树情形一样，BSP树下的分割概念是与空间维数无关的，因此，图12-26a表示二维空间中用黑线围成的凹多边形。“内部”单元用灰色表示，定义半空间的直线用深灰色表示，法向指向外侧。相应的BSP树如图12-26b所示。在二维空间中，“内部”和“外部”区域把平面分成由凸多边形组成的格子状；在三维空间中，“内部”和“外部”区域把三维空间分成由凸多面体组成的格子状，因此BSP树可以把任意的带“洞”的凹实体表示成凸的“内部”区域的并集。与八叉树不同，但与边界表示法相同，任意的BSP树表示的实体不一定是有界的。例如，一棵三维BSP树只由一个中间节点构成，“内部”和“外部”节点是它的子节点，这棵三维BSP树所定义的形体是只有一个平面界定的半空间。

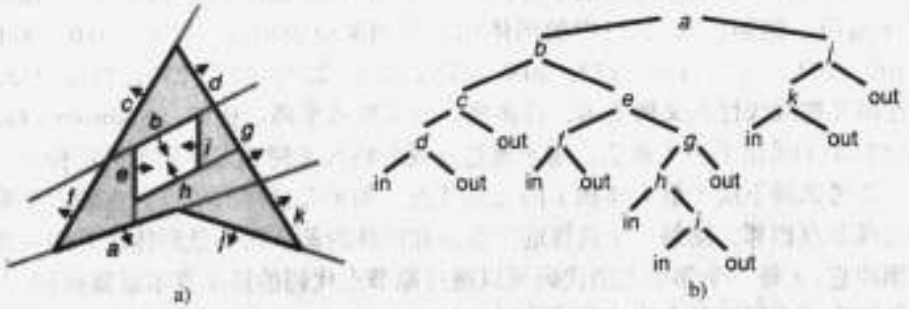


图12-26 二维空间中的BSP树表示，a)凹多边形用黑线作为边界，定义半空间的直线是深灰色的，而“内部”单元是灰色的，b)BSP树

确定一个点是否落在实体的内部、外部或上面这样的问题，称为点的分类问题[TILO80]。BSP树可用来对点进行分类，把要分类的点，从根节点开始，向下过滤。在每一个节点，把点代入节点的平面方程，如果点落在节点平面的后面（或里面），递归地通过左子节点，如果点落在平面的前面（或外面），则递归地通过右子节点。如果节点是叶子，则点给出叶子的值，或者“外部”或者“内部”。如果点落在节点平面上，则点通过两个子节点，与上一层的两个分类比

较，归到取值相同的一类中；若不同，则点位于“内部”和“外部”区域之间的边界上，分到“上面”类中。这种方法可以推广到对直线和多边形进行分类。但是与点的分类不同，直线或多边形可能部分地位于平面的两侧。所以，在每一节点处，节点平面与直线或者多边形求交，把直线或多边形分割成几个部分，各部分落在平面的前面、后面或者上面，再对各部分单独分类。

Thibault和Naylor描述了从边界表示构造BSP树的算法，为了便于对把BSP树与边界表示结合起来的混合表示进行布尔集合运算，以及确定那些位于BSP树的边界上的多边形[THIB87]。这些算法作用于符合以下条件的BSP树：它的每一个节点与一列嵌入到节点平面内的多边形相对应。利用BSP树的构建算法（将在15.5.2节中介绍）的改进形式，这些多边形可添加到树中。 556

虽然BSP树给出了一种简洁的表示，但是在树的构造中，以及进行布尔集合运算时，要对多边形进行分割，从而使得它所表示的记号比其他表示方法潜在地缺少了些紧凑性。但是利用BSP树所固有的与空间维数无关的优点，我们可以研究一种三维BSP树的闭的布尔代数，这种三维BSP树递归地需要把多边形表示成二维树，把边表示成一维树，把点表示成0维树[NAYL90]。

12.7 构造实体几何

在*构造实体几何*(CSG)中，简单的基本实体通过正则集合运算进行组合，其中包括基本实体的直接表示。形体以树的形式存储，集合运算是对中间节点和作为叶子的简单基本实体进行的（图12-27）。一些节点表示布尔算子，而另一些则表示平移、旋转和缩放等操作，很像第7章的层次结构。一般而言，布尔运算不满足交换律，所以树的边是有次序的。

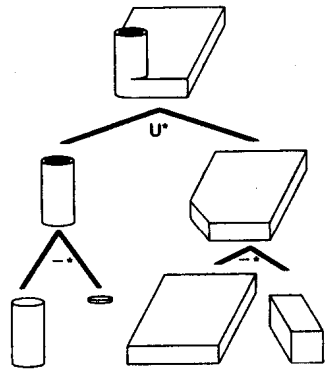


图12-27 由CSG及其树定义的形体

为了确定形体的物理性质以及便于绘图，需要把叶子的性质组合起来，以得到根节点的性质。一般的处理策略是第7章所介绍的深度优先遍历树的方法，把叶子节点沿树向上组合起来。这一步骤的复杂度依赖于叶子形体的存储表示方式，以及实际上在树根处的组合形体是否肯定能得到它的完全表示形式。例如，在实现时难以把两个边界表示的节点用正则布尔集合运算（已在12.5节讨论过）组合生成第三个用边界表示的形体。另一方面，通过处理叶子形体的表示而不是显式对它们进行组合，用15.10.3节将要介绍的CSG算法可以非常简单地生成一个类似的形体。生成类似于CSG表示的形体的其他算法包括[ATHE83；OKIN84；JANS85]；支持CSG的体系结构将在18.9.2节、18.10.2节和18.11.4节中讨论。 557

在一些实现过程中，基本实体是一些简单的实体，如立方体或球体等，以保证所有的正则组合是有效实体。在另一些系统中，基本实体包括本身并不是有界实体的半空间。例如，一个立方体可定义成六个半空间的交，一个有界圆柱体是无限长圆柱体被顶部和底部的两个平面半空间切割而成。使用半空间会出现生成实体的有效性问题，因为并不是所有的半空间的组合都可生成有效实体。但是，像用平面对形体切片这样的操作可用半空间，要不然需要利用另外一个实体的面参与操作，并且需要引入额外的上限边界，因为在作切片时，即便是只对单个切片感兴趣，也必须是整个实体参与正则布尔集合运算。

我们可以把单元分解法和空间位置枚举法看成是CSG的特殊情形，其中的惟一的操作符是

隐式的胶合运算符：两个形体的并，这两个形体可以是接触的，但内部一定不能相交（即形体间的正则布尔交集是空集）。

CSG的表示是不惟一的。如果一个系统允许用户用提拉运算来操作叶子实体，这种表示特别容易使人迷惑。对两个原来相同的形体施加同样的操作，会产生两个不同的结果，如图12-28所示。对CSG模型的编辑可以通过对子树的删除、添加、替换和修改等手段进行，而且CSG模型具有相对紧凑的存储格式，所有这些使得CSG模型成为主导的实体造型表示方法之一。

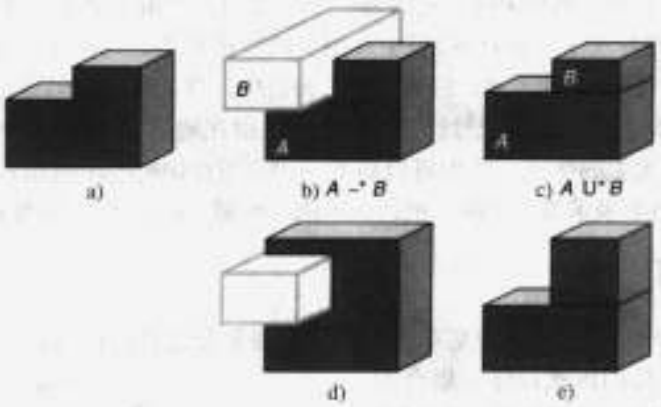


图12-28 图a)所示的形体可以用不同的CSG操作定义，如b)和c)中所示的操作。把b)和c)的两个形体的顶面向上提拉生成不同的形体，如d)和e)中所示

12.8 各种表示法的比较

这里已讨论了五种表示：基本实体举例，扫掠表示，边界表示，空间划分表示（包括单元分解法、空间位置枚举法、八叉树法和BSP树法），以及CSG表示法。我们利用12.1节所介绍的
558 准则，对它们进行比较。

- *精确性*。空间划分法和多边形边界表示法对许多形体只产生近似表示。但有一些应用，只要近似解是适当的（通常是比较粗糙的），这不算是一个缺点，如寻找机器人的行走路径。但是，要生成满意的视觉图形或者是用足够的精度交互地计算形体，就会产生很大的计算量而不实用。用第16章将要介绍的光滑消隐技术也不能对用多边形构造的人工制品产生栩栩如生的效果。因此支持高质量图形显示的系统经常用包括非多面体基本实体的CSG表示和含有曲面的边界表示。基本实体举例法也可以产生高质量的形体，但不能用集合运算来组合两个简单形体。
- *表示域*。能用基本实体举例法和扫掠法表示的形体的域是有限的。相比较而言，空间划分方法可以表示任何实体，尽管通常只是一种近似表示。利用直线边围成的多边形以及其他边和面的类型，边界表示可以用来表示很广泛的形体类型。但是很多边界表示系统仅限于简单的曲面类型和拓扑结构。例如，形体只能是二维流形的二次曲面的组合。
- *惟一性*。只有八叉树法和空间位置枚举法才能保证表示的惟一性：只用指定的大小和位置来表示形体。在八叉树的情况下，必须经过一些处理以保证表示方法是完全简化过的（也就是说，没有下面的灰色节点：其子节点都是黑的或白的）。基本实体举例法，一般不能保证表示的惟一性：例如，一个球体可以用球形的和椭球形的基本实体来表示。但是，若精心选取一组基本实体类型，惟一性是可以保证的。

- 有效性。在所有的表示方法中，边界表示最难以保证实体的有效性。不仅是顶点、线、面的数据结构可能不一致，面或边之间也可能相交。相反，BSP树表示一个有效的空间集合，但未必是有界实体。CSG树或八叉树的有效性，只需要进行简单的局部语法检查即可（如果基本实体是有界的，则CSG树表示的实体也是有界的），而空间位置枚举法的有效性则毋须任何检查。
- 封闭性。不能用基本实体举例法中的组合方法来构造基本实体，简单的扫掠在布尔运算下是不封闭的。因此在造型系统中，两者通常都不能作为内部表示方式。尽管一些特殊的边界表示在布尔运算下会碰到封闭性问题（例如，不能表示二维流形以外的边界），但这些情形通常可以避免。
- 紧凑性和效率。表示方式通常按它们是否生成“已估值的”或“未经估值的”模型分类。未经估值的模型包含了为了完成基本操作必须进一步处理（或估值）的信息，如确定形体的边界等。使用布尔运算，CSG表示生成的是未经估值的模型，因为它使用布尔运算执行的每一次计算都必须沿CSG树遍历，对表达式求值。因此，CSG模型的优点是它的紧凑性以及它记录布尔运算的能力，可以快速地做变换，并可以做快速的撤销（undo）操作，因为这些操作都记录在树的节点上。八叉树和BSP树也可以看成是“未经估值的”模型，由一系列欧拉操作生成的边界表示也可看成是“未经估值的”模型。但另一方面，如果用布尔运算生成形体，它的边界表示和空间位置枚举法通常可看成是“已估值的”模型。需要注意的是这些术语的使用都是相对的，例如，如果所做的操作是确定一点是否在形体内部，则求边界表示的值要比求CSG类似表示的值的工作量要大。 559

  如第15章所讨论的那样，产生用边界表示和CSG表示的形体的图像有许多有效的算法。虽然对大部分实体而言，空间位置枚举法和八叉树表示只是对形体进行粗略的逼近表示，但是对它们的操纵算法却比对其他表示方式的要简单得多。因此它们已用于由硬件实现的实体造型系统中，这些系统主要应用在对布尔集合运算的速度要求比生成的图像的精度要求更高的情形。

有一些系统使用多重表示方式，因为某种表示方式下的一些运算要比在其他表示方式下的运算效率高。例如，GMSOLID[BOYS82]在存储时使用CSG表示，使得存储更为紧凑，而使用边界表示，以便于对所需数据的快速查询，这些数据在CSG表示中的关系并不明确，如连通性。在GMSOLID系统中，反映形体的当前状态经常是用CSG表示的，只有当需要做进一步运算时，才更新为边界表示。更新处理在系统的后台进行。因此，用户在等待更新结果时，可以做其他的操作。除了有的系统保持两种独立表示方法，并且需要时可以相互转换外，也有些系统是混合表示的，在系统的某一细节层次上，表示方式之间可以相互转化，但系统中只有一种表示信息，其中的例子就是在12.6.3节中讨论的PM八叉树，它采用八叉树和边界表示的混合表示方式。有关多重表示和混合表示的一些问题，详见[MILL89]。

把所讨论过的所有表示方式都转化到空间位置枚举法或者八叉树表示法相对较为容易，但只能是逼近表示。这样的转换并不可逆，因为转换过程中丢失了一些信息。另外，容易把所有的表示方式都精确地转化为边界表示和PM八叉树表示。对边界表示的形体完成集合运算的算法，如12.5节所述，可以用于从 CSG到边界表示的转化：从叶子基本实体的多面体表示开始，对CSG树的每一层反复运用这一算法。Rossignac和Voelcker[ROSS89]实现了一个从CSG到边界表示的快速转换算法，这一算法确定CSG节点中称为CSG节点*活动区域*的那一部分——如果它发生改变将影响最终的实体；在执行布尔运算时，只需考虑实体在节点的活动区域范围内的那部分即可。另一方面，从边界表示到CSG的转换则要困难些，尤其是当转化成使CSG操作步数

最小的表示时。Vossler[VOSS85b]描述了一种通过自动识别简单扫掠的模式从扫掠表示到CSG表示的转换方法，这些简单扫掠体通过布尔运算构成更复杂的扫掠体，再转化成CSG表示。

正如12.1节所指出的那样，线框表示只包含顶点和边的信息，没有考虑面的信息，这种表示方法本质上是有二义性的。但是Markowsky和Wesley开发了一种算法，它可以由给定的线框表示导出所有可能的多面体[MARK80]，还开发了一种配套算法，由所生成的多面体生成给定
560 的二维投影图[WESL81]。

12.9 实体造型的用户界面

为实体造型系统开发用户界面，可为第9章所介绍界面设计技术的实用化提供了极好的机会。许多操作技术可以在图形界面上进行，包括正则布尔集合运算、提拉以及欧拉算子等的应用，在CSG系统中，允许用户通过修改或替换叶子实体或子树中的节点来编辑形体。表面间的光滑过渡可以用混合和切角两种操作实现。一个成功的系统的用户界面很大程度上依赖于系统内部表示的选择。但是基本实体举例法是个例外，因为它鼓励用户从专用参数方面考虑形体。

在第11章，我们注意到，可以用多种等价方法来表示同一条曲线。例如，画曲线系统的用户界面允许用户通过控制Hermite切向量或者指定Bezier控制点等方式输入曲线，而曲线在系统内部的存储则只是Bezier控制点。类似地，实体造型系统可以让用户用多种不同的表示形式来生成形体，而在系统内部则以另一种形式存储。与曲线表示一样，形体的每一种不同的输入表示都有一些明显的优点，使得这一方式成为生成形体的自然选择。例如，在边界表示的系统中，形体由平移扫掠或旋转扫掠方式定义。在一种表示中，用户界面也可以提供不同的方法定义同一形体。例如，定义球体有多种方法，其中的两种是这样的，指定球心以及球面上一点，或者指定球的直径的两个端点。第一种方法适合于定义球的中心点，而第二种方法则更适合于两形体之间定义球体。

形体所需要的精度通常是指定几种确定测量精度的方法，例如，通过定位设备或数据录入。由于一个形体的位置常与另一形体的位置有关，用户界面常常提供两形体间的约束方法。相关的技术是使用户具有定义网格线来约束形体的位置的能力，如8.2.1节所述。

在实体造型系统中，用户界面设计的一些最基本的问题起源于在传统的二维交互设备和显示设备上操纵和显示三维形体。这些问题的详细讨论见第8章和第9章。许多系统通过提供多个显示窗口来解决其中某些问题，它允许用户同时从不同的位置观察形体。

12.10 小结

如上所述，实体造型在CAD/CAM和图形学中都是很重要的。迄今为止，虽然形体的表示和处理手段已有了许多有用的算法和系统，但仍有许多难题需要解决。其中最重要的问题之一就是系统的健壮性。实体造型系统尤其受数值不稳定性的困扰。常用的算法需要实现比硬件更高的精度要求，以保持中间浮点运算的结果。例如，给定两个形体，其中一个形体是另一个形
561 体的副本做非常微小的变形得到的。此时，两形体之间的布尔集合运算就可能失败。

对非刚体、柔性物体、联接体的表示是必要的，对形体进行折弯和扭曲等变换的工作见第20章。许多形体的表示不能指定总的精度，更多的是，形体的形状是由带约束的参数来定义的，参数在一定的范围内取值。这些形体称为“有误差”的形体，对应于由机床和冲床加工出来的真实形体[REQU84]。新的表示方法可用来表示有误差的形体[GOSS88]。

所有已设计的实体有一个共同的、用于一些特殊场合的“特征”，如洞和切边。目前的一个研究领域是探讨自动识别形体的可能性和推断设计者的设计意图，确定每一个特征要完成什

么[PRAT84]。这一点允许检查所做的设计以保证特征按设计完成。例如，如果设计某些特征，在压力条件下给定一部分力，则可以自动确认这一功能的实现。也可以检查形体上的进一步操作，以保证不折不扣地实现这些特征。

习题

12.1 在12.4节中定义了完成∩*运算的结果，用同样的方法定义两个多面体形体间的∪*和－*运算的结果。试解释为什么运算结果要限制在正则集，并说明如何确定形体的每个面上的法向。

12.2 考虑确定一个合理的实体是否为空形体（没有体积）的任务。在每一种讨论过的表示方法中，完成这一测试有何困难？

12.3 考虑这样一个系统，其中的形体可以用扫掠表示，并且可以进行正则集合运算。在这样的系统中，对形体要做怎样的限制，才能保证系统的表示是封闭的？

12.4 实现在四叉树和八叉树上完成布尔集合运算的算法。

12.5 试解释，在四叉树和八叉树上实现集合运算时，为何不必区分普通的集合运算和在12.2节讨论的正则集合运算。

12.6 虽然作正则集合运算的几何含义是明确的，但是不清楚的是如何对待形体的性质。例如，由两个不同材料构成的形体的交集应指定哪些性质？在制造实际形体时，这一问题并不重要，但对于虚拟的图形世界，任何两种形体之间都可能有交，你认为有什么较好的解决方法？

12.7 试解释如何在图形包中要使用四叉树或八叉树来加快2D或3D拾取。

12.8 描述如何在基本实体举例法、边界表示法、空间位置枚举法以及CSG表示法中实现点的分类。

562

第13章 消色差光与彩色光

光栅图形的快速增长使得颜色和亮度成为当前计算机图形学不可缺少的部分。颜色是一个很复杂的主题，它涉及到物理学、生理学、心理学、艺术和图形设计。许多研究者对颜色的理论、度量以及标准进行了大量的、富有成效的研究。在这一章中，我们介绍与计算机图形学关系最紧密的颜色知识。

物体的颜色不仅依赖于物体本身，还依赖于照射它的光源、周围环境的颜色以及人的视觉系统。有些物体（墙、桌子、纸）反射光，有些物体（玻璃纸、玻璃）透射光。如果一个表面仅反射纯的蓝色光，那么在纯红色光的照射下，它呈黑色。类似地，如果一块玻璃仅透射纯红色光，那么如果透过它观察纯绿色光，结果是黑色。这些问题我们将稍后再讨论。我们首先讨论的是消色差光，它被用来描述黑色、灰色和白色。

13.1 消色差光

当我们观看黑白电视或黑白显示器时，我们所观察到的就是消色差光。观察消色差光不会产生红、兰、黄等感觉，光量是消色差光的惟一属性。如果从物理学中能量的角度讨论光量，可以用术语*亮度*（intensity）和*光强度*（luminance）；如果从心理学中观察到的亮度来描述，可以用术语*辉度*（brightness）。就像我们将要简要介绍的那样，这两种术语代表的含义有关系但并不相同。以一个数值来标识不同的亮度级很有用，定义0代表黑色，1代表白色，介于0、1之间的亮度级代表不同的灰色。

一个黑白电视机在单个像素处能产生多级亮度。行打印机、笔绘图仪和静电绘图仪只能产生两级：纸的白色（或亮灰色）和沉积在纸上墨水或调色剂的黑色（或黑灰色）。后文将要讨论的特定技术可以让只有两级亮度的设备产生多级亮度。

13.1.1 选择亮度值——gamma校正

如果我们要显示256种不同的亮度，那么我们应采用哪256级亮度呢？我们当然不希望128个亮度分布在0到0.1之间，另外128个亮度分布在0.9到1.0之间，因为亮度从0.1到0.9是不连续的。开始，我们也许想让亮度级在0与1之间均匀分布，但这种选择忽略了人眼的重要特性：人眼对亮度的比率而非亮度本身敏感。也就是说，我们观察到的亮度0.10与0.11之间的差别和0.50与0.55的差别是一样的。（这种非线性很容易观察到：将灯泡的功率从50瓦调到100瓦再到150瓦，你将看到从50瓦到100瓦的亮度增加比从100瓦到150瓦更大一些）。从辉度（即观察到的亮度）的角度衡量，亮度0.10与0.11之间的差别和0.50与0.55的差别是一样的。因此，为了使辉度均匀分布，亮度级应该呈对数分布而非线性分布。

为了在最低亮度I_0与最高亮度1.0之间寻找256级亮度，每一级亮度是它前一级亮度的r倍，我们使用如下关系式：

$$I_0 = I_0,\ I_1 = rI_0,\ I_2 = rI_1 = r^2I_0,\ I_3 = rI_2 = r^3I_0,\ \cdots,\ I_{255} = r^{255}I_0 = 1 \qquad (13\text{-}1)$$

因此，

$$r = (1/I_0)^{1/255},\ I_j = r^jI_0 = (1/I_0)^{j/255}\ I_0 = I_0^{(255-j)/255} \qquad \text{for } 0 \le j \le 255 \qquad (13\text{-}2)$$

一般地，对$n+1$级亮度，

$$r = (1/I_0)^{1/n},\ I_j = I_0^{(n-j)/n} \qquad \text{for } 0 \le j \le n \tag{13-3}$$

如果亮度只有4级($n = 3$)，I_0为1/8（为了说明问题，取的值比实际可能的值要大的多），$r = 2$，式(13-3)告诉我们相应的亮度级是1/8，1/4，1/2和1。

一个CRT可能达到的最小亮度值I_0介于最大亮度值的1/200到1/40之间，因此，I_0的典型的值在0.005与0.025之间。最小亮度值不是0，因为CRT的磷涂层是发光的。最大亮度值与最小亮度值之间的比值称为变化范围。一个特定CRT的变化范围的准确值可以这样来获得：在一个黑色的区域上显示一个白色的方块，然后用光度计测量这两个亮度。测量必须在完全的黑屋子里进行，避免环境光影响测量结果。取I_0的值为0.02，相应的变化范围为50，根据式(13-2)得到$r = 1.0154595\cdots$；并且根据式(13-1)256个亮度中的前几个与后几个分别为0.0200, 0.0203,
564 0.0206, 0.0209, 0.0213, 0.0216, ⋯, 0.9848, 1.0000。

因为CRT和胶片的非线性，在CRT上显示由式(13-1)定义的亮度不是一个容易的过程，将它们记录在胶片上更加困难。例如，对恒定的k和γ，磷涂层发出的亮度与电子束所含电子个数N的关系如下：

$$I = kN^{\gamma} \tag{13-4}$$

大多数CRT的γ值介于2.2和2.5之间。电子个数N与控制栅的电压成正比，从而也与像素的指定值V成正比。因此，对另外一个常数K，

$$I = KV^{\gamma},\ \text{即}\ V = (I/K)^{1/\gamma} \tag{13-5}$$

现在，给定一个想要的亮度I，我们首先通过检索可用的亮度表（由式(13-1)计算得到）或其他类似的表确定最接近的亮度I_j，

$$j = ROUND(\log_r(I/I_0)) \tag{13-6}$$

找到j之后，我们计算

$$I_j = r^j I_0 \tag{13-7}$$

下一个步骤是利用式(13-5)计算亮度值I_j所对应的像素值V_j：

$$V_j = ROUND\ ((I_j/K)^{1/\gamma}) \tag{13-8}$$

如果光栅显示器没有查色表，那么，V_j被放在相应的像素处；如果有查色表，那么，j被放在像素处，而V_j被放在查色表的第j项。

K, γ 和I_0的值依赖于所用的CRT，因而，在实际应用中，查色表的加载采用一种基于实际测量到的亮度的方法[CATM79，COWA83，HALL89]。在这种一般意义上使用查色表称为gamma校正，该命名从式(13-4)的指数部分得到。如果显示器有硬件gamma校正，那么，I_j而不是V_j被放在刷新缓存或查色表中。

如果不用基于比率的亮度值和gamma校正，量化误差（用可显示的亮度值逼近一个实际的亮度值）在黑色附近比在白色附近更显著。例如，采用4位、16种亮度，最大的四舍五入量化误差是1/32 = 0.031，这个值是亮度值1/16的50%，但仅仅是亮度值1.0的3%。采用基于比率的亮度和gamma校正，最大量化误差占辉度（观察到的亮度）的百分比是恒定的。

一个自然的问题是“多少个亮度级是足够的？”“足够”指的是数量多的足以显示连续色调的黑白图像。当比率r等于或小于1.01时能达到这种效果（低于这个比率，人眼不能区分亮度I_j与I_{j+1}）[WYSZ82，p.569]。因此，合适的亮度级个数可以通过将$r = 1.01$代如式(13-3)得到：

565 $$r = (1/I_0)^{1/n} \qquad \text{or} \qquad 1.01 = (1/I_0)^{1/n} \tag{13-9}$$

解上面的公式得到n为

$$n = \log_{1.01}(1/I_0) \tag{13-10}$$

其中，$1/I_0$是设备的变化范围。

几种显示媒体的变化范围$1/I_0$，以及为了保持$r = 1.01$同时充分利用变化范围的亮度级个数n在表13-1中列出，这些都是理论值，前提是具有完美的再现过程。实际上，再现过程中墨水的喷溅和小的随机噪声会大大降低打印媒体的n值。例如，图13-1显示了一个连续色调的照片，后续的5幅图13-2到图13-6用4、8、16、32个亮度级重新表示它。用4、8个亮度级时，从一个亮度级到下一个亮度级的变化或轮廓非常明显，因为相继亮度级之间的比率r比理想值1.01大出很多。用32个亮度级时，轮廓仅仅能被观察到；而对这个特定的图像，用64个亮度级时，轮廓完全消失。这表明，在纸上打印连续色调的黑白图像（就如本书中的图像）需要的最少亮度级个数是64。然而，对一个处于完全的黑屋子中经过很好调节的CRT，更大的变化范围意味着需要更多的亮度级。

表13-1 变化范围$(1/I_0)$与几种显示媒体所需要的亮度级数量$n = \log_{1.01}(1/I_0)$

显示媒体	典型变化范围	亮度级个数n
CRT	50~200	400~530
像纸	100	465
幻灯片	1000	700
黑白打印方式下的纸	100	465
彩色打印方式下的纸	50	400
黑白打印方式下的报纸	10	234



图13-1 一个连续色调的照片



图13-2 用4个亮度级表示连续色调的照片。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）



图13-3 用8个亮度级表示连续色调的照片。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）



图13-4 用16个亮度级表示连续色调的照片。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）



图13-5 用32个亮度级表示连续色调的照片。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）



图13-6 用64个亮度级表示连续色调的照片，与图13-5的差别非常小。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）

566
-
567

13.1.2 半色调逼近

许多显示器和硬拷贝设备是单色的，它们只能产生两个亮度级，即使是每像素具有2位或3位的光栅显示器，它所产生的亮度级个数也远少于我们的期望值。我们怎样扩展可用亮度的范围呢？答案在于我们眼睛的空间综合能力。如果我们从足够远的距离观察一个很小的区域时，我们的眼睛对区域内的细节进行平均，仅记录区域的总体亮度。



图13-7 放大的半色调模式，点的尺寸与原照片亮度的变化趋势相反。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）

这种现象被用于报纸、杂志、书上打印黑白照片，这个技术称为*半色调*（在计算机图形学中也称为*聚点有序抖动*[⊖]）。每一个小的分辨率单位内部由一个黑色的圆形区域填充，该圆形区域的面积与原照片的黑度$1-I$（I是亮度）成正比。图13-7显示了被放大了很多的半色调模式的一部分。注意，这里用水平（称为*屏幕角度*）模式构造倾斜45°的模式。用于报纸的半色调技术每英寸采用60~80个可变尺寸、可变形状的区域[ULIC 87]，而用于杂志和书的半色调技术每英寸采用110~200个区域。

图形输出设备能够模拟上述半色调技术中的面积可变的圆形区域。例如，具有两个亮度级的显示器上的一个2×2的像素区域可以用来产生5级亮度，不过这是以降低一倍空间分辨率为代价的。图13-8中的模式可以用来填充2×2的像素区域，其中处于“开”状态的像素个数与需要的亮度成正比。图13-9中的数字化的脸部图像的大小是351×351，它是以2×2模式显示的。

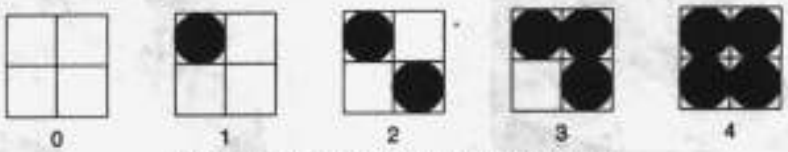


图13-8 用2×2的抖动模式模拟5个亮度级

⊖ “有序抖动”与“随即抖动”相对应，后者是一种不常用的技术。

一个$n \times n$的两级像素区域能产生n^2+1个亮度级。一般地，空间分辨率与亮度分辨率之间有一个折衷。采用3×3的模式将空间分辨率降低为原来的1/3，但它提供了10个亮度级。当然，这个折衷是受到我们视敏度（在正常的光照条件下，大约1弧分）、观察图像的距离以及图形设备的分辨率（每英寸的点数）的限制。

一个可行的3×3模式如图13-10所示，注意，这些模式可以用以下抖动矩阵表示：

$$\begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{bmatrix} \tag{13-11}$$

为了显示亮度I，我们打开所有小于或等于I的像素。



图13-9 一个连续色调的照片，数字化成351×351大小的图像，用图13-8中2×2模式显示。（卡耐基梅隆大学计算机图形学研究室Alan Paeth授权。）

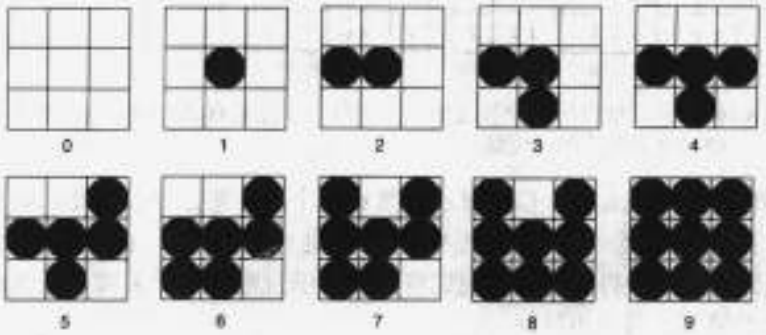


图13-10 用3×3抖动模式模拟10个亮度级

用$n \times n$像素模式逼近半色调时，必须注意不要在具有恒定亮度值的区域引入人工痕迹。例如，如果采用图13-11而不是图13-10中的模式，水平线将会出现在图像内部任何亮度值为3的大区域中。再者，模式必须按照递增的顺序安排，使得在亮度级j时处于开状态的像素，在亮度级为$k>j$时也处于开的状态。这将减小相继亮度级之间的差别，从而使轮廓效果降至最小。第三，模式必须是从中心向外扩展的，产生一种不断增大的效果。第四，对硬拷贝设备，如激光打印机、胶片记录器，它们很难产生孤立的处于“开”状态的像素，对一个特定的亮度来说，要求所有处于“开”状态的像素必须相邻，像图13-12中所示的模式是不能接受的。这就是“聚点有序抖动”中术语聚点的含义。Holladay[HOLL80]开发了一种通用的用于聚点有序抖动的抖动矩阵的定义方法。图像显示质量要求高时，n必须在8到10之间，理论上，它允许有65到101个亮度级。为了获得与每英寸150个点的显示屏幕差不多的效果，就要求分辨率介于每英寸150×8=1200个点到150×10=1500个点之间。高质量的胶片记录器能够达到这样的分辨率，但实际上，它并不能显示所有的亮度级，因为由单个黑白像素的构成模式会消失。

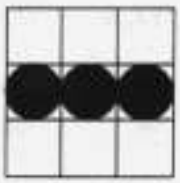


图13-11 不适合半色调的3×3抖动模式

半色调逼近并不仅限于具有两级亮度的显示器。考虑每个像素具有2位、有4个亮度级的显

示器，如果我们采用2×2模式，那么每个模式一共有4个像素，其中的每一个像素除了黑色外还可以取三个值，这就允许我们显示4×3+1=13个亮度。在这种情况下，一种可能的模式如图13-13所示，它们按照递增顺序排列。

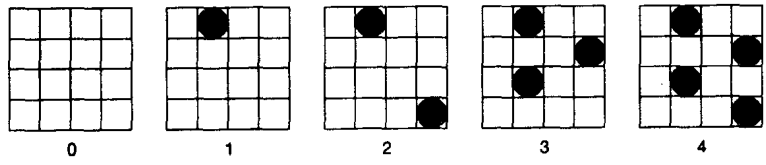


图13-12 4×4有序抖动点模式，其中的几个模式包含了单个的不相邻的点。这种分离的模式不适合于激光打印机和印刷机的半色调处理

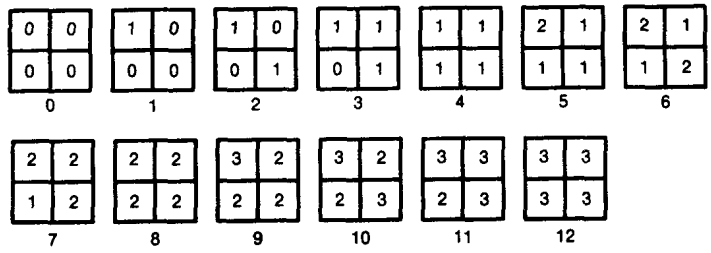


图13-13 以2×2抖动模式模拟的13个亮度级，其中每一个像素有2位（4个亮度级），单个像素的亮度和即为每个模式的亮度级

与激光打印机不同的是，一个CRT显示器显示单个点非常方便，因此，前文讨论过的对模式的聚合性要求可以放宽，并且散点有序抖动也可以采用。有很多可能的抖动矩阵：Bayer[BAYE73]开发了一些抖动矩阵，它们将图像显示过程中由于采用半色调而引入的纹理降到最小。对2×2的情况，抖动矩阵$D^{(2)}$为

$$D^{(2)} = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} \tag{13-12}$$

该式表示图13-8中的一组模式。

利用递归关系，可以从$D^{(n)}$计算出更大的抖动矩阵$D^{(2n)}$。假定$U^{(n)}$定义为全部由1组成$n \times n$矩阵，也就是：

$$U^{(n)} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & . & . \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \tag{13-13}$$

递归关系是

$$D^{(n)} = \begin{bmatrix} 4D^{(n/2)} + D_{00}^{(2)}U^{(n/2)} & 4D^{(n/2)} + D_{01}^{(2)}U^{(n/2)} \\ 4D^{(n/2)} + D_{10}^{(2)}U^{(n/2)} & 4D^{(n/2)} + D_{11}^{(2)}U^{(n/2)} \end{bmatrix} \tag{13-14}$$

将这个关系应用于$D^{(2)}$得到：

$$D^{(4)} = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} \tag{13-15}$$

568 ~ 571

以上介绍的技术假设待显示的图像远小于显示设备的像素阵列，从而可以用多个显示像素表示

一个图像像素。如果图像与显示设备的像素阵列大小一样怎么办呢？可以对有序抖动（聚点或者散点）进行一个简单的调整以适应这种情况，使得是否加亮像素(x, y)依赖于该点期望亮度值$S(x, y)$和抖动矩阵。为了显示(x, y)点，我们计算

$$\begin{aligned} i &= x \bmod n \\ j &= y \bmod n \end{aligned} \tag{13-16}$$

然后，如果

$$S(x, y) > D_{ij}^{(n)} \tag{13-17}$$

则加亮(x, y)；否则，不加亮。也就是说，图像阵列中的一个像素控制显示器上的一个像素。注意，图像中具有恒定亮度的一大块区域的显示结果与图像尺寸小于显示阵列时是一样的，因此，只有在亮度发生变化的区域中，图像阵列与显示阵列大小相等所产生的影响才是明显的。

图13-14a是一张大小为512 × 512的数字化脸部图像，采用$D^{(8)}$将它在512 × 512的具有两级亮度的显示器上显示出来。比较一下这个只有两级亮度的结果与本节前面出现的具有多级亮度的图像。采用有序抖动技术显示的更多图像请参见[JARV76a；JARV76b；ULIC87]，其中也给出了在具有两级亮度的显示设备上显示连续色调图像的更多方法。



a) b)

图13-14 采用下面两种方法重新生成的具有连续色调的图像：a)$D^{(8)}$有序抖动，b)Floyd-Steinberg误差扩散。（滑铁卢大学计算机图形学研究室Alan Paeth授权。）

由Floyd和Steinberg[FLOY75]开发的误差扩散技术是处理上述情况（即当图像与显示阵列大小相等时）的另一种方法，观察结果常常是令人满意的。将误差（像素的准确值与实际显示的逼近值之差）按如下方式加到图像阵列中位于当前像素之右和之下的4个像素值上：误差的7/16加到右端像素，误差的3/16加到左下端像素，误差的5/16加到下端像素，误差的1/16加到右下端像素。这样做的效果是使得误差扩散或混合到图像阵列的多个像素上，图13-14b便是采用这种方法创建的。 572

给定一个待显示的的图像S，要将其在亮度矩阵I中显示出来，S中被修改的值以及I中的显示值按照扫描线的顺序，从最上端的扫描线往下逐行计算。

```
double error;
K = Approximate (S[x][y]);      /* 计算与S最接近的可显示亮度值 */
I[x][y] = K;                    /* 显示像素（x, y）*/
error = S[x][y] - K;            /* 误差 */

/* 步骤1：将误差的7/16扩散到右端像素(x + 1, y)上 */
S[x + 1][y] += 7 * error/16;

/* 步骤2：将误差的3/16扩散到左下端像素上 */
```

```
S[x - 1][y - 1] += 3 * error/16;

/* 步骤3：将误差的5/16扩散到下端像素上 */
S[x][y - 1] += 5 * error/16;

/* 步骤4：将误差的1/16扩散到右下端像素上 */
S[x + 1][y - 1] += error/16;
```

为了避免将人工痕迹引入所显示的图像，我们必须保证4个误差的和等于error，不允许存在舍入误差。可以这样来做到这一点：将第4步的误差设为error减去前三步的误差。函数*Approximate*返回与实际像素亮度值最接近的可显示亮度值。对一个具有两级亮度的显示器来说，*S*的值简单地舍入成0或者1。

如果进一步采用从左向右和从右向左的扫描，可以得到更好的结果。对于从右向左的扫描，第1、2和4步中的从左向右方向上的误差计算反向即可。更详细的讨论和其他误差扩散方法请参见[ULIC87]，其他方法参见[KNUT87]。

假定图像阵列的尺寸小于显示阵列的尺寸，图像与显示器的亮度数相等，但我们想以显示阵列的尺寸显示图像。一个简单的情况是每个像素8位的512 × 512的图像和每个像素8位的1024 × 1024的显示器，如果我们简单地在水平方向上和竖直方向上复制图像像素，被复制像素所形成的方块将会很明显。为了避免这个问题，我们可以通过插值来计算像素的亮度。例如，如果图像*S*将以2倍的分辨率显示，那么显示的亮度值（$x = 2x'$，$y = 2y'$）为

$$I[x][y] = S[x'][y'];$$

$$I[x+1][y] = \tfrac{1}{2}(S[x'][y'] + S[x'+1][y']);$$

$$I[x][y+1] = \tfrac{1}{2}(S[x'][y'] + S[x'][y'+1]);$$

$$I[x+1][y+1] = \tfrac{1}{4}(S[x'][y'] + S[x'+1][y'] + S[x'][y'+1] + S[x'+1][y'+1]);$$

参见17.4节有关对图像进行两遍缩放变换的讨论，以及3.17节所描述的可用于该问题的滤波和
573 图像重建技术。

13.2 彩色

由彩色光刺激产生的视觉感受要比消色差光丰富得多，讨论彩色通常涉及三个量，它们称为色调（hue）、饱和度（saturation）与明度（lightness）。色调区别不同的颜色，如红色、绿色、紫色和黄色。饱和度指一种颜色距等亮度的有多远。红色的饱和度高，粉红色的饱和度相对较低。品蓝的饱和度高，天蓝的饱和度相对较低。轻淡的颜色饱和度相对较低，低饱和度的颜色比鲜艳的颜色（饱和度高的颜色）包含了更多的白光。明度在非彩色的含义上体现了所观察到的一个反射体的亮度。第四个术语*辉度*（brightness），用来替代明度指观察到的自发光（即发光而不是反光）体的亮度，如灯泡、太阳或CRT。

如果我们要在计算机图形学中精确地应用颜色，就有必要规定和度量颜色。对反射光，我们通过将未知颜色与一个“标准”的样品颜色集合比较而进行度量。未知颜色与样品颜色必须在标准的光源照射下观察比较，因为观察到的表面颜色同时依赖于表面本身和用于照明的光源。被广泛应用的Munsell颜色排序系统包含了若干组公布了的标准颜色[MUNS76]，这些颜色在三维空间中按照色调、亮度（也就是我们定义的明度）和色浓度（饱和度）来组织。每一个颜色都有名字，它们在颜色空间中是这样排列的：相邻颜色之间的距离相等（由大量的观察者判断

得出）。[KELL76]中有关于标准样品颜色、Munsell颜色空间的图表和颜色名称表的进一步讨论。

在打印工业和图形设计领域，颜色一般是通过匹配已打印出的颜色样品来指定的。彩图I-33便是取自于一个被广泛应用的颜色匹配系统。

画家经常通过高饱和度的或纯色颜料的色浓、色深和色调来区分颜色。向纯色颜料中加白色颜料导致色浓（tint）改变，即饱和度降低。向纯色颜料中加黑色颜料导致色深（shade）改变，即明度降低。向纯色颜料中同时加黑色颜料和白色颜料导致色调（tone）改变。所有这些步骤所产生的颜色的色调相同，它们仅仅改变了颜色的饱和度与明度。仅混合白色颜料与黑色颜料产生灰色。图13-15表示了色浓、色深和色调之间的关系。匹配一个颜色所需的混合颜料的百分比可以用作一种颜色的规范。Ostwald[OSTW31]颜色排序系统与画家的模型相似。

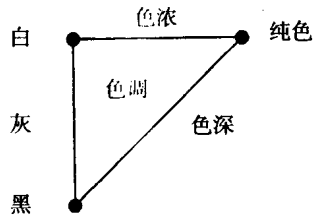


图13-15 色浓、色调与色深

574

13.2.1 心理物理学

Munsell和画家的颜色混合方法是主观的：它们依赖于观察者的判断、光照条件、样品的尺寸、周围的颜色以及环境的明度。寻找一种客观的、定量的方法是必要的，为此，我们转向物理学的一个分支：色度学。色度学中的重要术语有：主波长、色纯度和光强度。

当我们观察一束光时，主波长指的是我们所看到的颜色的波长，它与感知意义上的色调[⊖]相对应。色纯度对应颜色的饱和度，光强度对应一束光量的多少或光强。一束彩色光的色纯度为纯色光与白色光的比例，纯色光的波长是该彩色光的主波长。完全的纯色光的饱和度是100%，不包含白色光；而纯色光与白色光的混合光的饱和度介于0%与100%之间。白色光和灰色光的饱和度是0%，不含有任何具有主波长的颜色。感知上的术语与色度学术语的对应关系如下：

感知术语	色度学术语
色调	主波长
饱和度	色纯度
明度（反射体）	光强度
辉度（发光体）	光强度

本质上，光是波长介于400 nm到700 nm的电磁波，它们产生的颜色按顺序为靛、蓝、绿、黄、橙和红。彩图I-34显示了这些光谱的颜色。在每个波长处光的能量由光谱能量分布$P(\lambda)$表示，如图13-16和彩图I-34所示。光谱能量分布包含了无数个数值，每一个数值对应一个可见光谱的波长（实际上，光谱能量分布由光谱中大量的采样点构成，而采样点由分光辐射谱仪度量）。幸运的是，我们可以通过一个三元组（主波长，色纯度，光强度）更简洁地描述光谱分布的视觉效果。这就意味着，许多不同的光谱能量分布会产生同样的颜色：它们看起来一样。因此，光谱分布与颜色之间的关系是多对一的。看起来颜色一样的两个光谱能量分布称为条件等色。

575

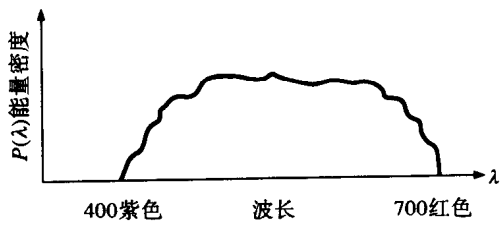


图13-16 一束光的典型光谱能量分布$P(\lambda)$

⊖ 我们后面将看到，有些颜色（如紫色）没有真正的主波长。

图13-17显示了对应一个特定颜色的无数个光谱分布（条件等色）中的一个，在主波长处，能量峰值达到e_2，而白色光均匀分布，能量为e_1。色纯度取决于e_1和e_2之间的关系：当$e_1 = e_2$时，色纯度为0%；当$e_1 = 0$时，色纯度为100%。辉度正比于光谱曲线和光效率函数（在后面讨论）乘积的积分，同时依赖于e_1和e_2。一般地，光谱分布比上面所显示的更复杂，仅仅通过观察光谱分布不可能确定它的主波长。在特殊情况下，主波长可能不是光谱分布中能量最大的分量。

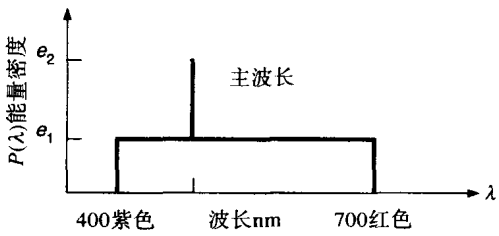


图13-17 光谱能量分布$P(\lambda)$，显示了主波长、色纯度和光强度

以上讨论与彩色CRT上红、绿、蓝三色荧光点有什么关系呢？它又与颜色的*三色激励理论*有什么关系呢？三色激励理论基于这样一种假设，即视网膜中有三种颜色感受体（称为视锥体），它们分别对红、绿、蓝三种颜色最敏感。基于这个假设的试验产生的光谱响应函数如图13-18所示。对蓝色的响应峰值在440 nm左右，对绿色的响应峰值在545 nm左右，对红色的响应峰值在580 nm左右（名词“红”和“绿”在这里可能会令人产生误会，因为545 nm和580 nm波长的峰值事实上落在黄色的范围）。曲线表明，人眼对蓝色的响应比它对红色和绿色的响应弱。

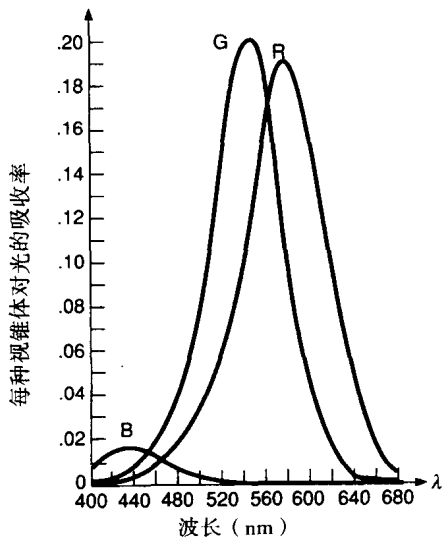


图13-18 人的视网膜中三种锥体的光谱响应函数

图13-19显示了*光效率函数*：随着主波长的变化，人眼对具有恒定光强度光的响应峰值落在波长550 nm左右（黄绿光）。实验证明，这条曲线是图13-18中三条曲线之和。

直观上，三色激励理论具有很大的吸引力，因为它大致与人们关于颜色的一种看法相对应，即一种颜色可以表示为红、绿、蓝（即所谓的基色）正的加权和。这个看法基本上是正确的：图13-20中的三个颜色匹配函数显示了典型观察者的观察结果，即对可见光谱内的所有主波长，匹配一种具有恒定亮度的颜色所需要的红、绿、蓝三种颜色光的量。

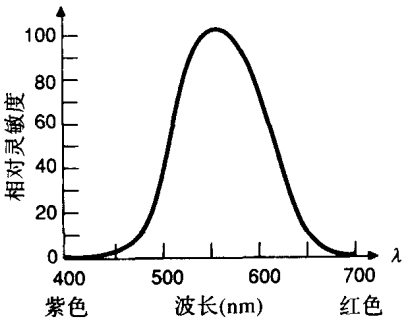


图13-19 人眼的光效率函数

图13-20中的负值说明，我们不能仅仅通过将基色相加匹配颜色，但是如果将一种基色加到颜色样品中，那么所得到的结果就可以用另外两种基色的混合色进行匹配。因此，图13-20中的负值表明要将基色加到待匹配的颜色中去。颜色匹配时必须取负值并不意味着以下结论是不正确的：将红、绿、蓝混合可以得到其他颜色。相反，用红、绿、蓝混合可以得到大范围内的颜色，否则，彩色CRT就不能正常工作了。但是，它确实说明一些颜色不能够通过红、绿、蓝混合得到，不能在普通的CRT上显示。

576
|
577

通过将不同的颜色排列在一起，由观察者判断这些颜色是相同的还是不同的，得到结论：人眼能够区分成千上万种颜色。如图13-21所示，如果两种颜色仅仅是色调不同，那么两个刚好可以区分的颜色波长的差别在光谱的两端超过10 nm，而在480 nm（蓝）和580 nm（黄）左右波长的差别小于2 nm。除了在光谱的两端，大多数不同色调之间的波长差别在4 nm以内。从而，一共可以区分的满饱和度的色调大约为128种。

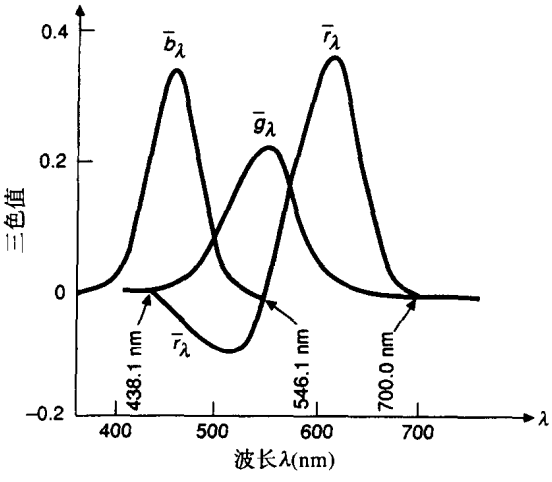


图13-20 颜色匹配函数，显示了匹配可见光谱中所有波长的光所需要的三基色的量

人眼对饱和度低的光的色调变化敏感度低，这并不奇怪，因为当饱和度趋于0时，所有的色调都趋于白色。对于固定色调和明度，人眼对饱和度变化的敏感度在可见光谱的两端更大，可区分23个级别；而在波长575 nm左右，仅能区分16个级别[JONE26]。

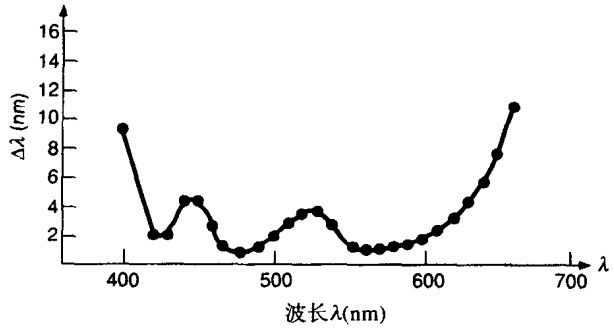


图13-21 刚好可区分的颜色之差关于λ的函数，纵坐标表示区分相邻颜色样品需要的最小波长差（来源于[BEDF58]中的数据）

13.2.2 CIE色度图

用三个固定基色的混合来匹配以至定义颜色是我们所希望的指定颜色的方法，然而图13-20指出，混合时需要负的权值，这很不方便。1931年，国际照明委员会(CIE)定义了三个标准基色，称为**X**，**Y**和**Z**，用以在颜色匹配过程中取代红、绿、蓝。三个相应的颜色匹配函数$\bar{x}_\lambda$，$\bar{y}_\lambda$和$\bar{z}_\lambda$如图13-22所示。采用这三种基色，只用正的权值就可以匹配我们能看见的所有颜色。基色**Y**有意识地这样来定义，使得它的匹配函数与图13-19中的光效率函数正好一致。注意，就像图13-20中的曲线不是红、绿、蓝的光谱分布一样，$\bar{x}_\lambda$，$\bar{y}_\lambda$和$\bar{z}_\lambda$也不是颜色**X**，**Y**和**Z**的光谱分布。它们仅仅是一个辅助函数，用来

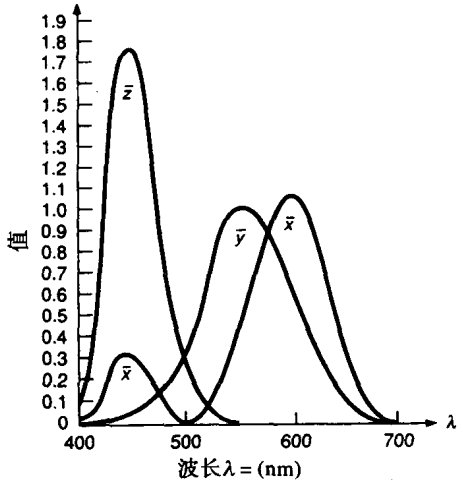


图13-22 1931年CIE三基色**X**、**Y**和**Z**的颜色匹配函数$\bar{x}_\lambda$，$\bar{y}_\lambda$和$\bar{z}_\lambda$

计算需要多少**X**，**Y**和**Z**才能匹配给定的一种颜色。

578
↓
579

颜色匹配函数由一张间隔为1 nm的表定义，它在文献[WYSZ82；BILL81]中可以查到，这些文献中也定义了基于视网膜2°视场的颜色样品的匹配值分布。1931年的表主要用于与计算机图形学相关的领域，后来1964年的基于10°视场的表并不常用，因为它的重点不是图形学中常用的颜色，而是更广泛的颜色域。

这三个CIE匹配函数是图13-20中匹配函数的线性组合，这就是说，由红、绿、蓝定义的颜色可以通过一个线性变换转换成由CIE基色表示，反之亦然。

匹配一个光谱能量分布是$P(\lambda)$的颜色需要的基色**X**、**Y**、**Z**的量为：

$$X = k\int P(\lambda)\,\bar{x}_\lambda\,d\lambda,\quad Y = k\int P(\lambda)\,\bar{y}_\lambda\,d\lambda,\quad Z = k\int P(\lambda)\,\bar{z}_\lambda\,d\lambda \tag{13-18}$$

对自发光体（如CRT）k是680流明/瓦，对反射体，通常选择k值使得白光的Y值为100，从而其他颜色的Y值介于0到100之间。因此，

$$k = \frac{100}{\int P_w(\lambda)\bar{y}_\lambda\,d\lambda} \tag{13-19}$$

其中，$P_w(\lambda)$是任意作为标准白光光源的光谱能量分布。事实上，这个积分是通过求和来计算的，因为能量分布不是由分析表达式表示的。

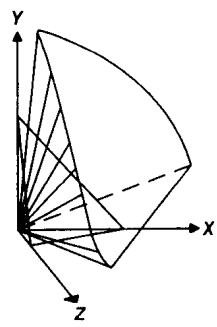


图13-23 CIE颜色空间中可见光形成一个锥体，它从坐标原点向外延伸。其中显示了$X+Y+Z=1$平面。（Cornell大学计算机图形学组，Gary Meyer授权。）

图13-23显示了XYZ空间中包含所有可见光的锥体，该锥体从坐标原点向正象限延伸，终止于一条光滑曲线。

假设(X, Y, Z)是匹配颜色**C**需要的CIE三基色的权，如公式(13-18)所示，那么，$\mathbf{C} = X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z}$。我们通过规范化（除以$X+Y+Z$）定义色度值（仅依赖于主波长和饱和度，与光的能
580
量多少无关），而$X+Y+Z$可看成总的光能：

$$x = \frac{X}{(X+Y+Z)},\ y = \frac{Y}{(X+Y+Z)},\ z = \frac{Z}{(X+Y+Z)} \tag{13-20}$$

注意，$x+y+z=1$，也就是说，x、y、z落在图13-23中的平面$(X+Y+Z=1)$之上。彩图II-1显示了CIE空间和其一部分$X+Y+Z=1$平面，同时也显示了该平面在(X, Y)平面上的正交投影。这个投影即是CIE色度图。

如果我们指定了x和y，那么$z=1-x-y$。但是，从x和y不能够恢复X，Y和Z。为此，我们还需要一个值，通常取Y，它给出了亮度信息。给定(x, y, Y)，从它到(X, Y, Z)的变换为

$$X = \frac{x}{y}Y,\quad Y = Y,\ Z = \frac{1-x-y}{y}Y \tag{13-21}$$

色度值仅依赖于主波长和饱和度，而与光能多少无关。通过绘制所有可见光的x和y，我们得到CIE色度图，如图13-24所示，它是图13-23平面中的平面$X+Y+Z=1$在(X, Y)平面上的投影。马蹄形区域的内部和边界表示所有可见光的色度值。（色度值相同但亮度不同的颜色对应区域中的同一个点。）光谱上100%纯色落在区域的曲线边界上。用来近似太阳光的标准白光由光源发光物C定义，它在区域的中心点标识出来。它的附近某一点（但不是C本身）的色度坐标$x=y=z=1/3$。发光物C的光谱分布近似于色温为6774° K的日光。

CIE色度图在很多方面都很有用。首先，通过用CIE三个基色匹配一种颜色，我们可以度量它的主波长和色纯度。（度量三色激励值X、Y和Z的仪器称为色度计，利用它和式(13-20)可以计算出色度坐标。而分光辐射谱仪可以同时度量光谱能量分布和三色激励值。）现在，假定被匹配的颜色在图13-25中的A点。当两种颜色混合在一起时，在色度图上新产生的颜色位于连接这两种颜色的直线段上。因此，颜色A可看成标准白光（发光物C）和纯色光在B点的混合；从而，B定义了颜色A的主波长，AC与BC的长度之比（表示成百分比的形式）是颜色A的色纯度。A与C越靠近，它包含的白光越多，色纯度越低。

色度图中不包含光强度，从而与光强度相关的色感也被排除在外了。例如，棕色介于橙色和红色之间，但与周围颜色相比，它的光强度非常低，在色度图上没有被显示出来。因此，记住这一点很重要，即色度图不是一个完整的调色板。（X, Y, Z）空间中有无数张平面，它们的投影都是色度图，在投影过程中它们失去了光强度信息。每一个这样的平面中包含了不同的颜色。

互为补色的两种颜色的混合可以产生白光（如图13-25中的D和E）。有些颜色（如图13-25中的F）不能由主波长定义，从而称它们为非光谱的。在这种情况下，连接F和C的直线与色度图的马蹄形曲线部分相交于B点，F的主波长就记为其补色主波长后附加c（这里大约是555 nmc），色纯度仍然定义为长度之比（这里是CF比CG）。必须由其补色的主波长的补进行描述的颜色是紫色和洋红色，它们处于色度图的下半部分。补色的主波长模型仍然可以用图13-17表示，如果我们从一个能量均匀的光谱分布中删除一部分频率为B的光，观察结果颜色即为F。

色度图的另一个应用是颜色域，它表示了将颜色加在一起的效果。通过调整混合比例，任意两种颜色（比如图13-26中的I和J），加在一起能够产生它们连线上的所有颜色。同样通过调整混合比例，第三种颜色K（见图13-26）与和I、J一起能产生三角形IJK颜色域的所有颜

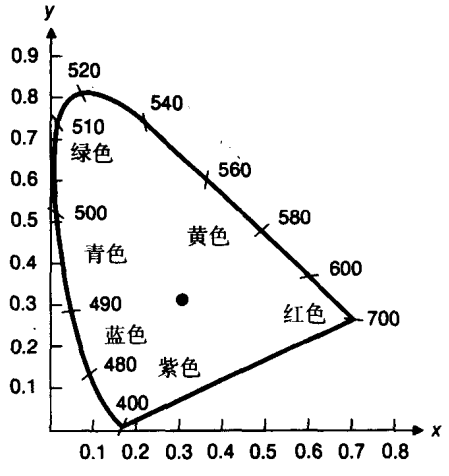


图13-24 CIE色度图。边界上波长的单位是纳米(nm)，黑点标识了发光物C的位置

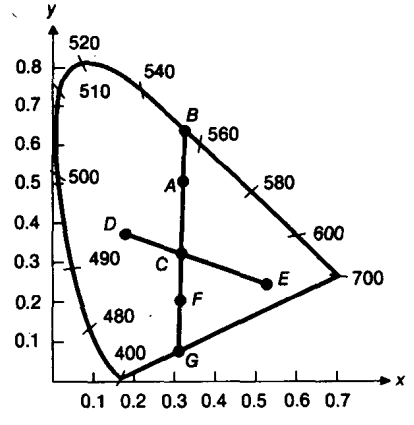


图13-25 色度图中的颜色。颜色A的主波长即为颜色B的主波长。颜色D和E互为补色。颜色F的主波长定义为颜色A的主波长的补

581
~
582

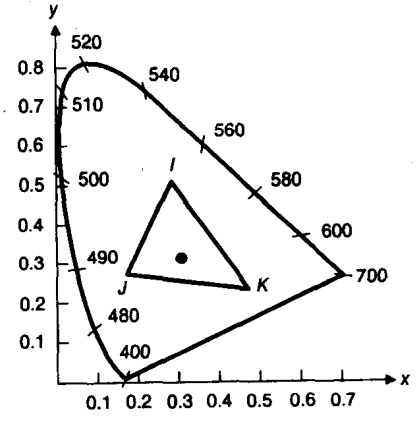


图13-26 混合颜色。混合颜色I和J可以产生直线段IJ上的所有颜色。混合颜色I、J和K可以产生三角形IJK中的所有颜色

色。色度图的形状表明，可见的红、绿、蓝三种颜色不能通过加法混合来匹配所有的颜色的原因：没有三角形一方面其三个顶点位于可见光区域之内，另一方面又完全覆盖了该区域。

色度图也可以用来比较不同彩色显示器和硬拷贝设备的颜色域。彩图II-2显示了彩色电视机监视器、胶片和打印机的颜色域。两个典型的显像管上荧光物质的色度坐标为：

	短时发光磷			长时发光磷		
	红	绿	蓝	红	绿	蓝
x	0.61	0.29	0.15	0.62	0.21	0.15
y	0.35	0.59	0.063	0.33	0.685	0.063

583

与彩色监视器相比，打印机的颜色域较小。为了准确地再现原先在监视器上显示的图像，监视器应该使用精简的颜色域，否则，准确再现是不可能的。但是，如果我们的目标是看起来不错，而不是完全准确再现，那么，颜色域之间的小的差别就不那么重要了。[HALL89]中讨论了如何压缩颜色域的问题。

CIE系统存在一个问题。考虑颜色$C_1 = (X_1, Y_1, Z_1)$到颜色$C_1 + \Delta C$的距离以及颜色$C_2 = (X_2, Y_2, Z_2)$到颜色$C_2 + \Delta C$的距离，这里$\Delta C = (\Delta X, \Delta Y, \Delta Z)$，两个距离都是$\Delta C$，但这两个距离看起来一般不一样。这是因为在13.2.1节中讨论的结果：沿着整个光谱，刚好可区分的颜色差是变化的。我们需要一个在感知上均匀的颜色空间，其中，相距的距离相等的颜色，观察者看起来它们的差别也是相等的。

为了满足这个要求，1976年开发了LUV均匀颜色空间。假定(X_n, Y_n, Z_n)为白光的颜色坐标，该颜色空间定义为

$$
\begin{aligned}
L^* &= 116\,(Y/Y_n)^{1/3} - 16,\ Y/Y_n > 0.01 \\
u^* &= 13\,L^*\,(u' - u'_n), \\
v^* &= 13\,L^*\,(v' - v'_n), \\
u' &= \frac{4X}{X + 15Y + 3Z}, \quad v' = \frac{9Y}{X + 15Y + 3Z}, \\
u'_n &= \frac{4X_n}{X_n + 15Y_n + 3Z_n}, \quad v'_n = \frac{9Y_n}{X_n + 15Y_n + 3Z_n}
\end{aligned}
\tag{13-22}
$$

由这些公式定义的可见颜色的3维体的形状当然与CIE（X, Y, Z）空间（图13-23）不同。

有了这些关于颜色的背景知识，我们现在将注意力转向计算机图形学中的颜色。

13.3 用于光栅图形的颜色模型

一个颜色模型定义了一个三维颜色坐标系和该坐标系的一个可见颜色子集，该子集与一个特定的颜色域相对应。例如，RGB颜色模型是三维笛卡儿坐标系中的一个单位立方体子集。

定义一个颜色模型是为了在一个颜色域中方便地指定颜色。我们最感兴趣的是彩色CRT监视器的颜色域，它由RGB（红、绿、蓝）三基色定义，如彩图II-2所示。在彩图中，我们看到一个颜色域是所有可见颜色的子集，因此，一个颜色模型不能用于指定所有的可见颜色。图13-27强调了这一点，它显示了CRT的颜色域是(X, Y, Z)空间的一个子集。

三个面向硬件的颜色模型是：用于彩色CRT监视器的RGB模型、用于彩色电视广播系统的YIQ模型和用于一些彩色打印系统的CMY（青色、品红色、黄色）模型。不幸的是，这些颜色模型都不易于使用，因为它们与直观的颜色概念：色调、饱和度、辉度之间并无直接的联系。为此，以易用性为目的开发了另外一类模型，[GSPC79；JOBL78；MEYE80；SMIT78]中介绍了几个这

样的颜色模型。这里，我们介绍三个：HSV模型（有时也称HSB模型）、HLS模型和HVC模型。

每一个颜色模型都给出了一种向其他模型转换的方法，对RGB模型来说，是到CIE的(*X*, *Y*, *Z*)空间的转换。转换是重要的，因为CIE是一个世界范围内的标准。所有的模型都有向RGB模型的转换，因此，我们可以从HSV模型等到RGB模型再到CIE标准进行转换。

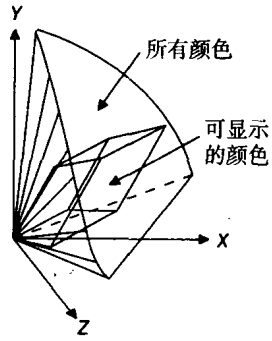


图13-27 CIE颜色空间中一个典型彩色监视器的颜色域，在监视器上能显示出来的颜色范围显然小于CIE空间中所有可见的颜色的范围。（Cornell大学计算机图形学组，Gary Meyer授权。）

13.3.1 RGB颜色模型

红、绿、蓝（RGB）颜色模型用于彩色CRT监视器，彩色光栅图形显示系统采用笛卡儿坐标系。RGB基色是加性基色，即单个基色的贡献加在一起得到结果颜色，如彩图II-3所示。我们感兴趣的子集是图13-28中的单位立方体，立方体的主对角线上的颜色包含等量的基色，代表灰色：黑色在(0,0,0)，白色在(1,1,1)。彩图II-4和彩图II-5显示了从不同角度观察RGB模型的结果。

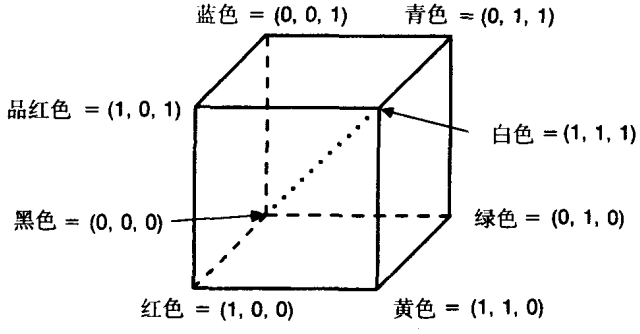


图13-28 RGB立方体，灰色位于主对角点画线上

584
~
585

RGB模型覆盖的颜色域由CRT上荧光物质的色度决定，两个荧光物质不同的CRT具有不同的颜色域。为了将在一个CRT颜色域中指定的颜色转换到另一个CRT的颜色域中，我们可以应用变换M_1和M_2，它们是从RGB颜色空间到（X, Y, Z）颜色空间的变换，变换的形式为：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{13-23}$$

其中，X_r, X_g和X_b是为了得到颜色X所需要的显像管RGB颜色的权值，依此类推。

定义M为颜色匹配系数组成的3 × 3矩阵，我们将式(13-23)写成：

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{13-24}$$

令M_1和M_2分别是从两个监视器颜色域到CIE颜色空间的变换，则$M_2^{-1}M_1$将监视器1的RGB颜色空间变换到监视器2的RGB颜色空间。如果所考虑的颜色同时落在两个监视器的颜色域中，那么这个矩阵乘积就能够完成转换工作了。如果颜色C_1落在监视器1的颜色域中但又不在监视器2的颜色域中，怎么办呢？对应的颜色$C_2 = M_2^{-1}M_1C_1$将落在单位立方体之外，因此无法显示。

一个简单但并不非常令人满意的办法是，截断颜色值，也就是说，将小于0的R、G、B值替换为0，将大于1的值替换为1。更好的但也更复杂的方法在[HALL89]中有所讨论。

RGB荧光物质的色度坐标在CRT生产厂商的技术说明书中可以查到，如果没有，也可以用色度计来直接度量色度坐标或用分光辐射谱仪度量$P(\lambda)$，然后用式(13-18)、式(13-19)和式(13-20)转换成色度坐标。记(x_r, y_r)、(x_g, y_g)和(x_b, y_b)分别为红、绿、蓝的坐标，定义C_r为

$$C_r = X_r + Y_r + Z_r \tag{13-25}$$

对红色，我们得到

$$x_r = \frac{X_r}{X_r + Y_r + Z_r} = \frac{X_r}{C_r},\ X_r = x_r C_r$$

$$y_r = \frac{Y_r}{X_r + Y_r + Z_r} = \frac{Y_r}{C_r},\ Y_r = y_r C_r$$

586

$$z_r = (1 - x_r - y_r) = \frac{Z_r}{X_r + Y_r + Z_r} = \frac{Z_r}{C_r},\ Z_r = z_r C_r \tag{13-26}$$

对C_g和C_b进行类似的定义，式(13-23)可以重写为

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} x_r C_r & x_g C_g & x_b C_b \\ y_r C_r & y_g C_g & y_b C_b \\ (1 - x_r - y_r)C_r & (1 - x_g - y_g)C_g & (1 - x_b - y_b)C_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{13-27}$$

未知数C_r, C_g和C_b可以用两种方法得到。其一，最大辉度的红色、绿色、蓝色的光强度Y_r, Y_g和Y_b可能是已知的，或者可以用高质量的光度计测量出来（便宜的光度计可能测量不出蓝色在2到10之间的值）。这些度量出来的光强度与已知的y_r, y_g和y_b组合在一起得到

$$C_r = Y_r/y_r,\ C_g = Y_g/y_g,\ C_b = Y_b/y_b \tag{13-28}$$

然后，将这些值代入式(13-27)，变换矩阵就全部由已知的值(x_r, y_r)，(x_g, y_g)，(x_b, y_b)，Y_r，Y_g和Y_b表示了。

如果知道或者测量出白色（$R = G = B = 1$）的X_w, Y_w和Z_w，我们也可以消除式(13-27)中的未知数。在这种情况下，式(13-27)可以重写为

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = \begin{bmatrix} x_r & x_g & x_b \\ y_r & y_g & y_b \\ (1 - x_r - y_r) & (1 - x_g - y_g) & (1 - x_b - y_b) \end{bmatrix} \begin{bmatrix} C_r \\ C_g \\ C_b \end{bmatrix} \tag{13-29}$$

求解C_r, C_g和C_b，将得到的结果代入式(13-27)。如果白色由x_w、y_w和Y_w给出，在求解式(13-29)之前，我们先找出：

$$X_w = x_w \frac{Y_w}{y_w},\ Z_w = z_w \frac{Y_w}{y_w} \tag{13-30}$$

13.3.2 CMY颜色模型

青色、品红色和黄色分别是红色、绿色和蓝色的补色。当将它们作为滤光片从白光中减去颜色时，它们称为*减性基色*。除了白色（而不是黑色）在坐标原点之外，CMY模型在笛卡儿坐标系中的子集与RGB模型一样。颜色通过如下方式指定：从白光中减去了什么，而不是向黑色中添加了什么。

当处理向纸上添加颜料的硬拷贝设备（如静电或喷墨绘图仪）时，了解关于CMY的知识非常重要。如果一个表面涂上了青色墨水，则它不反射红光，青色从反射的白光（它自己是红

色、绿色和蓝色之和）中滤去了红色。因此，从加性基色的角度来说，青色等于白色减去红色， 587
也就是等于蓝色加上绿色。类似地，品红色吸收了绿色，因此它等于红色加上蓝色；黄色吸收蓝色，因此它等于红色加上绿色。一个涂上青色和黄色的表面吸收了红色和蓝色，仅仅让白光中的绿光反射出来。一个涂上青色、黄色和品红色的表面吸收了红色、绿色和蓝色，从而是黑色的。这些关系如图13-29所示，可以从彩图II-6中看出来，也可以用下面的公式表示：

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{13-31}$$

由1组成的单位列向量是白色的RGB表示和黑色CMY表示。

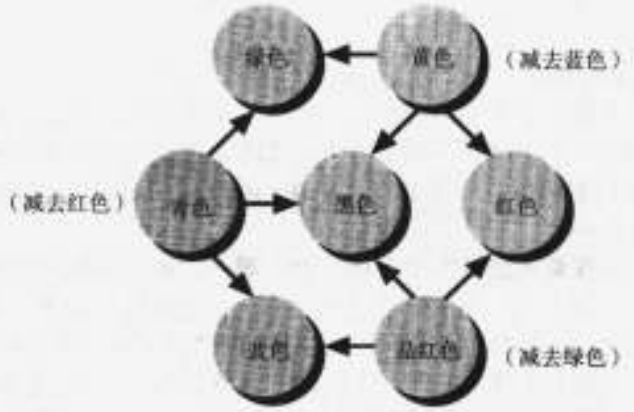


图13-29 减性基色（青色、品红色、黄色）和它们的混合色。例如，青色与黄色混合成绿色

于是，从RGB到CMY的转换为

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \tag{13-32}$$

利用这些变换可以将由红、绿、蓝三色的二进制组合得到的八种颜色转换成由青、品红、黄三色的二进制组合得到的八种颜色。这些转换常用于喷墨和静电彩色打印机。

另一个颜色模型CMYK采用黑色（简写为K）作为第四种颜色。CMYK模型用于打印出版 588
和一些硬拷贝设备的四色打印过程。给定一组CMY值，根据下面的关系式，黑色用来取代C、M和Y的等量部分：

$$\begin{aligned} K &= \min(C, M, Y) \\ C &= C - K \\ M &= M - K \\ Y &= Y - K \end{aligned} \tag{13-33}$$

这个内容在13.4节以及[STON88]中有进一步的讨论。

13.3.3 YIQ颜色模型

美国商业彩色电视广播系统采用YIQ模型，因而它与彩色光栅图形密切相关。YIQ是RGB的一种记录方式，目的是进行高效的传输以及向后兼容黑白电视。记录的信号采用NTSC（美国国家电视系统委员会）[PRIT77]系统传输。

YIQ模型中的Y代表的不是黄色，而是光强度，它的定义与CIE的Y基色相同。黑白电视只显示彩色电视信号中的Y分量；色度信息包含在I和Q信号中。YIQ模型采用三维笛卡儿坐标系，

可见颜色子集构成一个凸多面体，它映射为RGB立方体。

RGB到YIQ的映射定义如下：

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{13-34}$$

第一行的值反映出绿色和红色对辉度的贡献大，而蓝色对辉度的贡献相对较小。以上矩阵的逆矩阵完成从YIQ到RGB的转换。

式(13-34)中的RGB颜色是基于标准NTSC RGB荧光物质的，它的CIE坐标为

	红	绿	蓝
x	0.67	0.21	0.14
y	0.33	0.71	0.08

白色点（发光物C）的坐标是：$x_w = 0.31$，$y_w = 0.316$，$Y_w = 100.0$。

用YIQ模型指定颜色带来了一个潜在的因广播电视材料引起的问题：两个不同的颜色在一个彩色监视器上并排显示，看起来不一样；但是，如果转换成YIQ并在黑白电视上显示，它们
589 看起来就一样了。这个问题可以这样来解决，即在YIQ颜色模型空间中调整这两种颜色（仅调整Y值），使它们的Y值不一样。

YIQ模型利用了我们视觉系统的两个有用特性。第一，视觉系统对光强度的变化比对色调和饱和度的变化更敏感，也就是，我们区分空间中彩色信息的能力弱于区分空间中单色信息的能力。这表明，在表示Y、I和Q时，我们应该分配给Y更多的带宽，使得Y的分辨率高一些。第二，占据我们视场很小一部分的物体只能产生有限的色感，采用一个颜色维数就足以表示它了。这表明，I和Q中的一个可以比另一个拥有更低的带宽。NTSC将YIQ调制编码到一个广播信号中，利用以上特性在这个固定的带宽上传播最多的信息：4MHz分配给Y，1.5MHz分配给I，0.6MHz分配给Q。[SMIT78；PRIT77]中有关于YIQ的进一步讨论。

13.3.4 HSV颜色模型

RGB、CMY和YIQ模型是面向硬件的。相对应地，Smith的HSV（色调、饱和度、亮度）模型[SMIT78]（也称为HSB模型，其中的B（brightness）代表辉度）是面向用户的，它基于画家的颜色概念：色浓、色深与色调。该模型采用的坐标系是圆柱形坐标系，模型本身定义为其中的一个六棱锥或六面金字塔，如图13-30所示。六棱锥的顶面对应$V = 1$，其中包含了较亮的颜色，但是，$V = 1$平面中的颜色也并不同样的辉度。彩图II-7与彩图II-8显示了这个颜色模型。

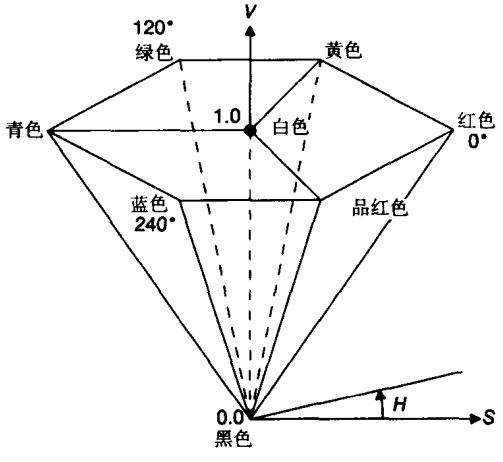


图13-30 单六棱锥HSV颜色模型。$V = 1$平面包含了所示区域中的RGB模型的$R = 1$、$G = 1$和$B = 1$平面

色调（即H）由绕纵轴的角度度量，红色对应0°，绿色对于120°，依此类推（见图13-30）。在HSV模型中，互补的颜色相差180°。S的值从0到1变化，它位于中心线
590 （V轴）上时值为0，位于三角边上时值为1。饱和度的度量是相对于该颜色模型所代表的颜色域的，而后者当然是整个CIE色度图的一个子集，因此，模型中饱和度为100%的颜色其色纯度小于100%。

六棱锥的V向高度是一个单位，顶点位于坐标原点。顶点对应黑色，V坐标是0。在该点处，H和S的值没有定义。对应$S=0$、$V=1$的点是白色，对应$S=0$而V取中间值的点是灰色。当$S=0$时，H的值没有定义。当$S\neq 0$时，H可有相应值。例如，红色对应$H=0$、$S=1$、$V=1$。事实上，对应$V=1$、$S=1$的任何颜色与画家所用的纯色颜料（用来混合产生其他颜色）都相近。添加白色颜料相当于降低S（不改变V），保持$S=1$降低V值改变色深，同时降低S和V值改变色调。当然，改变H相当于选择不同的纯色颜料。因此，H、S和V与画家颜色系统的概念密切对应，而与13.2节引入的术语不完全类似。

如果沿着主对角线从白色点到黑色点对RGB颜色立方体进行投影，所得到的投影结果与HSV六棱锥的顶面相对应，如图13-31所示。RGB立方体有子立方体，如图13-32所示。同样沿着主对角线观察，每一个子立方体也与图13-31中的六边形一样，只是小一点。在HSV空间中固定V值所得到的每个平面就对应着这样的RGB空间中子立方体的视图。从而，我们直观地建立了RGB模型与HSV模型的对应关系。图13-33和图13-34中的算法精确地给出了从一个模型到另一个模型的转换。

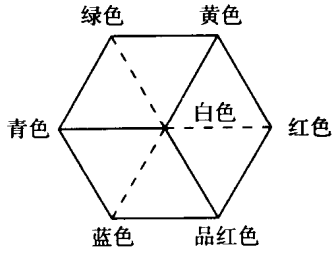


图13-31 沿着主对角线观察RGB颜色模型，立方体的可见边是实线，不可见边是虚线

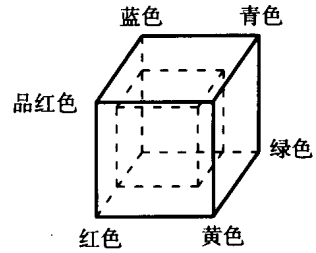


图13-32 RGB立方体和子立方体

```
void RGB_To_HSV (double r, double g, double b, double *h, double *s, double *v)
/* 给定：r，g，b属于[0,1] */
/* 期望：h属于[0,360 ], s,v属于[0,1]，或者如果s = 0，那么h = UNDEFINED */
/* （没定义）,它是超出[0，360]的值定义的常量* /
{
    double max = Maximum (r, g, b);
    double min = Minimum (r, g, b);
    *v = max;                                   /* 这是v值 */
    /* 下面计算饱和度s，如果红、绿、蓝的值都是0，饱和度是0 */
    *s = (max != 0.0) ? ((max - min) / max) : 0.0;
    if (*s == 0.0)
        *h = UNDEFINED;
    else {                                      /* 彩色的情况：饱和度不是0， */
        double delta = max - min;               /* 因而确定色调 */
        if (r == max)
            *h = (g - b) /delta;                /* 结果颜色在黄色与品红色之间 */
        else if (g == max)
            *h = 2.0 + (b - r) / delta;         /* 结果颜色在青色与黄色之间 */
        else if (b == max)
            *h = 4.0 + (r - g) / delta;         /* 结果颜色在品红色与青色之间 */
        *h *= 60.0;                             /* 将色调值转换成度数 */
        if (*h < 0.0)
            *h += 360.0;                        /* 确保色调值是非负的 */
    } /* 彩色情况 */
}  /* RGB_To_HSV */
```

图13-33 从RGB颜色空间到HSV颜色空间的转换算法

```
void HSV_To_RGB (double *r, double *g, double *b, double h, double s, double v)
/* 给定：h属于[0,360 ]或者UNDEFINED，s,v属于[0,1] */
/* 期望：r,g,b属于[0, 1] */
{
    if (s == 0.0) {          /* 颜色在黑白中心线上 */
        /* 非彩色：没有色调 */
        if (h == UNDEFINED) {
            *r = v;          /* 这是彩色情况 */
            *g = v;
            *b = v;
        } else
            Error();         /* 按照规定，如果s = 0并且h有值，则出错 */
    } else {
        double f,p,q,t;      /* 彩色：s! = 0，从而有色调 */
        int i;

        if (h == 360.0)      /* 360°与0°等价 */
            h = 0.0;
        h /= 60.0;           /* h属于[0，6] */
        i = floor (h);       /* floor返回小于等于h的最大整数 */
        f = h - i;           /* f是h的小数部分 */
        p = v * (1.0 - s);
        q = v * (1.0 - (s * f));
        t = v * (1.0 - (s * (1.0 - f)));
        switch(i) {
            case 0: *r = v;  *g = t;  *b = p; break;
            case 1: *r = q;  *g = v;  *b = p; break;
            case 2: *r = p;  *g = v;  *b = t; break;
            case 3: *r = p;  *g = q;  *b = v; break;
            case 4: *r = t;  *g = p;  *b = v; break;
            case 5: *r = v;  *g = p;  *b = q; break;
        }
    } /* 彩色情况 */
} /* HSV_To_RGB */
```

图13-34 从HSV颜色空间到RGB颜色空间的转换算法

13.3.5 HLS颜色模型

HLS（色调、明度、饱和度）颜色模型定义为圆柱坐标系中的双六棱锥，如图13-35所示。色调是绕双六棱锥纵轴的角度，红色对应0°（有些文献将蓝色置为HLS模型的0°，为了与HSV模型保持一致，我们仍然使红色对应0°）。当逆时针遍历其边界时，颜色出现的顺序与CIE色度图的一样：红色、黄色、绿色、青色、蓝色和品红色。这与HSV单六棱锥模型的颜色排列顺序也是一样的。事实上，我们可以将HLS模型看成HSV模型的变形：将$V=1$平面上的白色点向上拉伸，形成上面的六棱锥。与单六棱锥一样，在双六棱锥中互补的颜色相差180°，从纵轴到外表面饱和度从0变化到1，明度为0对应黑色（双六棱锥的下端点），明度为1对应白色（双六棱锥的上端点）。彩图II-9从某个角度显示了HLS模型。同样地，在这个模型中，术语色调、明度和饱和度与前文所讨论的相应术语含义类似但不完全相同。彩图II-10显示了HLS模型一个不同的侧面。

图13-36和图13-37中的程序完成了HLS模型和RGB模型之间的相互转换。它对Metrick[GSPC79]给出的程序进行了微小的修改：当$S=0$时，让H为UNDEFINED（无定义），使$H=0$对应红色而不是蓝色。

591 ~ 593

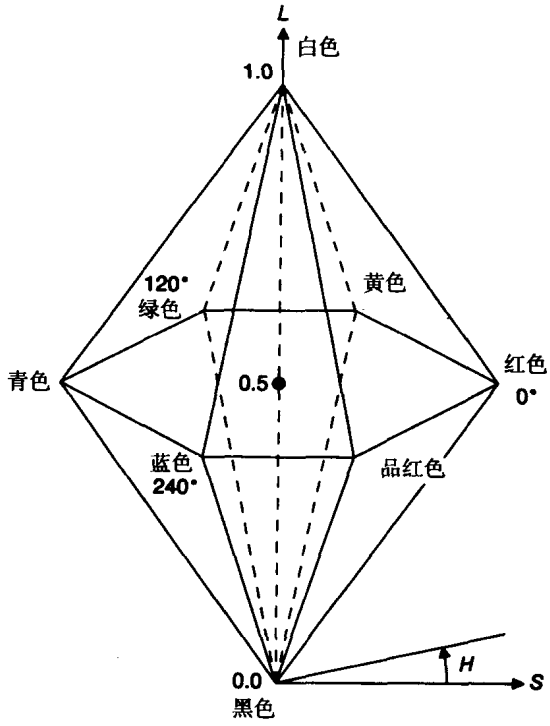


图13-35 双六棱锥HLS颜色模型

```
void RGB_To_HLS (double r, double g, double b, double *h, double *l, double *s)
/* 给定：r，g，b属于[0，1] */
/* 期望：h属于[0,360 ],l,s属于[0,1]，或者如果s = 0，那么h = UNDEFINED（没定义） */
{
    double max = Maximum (r, g, b);
    double min = Minimum (r, g, b);
    *l = (max + min) / 2.0;                    /* 这是明度 */
    /* 下面计算饱和度 */
    if (max == min) {                          /* 非彩色情况，因为r = g = b */
        *s = 0;
        *h = UNDEFINED;
    } else {                                   /* 彩色情况 */
        double delta = max - min;

        /* 首先，计算饱和度 */
        *s = (*l <= 0.5) ? (delta / (max + min)) : (delta / (2.0 - (max + min)));
        /* 接着计算色调 */
        if (r == max)
            *h = (g - b) / delta;              /* 结果颜色在黄色与品红色之间 */
        else if (g == max)
            *h = 2.0 + (b - r) / delta;        /* 结果颜色在青色与黄色之间 */
        else if (b == max)
            *h = 4.0 + (r - g) / delta;        /* 结果颜色在品红色与青色之间 */
        *h *= 60.0;                            /* 将色调值转换成度数 */
        if (h < 0.0)
            *h += 360.0;                       /* 确保色调值是非负的 */
    } /* 彩色情况 */
}  /* RGB_To_HLS */
```

图13-36 从RGB颜色空间到HLS颜色空间的转换算法

```
void HLS_To_RGB (double *r, double *g, double *b, double h, double l, double s)
/* 给定：h属于[0,360 ]或者UNDEFINED，l和s属于[0,1] */
/* 期望：r，g，b属于[0，1] */
{
    double m1, m2;

    m2 = (l <= 0.5) ? (l * (l + s)) : (l + s - l * s);
    m1 = 2.0 * l - m2;
    if (s == 0.0) {                  /* 非彩色：没有色调 */
        if (h == UNDEFINED)
            *r = *g = *b = l;        /* 这是彩色情况 */
        else Error ();               /* 如果s = 0并且h有值，则出错 */
    } else {      /* 彩色情况，因而由色调 */
        *r = Value (m1, m2, h + 120.0);
        *g = Value (m1, m2, h);
        *b = Value (m1, m2, h - 120.0);
    }
}  /* HLS_To_RGB */

static double Value (double n1, double n2, double hue)
{
    if (hue > 360.0)
        hue -= 360.0;
    else if (hue < 0.0)
        hue += 360.0;
    if (hue < 60.0)
        return n1 + (n2 - n1) * hue / 60.0;
    else if (hue < 180.0)
        return n2;
    else if (hue < 240.0)
        return n1 + (n2 - n1) * (240.0 - hue) / 60.0;
    else
        return n1;
}  /* 亮度 */
```

图13-37 从HLS颜色空间到RGB颜色空间的转换算法

与HSV模型一样，HLS模型易于使用。灰色对应$S=0$，但饱和度最大的色调在$S=1$、$L=0.5$处。如果用电位器指定颜色参数，L取值0.5时对应最强的颜色是一个不利因素；而在HSV模型中，$S=1$、$V=1$才对应最强的颜色。但是，与HSV模型类似的是，$L=0.5$平面上的颜色看起来辉度并不全相同，因此，看起来辉度相同的两种不同颜色一般具有不同的L值。此外，在13.2节讨论的意义上，HLS模型以及本节讨论的其他模型视觉上都不是均匀的。

由Tektronix公司最近开发的TekHVC（色调，亮度，色度）颜色系统对13.2节讨论的均匀颜色空间CIE LUV进行了修改。在该颜色空间中，颜色之间的度量距离和观察距离大致相等。这一点是CIE LUV和TekHVC模型很大的优点。图13-38显示了HVC模型的

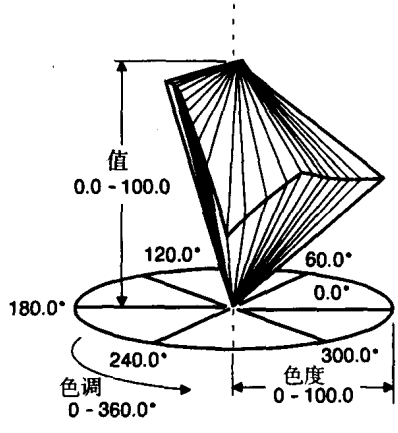


图13-38 TekHVC颜色模型。（Tektronix公司授权。）

一个视图，彩图II-11显示了另一个视图。从CIE到TekHVC的变换细节还没有被发布，但从式(13-22)可以看出，从CIE XYZ到CIE LUV的变换在计算上是简单的，立方根是其中最复杂的运算。因此，我们希望在将来，视觉上均匀的颜色空间会得到更广泛的应用。

13.3.6 颜色的交互指定

许多应用程序允许用户指定区域、直线段、文本等的颜色。如果系统只提供少数几种颜色，将可用的样品颜色在菜单列出来以供选择是可行的；但如果颜色多得无法合适地列出来，怎么办呢？

简单的办法是用英语名称（直接输入或者用滑动刻度盘输入）在颜色空间中指定颜色的数值坐标，或者直接与颜色空间的图形表示进行交互。命名方法通常不能令人满意，因为名字的含义模糊并带有主观性（“淡海蓝色外加一点绿色”），这与图形交互技术的目标是相对立的。但[BERK82]中描述了一种定义良好的颜色命名机制——CNS，它用“带绿色的黄色”、“绿黄色”和“带黄色的绿色”等术语区别介于绿色与黄色之间的三种色调。在一个试验中，一组用户通过CNS指定颜色，另一组在RGB或HSV空间中输入数值坐标指定颜色，结果是前一组比后一组更准确。

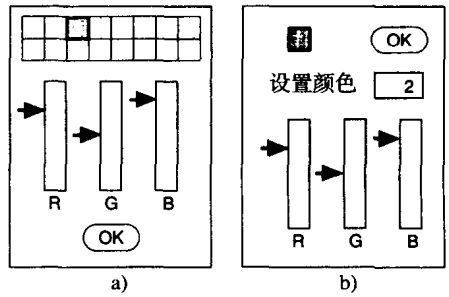


图13-39 设置颜色的两种常用方法。在a)中，用户在16种颜色中进行选择，被选中的颜色由黑框标识，RGB滑动刻度板控制颜色，OK按钮用来释放颜色控制板。在b)中，用户输入所选颜色的序号，当前的颜色显示在灰色的框里。在这两种情况下，用户必须同时看到实际的颜色，以便了解颜色改变的结果

在任一种颜色模型中指定颜色坐标可用滑动刻度板实现，如图13-39所示。如果用户了解其中的每一个值对颜色的影响，这项技术则非常有效。也许最好的颜色交互指定方法是与颜色空间的图形表示直接交互，如图13-40所示。在圆形区域（代表$V=1$平面）内转动直线决定了HSV立体中哪一个剖面在三角形区域内显示，三角形区域内的光标可以移动用来确定饱和度与亮度值。当直线或光标移动时，数字读数器的值发生改变。当用户在读数器中直接键入新的数值时，直线段和光标的位置也随着发生变化。颜色样品框显示当前被选中的颜色。彩图II-12展示了一个用于HSV模型的类似方法。

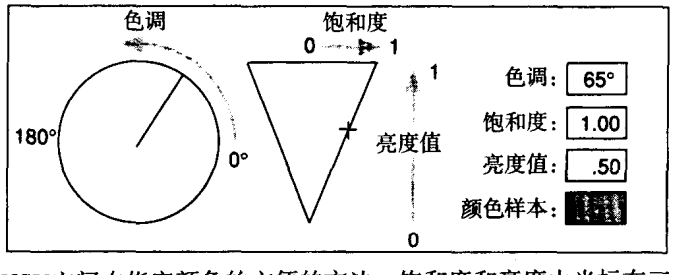


图13-40 一种在HSV空间中指定颜色的方便的方法。饱和度和亮度由光标在三角形区域内指示，色调由圆形区域内的直线段指示。用户可以移动图中的直线段和光标指示器，读数器中的数值将随着更新。另一方面，用户也可以键入新的数值，那么指示器也随着更新位置。也可以为H、S和V分别增加滑动刻度板，让用户一次在一个维数上精确控制它们的数值，而不再需要键入

[SCHW87]中描述了颜色匹配试验，其中用户在多种模型（包括RGB、YIQ、LAB[WYSZ82]

和HSV）中用数据输入板指定颜色，结果发现，用HSV模型慢但是准确，而用RGB模型快但是准确性差一些。有一种被广泛认同的看法是：HSV模型易于使用，这使它常常成为用户的首选模型。

许多允许用户指定颜色的交互式系统允许用户将颜色设置以一系列相邻颜色块的形式显示出来，如图13-39a所示，或者仅显示正在设置的单个像素值的颜色，如图13-39b所示。当用户操纵滑动刻度板时，颜色样品发生改变。然而，一个人对颜色的观察结果受周围颜色以及颜色区域大小的影响，如彩图II-13所示，因此，在反馈区域中观察到的颜色也许与实际观察结果不一致。所以，用户在设置颜色时，也要同时观察一下实际的显示结果。

13.3.7 在颜色空间中进行插值

颜色插值至少在三种情况下是必须的：Gouraud明暗处理（16.2.4节）、反走样（3.17节）和对两个图像进行溶合，如淡入淡出序列。颜色插值的结果依赖于我们在哪个颜色模型中进行插值，因此，必须要小心地选择一个合适的颜色模型。

如果从一个颜色模型到另一个颜色模型的转换将直线段（表示插值路径）变为直线段，那么在这两个模型中进行插值的结果是一样的。RGB、CMY、YIQ和CIE颜色模型就属于这种情况，它们之间通过一个简单的仿射变换联系起来。然而，RGB模型中的一条直线段通常不能转换到HSV或HLS模型中的一条直线段。彩图II-14显示了在HSV、HSL、RGB和YIQ颜色空间中对两个相同的颜色进行线性插值的结果。考虑在红色和绿色之间插值。在RGB模型中，红色 = (1,0,0)，绿色 = (0,1,0)，它们的插值（为了方便取权值都等于0.5）是(0.5,0.5,0)。将算法RGB_To_HSV（图13-33）应用于这个结果，我们得到(60°,1,0.5)。现在，在HSV模型中表示红色和绿色，我们有(0°,1,1) 和(120°,1,1)，用同样的权值在HSV模型对它们插值，得到(60°,1,1)，结果与在RGB模型中插值相差0.5。

作为第二个例子，考虑在RGB模型和HSV模型中对红色和青色插值。在RGB模型中，我们从（1,0,0）和（0,1,1）开始，插值得到（0.5,0.5,0.5），而这个结果颜色在HSV模型中表示为(UNDEFINED,0,0.5)。在HSV模型中，红色和青色分别是(0°,1,1)和(180°,1,1)，插值结果得到(90°,1,1)；亮度和饱和度保持最大时得到了一个新的色调，而“正确”的结果是：两个等量的补色混合得到的应该是灰色。这又一次表明，插值然后变换与变换然后插值所得到的结果是不同的。

594 ~ 598

Gouraud明暗处理方法可以采用任一种颜色模型，因为两个待插值的颜色非常靠近，从而插值路径也非常靠近。当两个图像做溶合时，如在淡入淡出序列和反走样中，颜色有可能相距很远，此时采用加性模型（如RGB）比较合适。另一方面，如果目标是在两个具有固定色调（或饱和度）的颜色之间进行插值，并且希望保持所有颜色的色调（或饱和度），那么采用HSV或HLS模型更好。但是要注意，在HSV或HLS模型中进行固定饱和度插值，观察者看起来饱和度不一定是固定的。

13.4 颜色再现

打印是彩色图像的一个再现过程，再现的方式与黑白图像类似，只不过此时打印的是4组半色调点，每一组对应一种减性基色，余下一组对应黑色。通过一个称为底色消除的过程，黑色取代了等量的青色、品红色和黄色。这样创建的黑色比用三种基色混合产生的黑色更黑，同时因为减少了需要的青色、品红色和黄色的墨水，加速其变干。每一个半色调点的方向不同，从而它们之间不会形成干涉现象。彩图II-15显示了一个放大了的半色调彩色模式。我们的眼睛在空间上综合了相邻点的反射光，因此我们看到的颜色由相邻点基色的混合比例确定。这种对不同的颜色进行空间综合的现象与我们前面所谈到的在观察一个彩色监视器上红色、绿色和蓝色三个荧光点时所发生的现象是一样的。

于是，我们推断，在打印纸和CRT上彩色地再现与黑白地再现依赖于同样的空间综合。13.1.2节中讨论的黑白抖动技术也可以用于彩色，以增加可用的颜色个数，同样这也是以降低空间分辨率为代价的。考虑每个像素有3位（红、绿、蓝三色分别有1位）的彩色显示器，若采用2×2的像素模式，我们能获得125种不同的颜色：红、绿、蓝每一种模式能产生5个亮度级，应用图13-8的半色调模式，结果得到5×5×5=125种组合。

并不是所有的彩色再现都依赖于空间综合。例如，静电彩色复印机、喷墨绘图仪和热彩色打印机事实上在纸的表面混合了减性基色颜料，获得了一小组不同的颜色。在静电复印技术中，首先彩色颜料连续分三步沉积出来，然后加热并溶合在一起。绘图仪喷出的墨水先混合再烘干。空间综合可进一步用来扩展颜色范围。

当我们需要将每像素具有n位的一个彩色图像显示在每像素具有$m(m<n)$位的显示器上，而又不能降低空间分辨率时，相关的量化问题出现了。这里，颜色分辨率必须降低。在这种情况下，存在两个关键问题：如何选择应该显示的2^m种颜色？如何将图像中的2^n种颜色映射成可以显示的较少的2^m种颜色。

一个简单的答案是采用预定义的可显示颜色集合，以及一个从图像颜色到显示器颜色的固 599
定映射。例如，如果$m=8$，一个典型的分配方案是，红色和绿色分别占3位，蓝色占2位（因为人眼对蓝色的敏感程度低）。从而，256种可显示的颜色是8种红色、8种绿色和4种蓝色的混合色。红、绿、蓝三色的值按比例分布在0.0到1.0范围之内，如13.1.1节中讨论的那样。对一个每种颜色有6位从而每种颜色有64个级别的图像来说，64种红色被映射成8种可显示的红色，绿色也一样，64种蓝色仅被映射成4种蓝色。

然而，如果采用这种方法，当图像中的蓝色在颜色空间中集中在一起时，它们可能被显示成一种同样的蓝色，而其他三种可显示的蓝色没用上。一个具有适应性的方法是考虑概率分布，基于蓝色值的分布将蓝色值分成若干区间。Heckbert[HECK82]描述了两种这样的方法：普及算法和中值分割算法。

*普及算法*为图像颜色创建一个直方图，根据直方图在映射中应用出现频率最高的2^m种颜色。*中值分割算法*递归地用一个长方体匹配图像中出现的颜色，在长边的中点处将它分割成两半。当创建了2^m个长方体时，递归结束。每个长方体的质心处颜色用来替代位于该长方体之内的所有图像颜色。中值分割算法比普及算法要慢，但产生的结果更好。[WAN88]中讨论了另一种分割长方体的方法。

创建精确的颜色再现比产生近似颜色要难得多。可以调整两个监视器，使它们创建相同的三色激励值，所涉及到的多个步骤在[COWA83；MEYE83]中有详细讨论。这些步骤是：度量监视器荧光物质的色度坐标；调节监视器电子枪的辉度与对比度，使得当$R=G=B$时，它们产生相同的白色；为每一个电子枪确定合适的γ校正。

制作看起来与显示器上的图像完全一样的幻灯片或电影胶片很困难，因为这涉及到许多因素，包括监视器的γ校正、胶片记录器的CRT的γ校正、胶片记录器的CRT所发出的光的颜色、胶片记录器的滤光器、胶片的类型、感光化学物质的质量和温度、曝光时间、幻灯或胶片投影机灯泡所发出的光的颜色。幸运的是，所有这些因素都可以量化和控制，虽然困难相当大。

在打印材料上控制颜色匹配也相当困难，采用青、品红、黄和黑这几种基色，打印过程需要小心地控制以保持定位和墨水流。纸的纹理、吸收率以及光泽也影响结果。使问题更复杂的是，13.3.2节讨论的简单减性CMY颜色模型不能直接拿来用，因为它没有考虑以上这些因素。更详细的讨论请参见[STON88]。

虽然在颜色再现过程中我们已经非常仔细了，但结果与原图好像总不能完全一样。光照条件和显示器的反射可能会使具有相同色度坐标的颜色看起来不同。幸运的是，颜色再现的目的
600 通常是（并不总是）保持图像不同部分之间的颜色关系，而不是获得一个精确的拷贝。

13.5 在计算机图形学中应用颜色

我们应用颜色的目的是为了美，为了建立一种氛围和情绪，为了真实，为了强调，为了识别相关联的区域，或者为了进行编码。通过仔细的考虑，颜色可以被有效地用来实现这些目标。另外，用户喜欢颜色，尽管还没有数据证明颜色能提高他们的效率。虽然对价格非常在意的购买者可能藐视彩色显示器，但我们相信，任何可以鼓励人们使用计算机的事物都是重要的。

粗心地应用颜色可能会使显示器比一个相应的单色演示可用性更差、更不具吸引力。在一个实验中，引入了无意义的颜色使用户性能减少到引入之前的三分之一[KREB79]。对颜色的应用应该适度，任何装饰性的使用应服从于功能性使用，使颜色不会被误解为具有什么潜在的含义。因此，就像人机界面的其他方面一样，颜色的使用必须由实际用户来测试，以便发现和解决问题。当然，有些人可能有其他偏爱，因此，通常的做法是，基于颜色使用规则提供缺省的选择，同时给用户提供改变缺省值的手段。一个谨慎的颜色选择方法是，首先为单色显示器设计，保证颜色使用是完全冗余的。这避免了给没有彩色设备的用户带来问题，也意味着这个应用可以在单色显示器上应用。关于在颜色不足时如何设计的更多信息请参见[MEYE88]。在彩图I-26到彩图I-29显示的窗口管理器中，颜色选项设计是相当谨慎的。颜色不是按钮状态、选中的菜单项等的惟一编码。

许多书讨论了颜色在美学上的应用，包括[BIRR61]，我们在这里仅介绍有助于产生和谐颜色的几条简单规则。颜色美学最基本的规则是根据一些方法选择颜色，通常，在颜色模型中沿着一条光滑的路径遍历颜色，或将颜色限制在颜色空间内的平面或六棱锥上。这也许意味着选择具有恒定明度的颜色。此外，颜色最好在观感上是均匀分布的（这与在某个坐标方向上等间隔分布是不一样的，实现起来很困难）。注意，在不同的颜色空间中对两个颜色进行线性插值（如在Gouraud明暗处理方法中）得到的结果不同（参见习题13.10和彩图II-14）。

随机地选择色调和饱和度通常显得有些过分艳丽。Alvy Ray Smith做了一个非正式的实验，在这个实验中16 × 16个格子用随机产生的颜色分别填充，得到的结果并不意外：格子毫无吸引力。根据H、S和V值对256种颜色进行排序并按照新的顺序重新显示，格子看起来明显改善了。

关于这些规则的更多具体的例子表明，如果一个图表中仅包含几种颜色，背景应该选用其中一种颜色的补色。如果一幅图中包含许多颜色，背景应该选用一种中性的颜色（灰色），因为它既和谐又不显眼。如果两个相邻的颜色不是特别和谐，可以用细的黑色边框将它们分开。边框的应用在非彩色（黑/白）视觉通道中更有效，因为黑色的轮廓能加速形状检测。以上讨
601 论的一些规则在ACE（A Color Expert）中被编码，ACE是一个用于选择人机界面颜色的专家系统。一般地，尽量少用不同的颜色总是好的（真实感图形的明暗处理除外）。

颜色可以用于编码，如第9章讨论的以及彩图II-16显示的那样。但是，按照次序有几点需要注意，第一，颜色代码很容易携带并不想要的含义。将A公司的收入显示成红色，而B公司的收入显示成绿色可能足以使人理解为：A公司出现了经济困难，因为我们通常总是将颜色与一些含义联系在一起的。亮的、饱和度高的颜色比暗淡的、灰白的颜色更突出，可能会强调并不存在的重点。显示器上两个具有同样颜色的元素可能因为相同的颜色代码而被视为是相关联的，虽然实际并非如此。

当颜色同时被用来对菜单项分组和区分图形元素如印制电路板的不同层或VLSI芯片时，

这个问题经常出现。例如，我们倾向于把绿色的图形元素与同样颜色的菜单项关联起来。这是限制应用于人机界面元素（如菜单项、对话框和窗口边界）的颜色的原因之一。（另一个原因是将尽量多的颜色留给应用程序自己。）

许多颜色使用规则是基于生理学而不是美学的考虑。例如，因为人眼对亮度的空间变化比对色调的空间变化更敏感，线段、文本以及其他的细节图形不仅要在色调上而且要在辉度（观察到的亮度）上与背景相区别，特别是对那些包含蓝色分量的颜色，因为只有相对较少的视锥体对蓝色敏感。从而，两个仅仅在蓝色分量上有所区别的等亮度彩色区域的边界很难辨别出来。另一方面，对蓝色敏感的视锥体比对红色和绿色敏感的视锥体分布得更广，因而我们的外围彩色视觉对蓝色更好（这解释了为什么许多警车的闪光灯现在采用蓝色而不是红色）。

蓝色与黑色在辉度上相差甚小，它们是一对特别差的组合。类似地，黄色在白色上很难区分，因为这两种颜色都很亮（参见习题13.11）。彩图I-28与彩图I-29显示了在白色的背景上，利用黄色非常有效地加亮了黑色文本。黄色与黑色文本的对比度很大，并且也很突出。另外，应用于文本反色时，用黄色加亮并不比用黑色加亮（也就是说，被加亮的文本是白色，文本背景是黑色，这在单色显示器上很常见）更好。

蓝色背景上的白色文本提供了很好的对比度，同时它也不像白色在黑色上那么刺眼。最好避免低饱和度和低光强度的红色和蓝色，因为红绿色盲者（最常见的色觉缺陷）不能区分这些颜色。*Meyer*和*Greenberg*描述了如何为色盲观察者选择颜色[MEYE88]。

眼睛不能区分非常小的物体的颜色，就像在介绍YIQ NTSC颜色模型时已经提到的那样，因此，颜色编码不应该用于小的物体。特别地，判断对着20到40弧分（1弧分 = 1/60度）的物体的颜色容易出错[BISH60；HAEU76]。一个高0.1英寸的物体，如果从24英寸（一个典型的观察距离）处观察，它占据这么大的弧度数；如果15英寸显示器纵向上有1024条扫描线，那么它对应7个像素高。很明显，单个像素的颜色是很难辨别的（参见习题13.18）。

观察一个区域所感受到的颜色受周围区域的颜色影响，这在彩图II-13中非常明显。如果用颜色对信息编码，这种影响就特别成问题。当周围区域的颜色是灰色或相对不饱和的颜色时，这种影响达到最小。 602

区域的颜色实际上能够影响它的观察尺寸。Cleveland和McGill[CLEV83]发现，对于一个红色方块和一个绿色方块，看起来，红色方块更大。这种效果足以使观察者认为红色方块比绿色方块更重要。

如果一个用户盯着颜色饱和度很高的一块大区域几秒钟，然后看往别处，这块大区域的残留图像就会出现。这种影响令人不安，并且使眼睛疲劳。因而，使用高饱和度颜色的大区域是不明智的。同样，不同颜色的大区域看起来与观察者距离不一样，因为光的折射依赖于波长。当观察者从凝视一种颜色的区域改为凝视另一种颜色的区域时，眼睛改变了聚焦点，这种聚焦的改变形成了深度不同的印象。红色和蓝色，在光谱的两端，具有最强的深度不一致效果，红色看起来更近，蓝色看起来更远。因此，同时将蓝色用于前景物体而红色用于背景是不明智的，反过来则很好。

了解了关于颜色使用的所有这些危险和缺陷，你对我们把谨慎地使用颜色作为第一个规则还感到惊奇吗？

13.6 小结

随着彩色监视器和彩色硬拷贝设备在许多应用中变成标准配置，颜色在计算机图形学中也越来越重要。在这一章中，我们介绍了那些与计算机图形学最相关的颜色概念，如果想了解这

方面的更多信息，请查阅关于颜色的大量文献，如[BILL81；BOYN79；GREG66；HUNT87；JUDD75；WYSZ82]。更多的关于在计算机图形学中应用颜色的艺术和美学方面的问题请参见[FROM84；MARC82；MEIE88；MURC85]。关于如何精确调节监视器以及如何匹配监视器颜色与打印颜色的问题在[COWA83；STON88]中有所讨论。

习题

13.1 计算每个像素有w位，大小为$m \times m$的像素模式能表示的亮度数目。

13.2 写一个程序，采用查色表对黑白显示器进行γ校正。输入参数是γ, I_0, m（期望的亮度数目）和K（式(13-5)中的常量）。

13.3 写一个算法，在具有两级亮度的输出设备上显示像素阵列，算法的输入是每个像素有w位$m \times m$像素亮度阵列和一个$n \times n$的增序矩阵。假定输出设备的分辨率是$m \cdot n \times m \cdot n$。

13.4 应用有序抖动重做习题13.3。现在，输出设备的分辨率是$m \times m$，与输入的像素阵列一样。

13.5 写一个算法，用$n \times n$填充模式在只有两级亮度的设备上显示一个填充多边形。

13.6 当用一定的模式填充显示在隔行扫描显示器上的多边形时，所有处于“开”状态的位要么落在奇数行扫描线上要么落在偶数行扫描线上，引起了一定的闪烁。修改习题13.5中的
603 算法，交换$n \times n$模式的各行使得模式的各行能够交替利用奇数和偶数行扫描线。图13-41显示了对图13-8采用亮度1得到的结果，a)图进行了交替，b)图没有。



图13-41 用两种方法对图13-8取亮度1得到的结果：a)交替（加亮的像素同时落在两条扫描线上），b)非交替（所有加亮的像素落在同一条扫描线上）。

13.7 给定一个光谱能量分布，如何计算它所对应颜色的主波长、色纯度和光强度？

13.8 在RGB立方体、HLS双六棱锥和HSV六棱锥上标出亮度值分别是0.25、0.50和0.75的点的位置，其中亮度值由$Y = 0.30R + 0.59G + 0.11B$定义。

13.9 为什么CIE色度图光谱两端用直线段连接？

13.10 用R、G、B表示YIQ模型中的I、HSV模型中的V和HSV模型中的L，注意I、V和L是不一样的。

13.11 在YIQ颜色空间中计算加性基色和减性基色的光强度，按照光强度对它们排序。这个排序给出了它们的相对亮度，就像在黑白电视上显示的以及我们眼睛的观察到的那样。

13.12 讨论一个光栅显示器的设计，如果它采用HSV或HLS模型（而不是RGB模型）来表示颜色。

13.13 哪一种颜色模型最能表现颜色的协调规则？

13.14 证明当$R = G = B = 1$时，式(13-27)可以重写为式(13-29)。

13.15 如果在式(13-29)中用于计算C_r、C_g和C_b的白色由x_w、y_w、Y_w给出，而不是X_w、Y_w和Z_w，那么C_r、C_g和C_b的代数表达式是什么？

13.16 重写HSV到RGB的转换算法，使它的效率更高。用表达式$vs = v*s$；$vsf = vs*f$；$p = v -$

vs；$q = v - vsf$；$t = p + vsf$替换p，q和t的赋值表达式。假定R、G、B在区间[0，255]中，看看有多少运算可以转换成整数运算。

13.17 写一个程序，在显示器中并列显示两个16 × 16网格，用颜色填充它们。左边的网格包含从HSV颜色空间中随机选出的256种颜色（对H、S、V，用随机数发生器从10个等间隔的数值中选择一个）。右边的网格包含同样的256种颜色，但是按照H、S、V的值做了排序，变换排序关键字H、S、V，观察实验结果。

13.18 写一个程序，在一个灰色背景上显示橙色、红色、绿色、蓝色、青色、品红色和黄色小方块，每一个方块与其他方块是相互分离的，大小为$n \times n$像素，n是一个输入的变量。为了在24英寸和48英寸的距离处清楚地判断每个方块的颜色，n的值必须为多大？这两个n值之间有什么关系？不同的背景色对这个结果有什么影响？

13.19 给定亮度变化范围是50、100和200，计算为了保存256个亮度级所需要的查色表每单元的位数。

13.20 写一个程序，在RGB、HSV、HSL模型中对两个颜色线性插值。接受两个颜色作为输入，并允许在这三个模型的任一个中指定它们。 604

第14章　可视图像真实感的探讨

前几章我们讨论了简单的二维和三维图元的图形技术。我们所产生的图片（例如，第6章中房屋的线条图）表示了在现实生活中其结构和外观上都复杂得多的物体。本章中，我们介绍一种越来越重要的计算机图形学的应用：生成三维场景的真实感图像。

何为*真实感图像*？由绘画、拍照或是计算机所产生的场景图片，在什么意义上可以说是“真实的”是一个很有争议的学术题目[HAGE86]。我们较为广泛地使用这一名词，用以指那些体现了光线与真实物体相互作用所具有的许多效果的照片。这样，我们将真实感图像看成图片及其生成技术的一种引伸和不严格的说法，也就是说，看成“多”或“少”真实感。在延伸的一端的例子是通常被称为*照相真实感*（或*真实感照相*）。这些图像试图综合聚焦在对着所描述物的照相机胶片底版上的光照强度的区域。按照我们的方法，延伸的另一端，我们将寻觅能持续地提供较低的可视线索的图像。

您在思想上应容忍这样的观点，那就是：一张更有真实感的图片不必是更理想的或更有用的图片。如果一张图片的最终目的是传达信息，那么，一张没有复杂的阴影和反射的图片也许比一张真实感照片的绝技更成功。另外，在随后章节中，概要描述该技术的多种应用，真实性被有意地改变成了审美的效果或满足天真的观察者的期望。同样的道理，科幻影片的特性可以做到在外层空间中武器的爆炸声——在真空中这是不可能的。例如，在彩图II-20中显示的天王星，Blinn在行星黑暗面加入了额外的光线，形成反差，以使得所有的特征是同时可见的，否则，星球的黑暗面将是黑的。在物理上的随意性可产生具有魅力的、令人难忘的有用的图片。 605

生成真实感图片涉及到若干步骤，这些步骤将在随后的章节中详述。虽然，这些步骤往往视为构成了概念流水线，但正如我们将看到的，根据所用的算法，这些步骤的实现顺序可以不同。首先，用第11、12和20章中所讨论的方法来生成对象的模型。其次，选择一个观察规范（如第6章中所开发的）和光照条件。由第15章所讨论的算法来决定观察者的可见面。在可见面的投影中，对每个像素所赋的颜色是对象所反射和透射的光线的一个函数，并由第16章所论述的方法确定。然后，最终的结果照片可使用第17章的合成技术，与早先所生成的那些照片组合而成（例如，再使用一个复杂的背景）。最后，正如在第21章所讨论的那样。若我们正产生一个动画序列，在造型、光照以及观察者所做的规定等随时间所做的改变，必须加以定义。从模型产生图像的过程通常被称为*绘制*。*光栅化*这个术语是专门指从输入几何图元到确定像素值的各个步骤。

本章从多个角度提出真实感绘制问题。首先，我们注意一些已经使用真实感图像的应用。然后，我们概略地回顾历史进程，考察了能成功地生成较真实的图片的一系列技术。每种技术都以一幅用到其新技术的标准场景的图片来说明。其次，我们观察了由走样产生的问题。这是当图像被表达成像素的离散阵列时必须处理的问题。最后，我们以如何进入后续各章的建议结束本章。

14.1　为什么讨论真实感

产生真实感的图片是模拟、设计、娱乐和广告、科研和教育以及指挥和控制等领域中的一项重要目标。

模拟系统出示的图像，不仅要有真实感，而且要动态地变化。例如，飞行模拟器显示的视

图就应当是从一架飞行飞机的驾驶仓所看到的。为了产生运动的效果，该系统每秒钟应多次产生并显示一个新的稍微不同的视图。如彩图I-5所示的模拟器已用于训练宇宙飞船、飞机和轮船的驾驶员，最近，还用于训练汽车驾驶员。

汽车、飞机和房屋等三维物体的设计者想要看一看他们初步设计的样子。与构造模型或原型相比，产生真实的由计算机生成的图像往往是一个比较容易的、代价较小的和更为有效的看初步结果的方法，并且还允许考虑更多不同的设计。若设计工作本身也是用计算机辅助进行的，那么，目标对象的一种数字化的描述也已经可用于产生图像。理想情况下，设计者还可依据所显示的图像交互地修改设计。彩图II-17显示一个由汽车设计系统生成的图像，用来决定在各种
606 光线条件下，一辆汽车看起来是什么样子。真实感图像还往往与程序相结合，用来分析正在设计中的对象的其他方面，例如，其质量特性或对应力的反应等。

计算机成像术广泛地应用于娱乐界，不论是传统的动画卡通，还是作为商标徽记、广告宣传和科学幻想电影的真实感和超现实的图像（见彩图D、F、I-11、I-12、II-18）。计算机产生的卡通可仿制传统的动画，还可以借助于引入更复杂的动作和更丰富或更有真实感的成像术而胜过手工技术。某些复杂的真实感图像的生产成本低于从该对象的物理模型所拍摄的照片。现在已生成出另外一些由真实模型实施极为困难或者不可能的图像。为娱乐表演创建的专用硬件和软件包括复杂的绘画系统、产生特殊效果和组合图像的实时系统。随着技术的进步，家庭和游乐场所的视频游戏产生出真实感不断提高的图像。

真实感图像正变成研究和教育方面的一种基本工具。一个特别重要的实例是在分子建模方面图形学的应用，如彩图II-19所示。有趣的是，在这里真实感的概念是如何展现的：真实感的绘图并不是“真”的原子，而是模仿的球和棒的形态和体积的模型。这允许建造比物理模型可能达到的更大的结构，并且它还允许有特殊效果，例如，表示反应逼真的震动键和色彩变化。在一个宏观的尺度上，在JPL制作的电影显示出NASA的空间探索任务，显示在彩图II-20中。

真实感成像术的另一个应用是在指挥和控制方面，其中，用户需要有关的信息并控制由图片表达的复杂过程。与模拟不同，模拟是试图去模仿用户将有效看到的和感觉到的情形，而命令和控制的应用往往是针对所做的决策产生符号显示，强调某些数据并抑制另一些数据。

14.2 基本的困难

完全实现视觉真实感的基本困难是真实世界的复杂性。请注意你周围环境的丰富多彩。有许多表面纹理、精细的色彩灰度、明暗阴影、反射和周围物体的细微的不规则性。请想一想起皱的衣服上的图案、皮肤的皱纹、蓬乱的头发、在地板上磨损的痕迹以及碎裂的墙上油画。所有这些组合起来产生了一个“真实的”可视的体验。为模拟这些效果，计算代价可以很高：在高性能计算机上产生彩图A至彩图H那样的图片，可能要花几分钟甚至几个小时。

在探索真实感方面，更容易满足的子目标是提供充分的信息让观察者理解几个对象之间的三维空间关系。这个子目标能以一个低得多的代价来完成，它也是在CAD和许多其他应用领域中一个通常的要求。虽然，高度真实感的图像传达了三维空间关系，但它们也往往传达了更多的信息。例如，图14-1是一个简单的线条图形，但却令人信服地告诉我们：一个建筑物的一部分是在另一个建筑物的后面。这并不需要显示铺砌瓦片和砖块的建筑

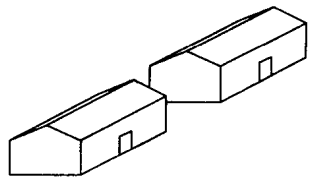


图14-1 两幢房屋的线条图

物的表面，也不需要显示建筑物投射的阴影。实际上，在某些相关情况中，这些特殊细节也许只会分散观察者对正被描述的更重要的信息的注意力。

在描述空间关系方面，一个长期存在的困难是：绝大多数显示设备是二维的。因此，必须将三维物体投影成二维，投影会伴随着相当可观的信息丢失，有时这会在图像上产生模糊性。本章介绍的一些技术可用来将我们在可视环境中通常存在的这类信息找回来。使得人们的深度感知机制能够妥善地解决遗留的歧义性问题。

看一看图14-2a中的Necker立方体幻觉的例子，即一个立方体的二维投影。我们确实不知道它所表达的是图14-2b中的立方体还是图14-2c中的立方体。观察者很容易在二者间捉摸不定。这是因为图14-2a没有包含一个无歧义性解释的足够信息。

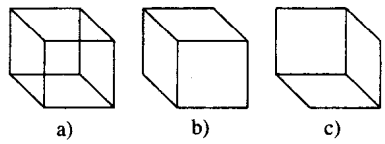


图14-2 Necker立方体幻觉。a)中的立方体相应于b)中的立方体还是相应于c)中的立方体呢

观察者关于显示对象的情况知道得越多，他们就越易于构造Gregory所谓的目标假设[GREG70]。图14-3显示出Schröder阶梯的例子——我们认为它是向下看的阶梯呢？还是由下边看的向上的阶梯呢？我们喜欢选择前一种解释，也许是因为我们更经常地看到阶梯在我们的脚下，而不是在我们的头上，并且因此对从上面观察的阶梯“知道”得更多。然而，只要对想像稍一延伸，我们就能观察到这个图形的另一种解释。可是，只要眼睛一眨，对许多观察者来说就会发生相反的情况，阶梯又再一次表现为从上面观察到的。当然，加上相关的条件（例如一个人站在一级台阶上）将解决这一歧义性问题。

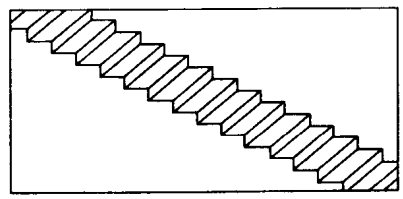


图14-3 Schröder阶梯的例子。这是由上面观察的阶梯还是由下面观察的阶梯

在随后的几节，我们将沿着通向真实感图像的路径列出一些步骤。这条路径实际上已是一组纠缠在一起的小路，而不是单一的笔直的道路。但为了简单起见，我们已经将它线性化了，提供了一个简捷的描述。我们陈述的第一种技术可应用到静态线条图。这些方法集中到在几个物体的二维显示中表示出三维空间关系的途径。接着介绍明暗图像技术，它靠光栅图形硬件来实现，它集中于光线与物体的相互作用。其后讨论逐步增加模型复杂度和动态性的问题，这适用于线条和有明暗的图片。最后，我们讨论真三维图像的可能性、显示硬件的进展，以及在完全交互的环境合成中图片生成的未来地位问题。

14.3 线条图的绘制技术

本节中，我们集中讨论真实感问题的一个子目标：在二维平面上显示三维的深度关系的问题。第6章所定义的平面几何投影可服务于此目标。

14.3.1 多正交视图

最容易产生的投影是平行正交投影。例如，平视图和立视图，其中的投影平面是垂直于一个主轴的。由于抛弃了深度信息，平视图和立视图通常是一起显示的，如图14-4中一个“L”型字母块的顶视图、前视图和侧视图。这种具体画法不难。然而，从一组这样的视图理解复杂的机械零件图也许需要研究好几个小时。当然，训练和经验会提高人的解释能力，对所表达的物体类型的精通可促进

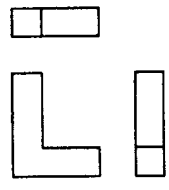


图14-4 “L”型字母块的顶视、前视和侧视正交投影

一个正确的对象假设的形成。但是，像彩图II-21中所示的“标准场景”那样复杂的场景，在只显示三个这样的投影时，往往也是令人困惑的。虽然，单个点可以无歧义性地从三个互相垂直的正交投影来定位，但做这种投影时，多个点和线也许会一个掩盖另一个。

607 ~ 609

14.3.2 轴测投影和斜投影

在轴侧投影和斜投影中，一个点的z坐标影响投影中它的x和y坐标，正如彩图II-22说明的那样。这些投影提供恒定的透视缩小，因此，缺少透视投影所提供的那种平行线的汇聚以及伴随距离增加而出现的物体尺寸的减少。

14.3.3 透视投影

在透视投影中，一个物体的大小与它和观察者的距离成反比。图14-5中所示的一个立方体的透视投影反映了这种比例。然而，仍然有岐义性问题。这个投影好像是一个像框，或者是被削除顶部的金字塔的平行投影，或者是两个面相等的矩形六面体的透视投影。如果假想的物体是一个被除顶部的金字塔，那么，较小的方形表示更接近于观察者的面；如果假想的物体是一个立方体或矩形六面体，那么，较小的方形表示远离观察者的面。

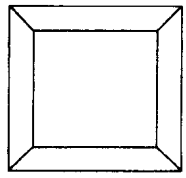


图14-5 一个立方体的透视投影

我们对透视投影的解释往往是基于这样的假定，即较小的物体在较远的地方。在图14-6中，我们或许假设较大的房子更接近于观察者。然而，至少当没有其他线索暗示时，例如，没有树或窗户时，看起来较大的房子（或许是一个大楼）实际上可能比一个看起来较小的房子（如一个小别墅）的距离更远。当观察者知道被投影的物体有许多平行线时，透视投影会进一步帮助传达深度信息，因为平行线会汇聚到它们的灭点。与尺寸减小的效果相比较，这种汇聚实际上是一种更强的深度信息。彩图II-23显示了一个标准场景的透视投影。

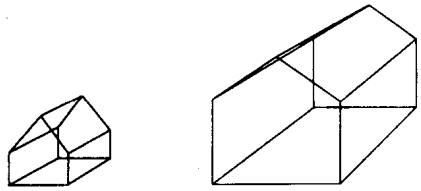


图14-6 两幢房屋的透视投影

14.3.4 深度提示

一个目标的深度（距离）可以由图像的亮度来表示：有意使距离观察者较远的那部分物体以较低的亮度显示（见彩图II-24）。这种效果就是所谓的深度提示。深度提示利用了这样的事实：远的目标比近的目标显得更加朦胧，尤其是穿过薄雾观察看时更是如此。作为空中透视，画家用亮度变化（以及纹理、清晰度和颜色）描绘距离，这样的效果是令人信服的。因此，深度提示也可以看成是大气散射效果的一个简化形式。

610

在向量显示中，深度提示是沿一条向量进行光线亮度的插值来实现的，插值是以该向量起点和终点z坐标的函数来进行的。彩色图形系统经常推广这一技术来支持在图元的色彩与用户指定的深度提示色彩之间进行插值，后者一般是背景色。为了将效果限制在有限的深度范围内，PHIGS + 允许用户指定前、后两个深度提示平面，在它们中间存在深度提示。按每个平面分别赋给比例因子，表征用于前平面之前和后平面之后的原始颜色的比例和深度提示颜色：两平面之间点的颜色在这两个数值之间进行线性插值。肉眼的亮度分辨率低于空间分辨率，所以．对于精确地描绘很小情况的距离差别，深度提示是没有用的。当然，在描绘大的距离差别时，它十分有效，或者在描述小的距离时，它可作为一种放大的提示信息。

14.3.5 深度裁剪

深度裁剪可提供更进一步的深度信息。如彩图II-25所示，安放了一个后裁剪面，切过正在

显示的物体。观察者知道用裁剪平面裁掉了部分物体。也可使用前裁剪面。借助于允许一个或两个平面的位置动态变化，系统可对观察者传达更多的深度信息。后平面深度裁剪可被想像成深度提示的特殊情况：在普通的深度提示中，亮度是z坐标的平滑函数；在深度裁剪中，它是一个阶跃函数。彩图II-25组合了两种技术。一种与深度裁剪有关的技术是借助于在物体上与某些面相交的所有点形成高亮显示。当裁剪平面动态地移过物体时，这种技术特别有效，并且已经用于协助说明沿第四维的深度［BANC77］。

14.3.6 纹理

简单的向量纹理（例如，断面线）可用于一个物体。这些纹理伴随着一个物体的形状并描绘出更清晰的轮廓线。对完全相同的一组面之一加上纹理可澄清一个潜在的歧义性投影。在透视投影中，加纹理特别有用，因为它加上了许多线，其汇聚和透视缩小特性可提供有用的深度提示。

14.3.7 颜色

颜色可用于象征性地将一个物体与另一个物体区别开，如彩图II-26所示，其中每个物体都赋予一种不同的颜色。颜色也可用于线条图，以提供其他信息。例如，一个物体中每个向量的颜色也可由颜色的插值来确定，它是向量端点的温度的编码。

14.3.8 可见线的判定

我们指示的线条图的最后一个技术是可见线判定或隐藏线消除，其结果是在屏幕上只显示可见的（也就是不被遮挡的）线或线的一部分。惟有以边（线）为界的面可能挡住其他的线， 611
这样，挡住其他物体的那些物体必须是面的集合或者是实体。

彩图II-27显示出隐藏线消除的效果。由于移走了隐藏线，视图隐蔽了所有的不透明物体的内部结构，但它们并非显示深度关系的最有效方法。清除了隐藏线的视图提供的深度信息少于剖面视图，隐藏线以虚线显示可能是一种有用的折衷办法。

14.4 明暗图像的绘制技术

14.3节提出的技术可以在向量显示器和光栅显示器中建立线条图。本节所介绍的技术利用光栅设备的性能去显示有明暗效果的区域。当图片是用光栅显示器绘制时，问题是由于光滑的边沿和明暗效果必须在太粗糙的像素栅格上重新产生而引起的。最简单的绘制明暗图像的方法成为走样问题的牺牲品，这一问题最初在3.17节已遇到过。在14.10节，我们将介绍走样的理论背景，并解释如何通过反走样技术来克服走样。由于反走样技术在产生高质量图片中起着基础作用，所以本章中所有的图片都是用反走样技术来建立的。

14.4.1 可见面的判定

与可见线判定相类似，可见面判定或隐藏面消除则仅仅显示观察者可见的那部分面。正如我们曾见过的那样，没有可见线判定也可理解简单的线条图。当有少量的线时，在前面的那些线也许并不是观察它们后面的那些线的主要障碍。在光栅图形中，若表面绘制成不透明的区域，那么，对于制作有意义的图片而言，可见面的判定是至关重要的。彩图II-28显示一个例子，在其中，一个物体的所有面都画成同样颜色。

14.4.2 光照和明暗处理

彩图II-28的一个问题是每个物体都被表示成一个平板剪影。我们走向真实感的下一步是可见面的明暗效果。最终，每个面的外表应取决于照射它的光源的类型、它的特性（色彩、纹理、反射）以及它与光源、观察者和其他面的相对位置和方向。

在许多真实的可视环境中，大量的环境光从所有的方向照射过来。环境光是最容易模拟的

一种光源类型，原因是，在一个简单光源模型中，假设它在所有的面上都产生恒定的光照，不管表面的位置和方向怎样。然而，依靠环境光自身会产生非常不真实的图像，这是因为很少有真实环境是惟一的用均匀的环境光来照射的。彩图II-28是这种方法产生明暗效果的一个例子。

612 *点光源*（它的光线是从单一的点发出的）可近似于一个小的白炽灯泡。一个*定向光源*（它们的光线全来自同一个方向）可用于表示一个远距离的太阳，可将它近似为一无限远的点光源。这些光源的效果依赖于面的方向，所以，模仿它需要增加额外的工作。如果面正交（垂直）于入射光线，它将被照射得很明亮；朝着光线的面越倾斜，照射得越少。当然，光照的变化对一个物体的三维构造是强有力的提示。最后，*分布的或扩展的光源*（它们的表面区域发出光线，例如，日光灯）要模仿它们更加复杂，这是因为它们的光线既非来自单一方向，也非来自单一的点。彩图II-29显示出以环境光和点光源对场景照射，并对每个多边形分别进行明暗处理的效果。

14.4.3 插值明暗处理

*插值明暗处理*是计算明暗信息的一种技术，它计算每个多边形顶点的明暗信息，在多边形的内部则通过插值来决定每个像素上的明暗。当意欲用一个多边形物体来近似一个曲面时，这方法尤其有效。在这种情况下，每个顶点上明暗信息的计算可基于在该点的面的实际方向，并可用于共享这个顶点的所有多边形。在一个多边形内对这些数值进行插值近似于在一个曲面内明暗程度的平滑变化，而不是在一个平面内变化。

即使物体假设为多面体而不是曲面，也可得益于插值明暗技术，其原因在于，对一个多边形的每个顶点计算出的物体明暗信息可以是不同的，但一般说来比曲面物体顶点的差别要小得多。当对一个真实多面体物体计算明暗信息时，对多边形顶点确定的值仅用于该多边形，并不用于共享该顶点的其他多边形。彩图II-30显示出Gouraud明暗处理技术，这类插值明暗技术在16.2节讨论。

14.4.4 材质属性

当确定物体的明暗程度时，若把每个物体的*材质属性*考虑进去将进一步加强真实感。有些材料是较暗淡的，并在各个方向均匀地减弱和分散反射光线，像一支粉笔那样。另一些是有光泽的，并且只在相对于观察者和光源的一定方向上反射光线，比如一面镜子。彩图II-31显示出当某些物体模仿成有光泽的时侯，我们所看到的场景。彩图II-32用的是Phong明暗处理模型，这是一种更精确地用插值生成明暗图像的方法（16.2节）。

14.4.5 曲面造型

虽然插值明暗技术大大改善了图像的外观，但物体在几何上仍是多边形。彩图II-33用了包含曲面的物体模型，在图像的每一像素点上计算全部明暗信息。

14.4.6 改进光照和明暗效果

613 对大多数计算机所生成图像有“不真实”外观的最重要原因之一是，未能精确模拟光线与物体互相作用的许多方面。彩图II-34采用了更好的光照模型。16.7节至16.13节讨论了有效的物理上正确的光照模型设计的进展，产生的图片如彩图III-19至彩图III-29和彩图I-9。

14.4.7 纹理

正如14.3.6节所讨论的，物体的纹理不仅提供了附加的深度提示，而且，还可模拟真实图像的表面细节。彩图II-35和彩图II-36表示出模拟表面纹理的多种方法，从表面色彩的变化（如以图案装饰的球）到表面几何形状的变形（如彩图II-36中的有条纹的环面和扭曲的圆锥）。

14.4.8 阴影

将一个物体的阴影投射到另一个物体，可导致进一步的真实感。请注意，这是在表现一个物体的可视面受另一物体影响时首先会遇到的。彩图II-36显示出由位于场景后方的灯构成的阴影。

阴影增强了真实感并提供了附加的深度信息：如果物体*A*在*B*面上投射出阴影，那么，我们知道*A*是在*B*和一个直接的或反射的光源之间。一个点光源将投射出清晰的阴影，因为从任何一个点，不论它是完全可见的还是不可见的都很明确。一个扩展的光源投射出“柔和”阴影，因为从看到全部光源的那些点起，只见到一部分光源的那些点，直到完全看不见的那些点，有一个平滑地过渡。

14.4.9 透明性和反射

迄今为此，我们只讨论了不透明的面。在图片制作方面，透明的面也是很有用的。透明性的简单模型不包括光线穿过一个透明实体时的折射（弯曲）。然而，若透明度并非用来像显露出物体内部几何形状那样模仿真实性，那么，没有折射可以是一个明显的优点。更复杂的造型包括折射、漫射透明性以及随距离加大的光线衰减。类似地，光线反射模型可模仿一个理想的反射另一个物体的镜面反射模型或者模仿不太光泽的平面的漫反射。彩图II-37显示出地面和茶壶的反射效果；彩图III-7和彩图III-10显示透明性。

像构造阴影一样，构造透明性和反射除了需生成明暗效果的表面以外，还需要其他表面的知识。更进一步，折射透明度是我们曾经提到的第一个效果，它要求实际造型的物体应作为实体而不单是作为表面。我们必须知道有关光线所穿过的材料的某些情况以及正确地模仿折射时，光线所经过的距离。

14.4.10 改进的相机模型

至此，所显示的图片都是基于针孔透镜和无限快的快门这样一种相机模型的。所有物体都是以精确的聚焦并在一瞬间反映出来的。更精确地模仿我们（和相机）看世界的方法是有可能的。 614
例如，借助于透镜焦点特性的模拟，我们可产生彩图II-38和彩图II-39那样的图片，它们显示出*景深*：物体的某些部分在焦点中，那些更近或更远的就在焦点之外。其他一些技术允许特殊效果的应用，例如，鱼眼透镜。许多早期用计算机产生的超现实图片部分地反映出缺乏景深的效果。

以通常的静态或移动相机所取得的照片上的移动物体看起来不同于静止物体。这是由于快门在一个有限的时间内打开，移动物体的可见部分在胶片底板上是模糊的。这种效果称为*运动模糊*，被模仿在彩图III-16和彩图IV-14中。运动模糊不仅在静止中取得运动效果，而且在高质量的动画中也是极为重要的，如第21章所述。

14.5 改进的物体模型

与所用的绘制技术无关，探索真实性的工作也部分集中于建立更可信的模型的方法上，无论是静态的还是动态的。有些研究者已开发出特定类型物体的模型。例如，雾、波浪、山和树；见彩图IV-11~彩图IV-21。其他的研究者还集中精力于大量物体的自动定位，例如，在森林中的树木，靠手工来做它太乏味了（彩图II-25）。这些技术在第20章讨论。

14.6 动力学

所谓*动力学*，指的是在一系列图片中充满着变化，包括位置、大小、材质属性、光照和观察规范的变化等，实际上指任何部分的场景或应用于它的技术的变化。在走向更为真实的静态图像的过程中，动力学的好处是可以独立加以考查的。

如第21章所描述，最普遍的动力学类型是运动动力学，从简单的在用户控制下完成的变换到复杂的动画。从计算机图形学领域出现以来，运动已成为一个重要的部分。在早期的低速光栅扫描图形硬件的时代，运动能力是向量图形系统具有很强竞争力的一个卖点。如果将同一物体的一系列投影（每个都是从同一物体周围稍微不同的视点取得的）快速、连续地显示出来， 615

那么，该物体看起来是在旋转。借助于这些视图的整体信息，观察者可建立一个物体假设。

举个例子，一个旋转的立方体的透视投影可提供几种类型的信息。有一系列不同的投影，它们本身是有用的。在这种情况下，靠近旋转中心的点与远离旋转中心的点相比，其最大线速度较低，这就是靠运动效果来补充的。这种区别可帮助弄清楚一个点离开旋转中心的相对距离。此外，在透视投影下改变距离时，立方体不同部分的大小变化提供了有关深度关系的附加提示。当运动在观察者交互控制下时，它会变得更有用。借助于有选择的变换一个物体，观察者也许有可能更快地构成一个物体的设想。

与用简单的变换去弄清复杂模型相反，若在真实感的形态中运动，令人惊讶地简单造型看起来极有说服力。例如，位于一个人体模型的关键部分的少数几个点自然地运动时，可提供运动中人体的令人信服的例证。这些点本身看起来并不像一个人，但它们确实告诉观察者一个人是存在的。人们很清楚地知道，绘制运动中的物体比表示静态物体需要更少的细节，这是因为观察者更难于拾取出正在运动中的物体的细节。例如，电视观众往往很惊讶地发现单独的一帧电视画面看起来是多么贫乏，并带有颗粒状。

14.7 立体观测

迄今为止，我们曾讨论过的全部技术都是对观察者的双眼呈现相同的图像。现在做一个实验。先用一只眼看桌面，然后，再用另一只眼看。由于我们的一双眼睛相互分开几英寸，两次观察稍有区别，如图14-7所示。由于这种分离产生的双眼视差提供了一个称为立体观测或立体视觉的强有力的深度提示信息。我们的大脑融合这两个分离的图像为一个图像，并被解释成三维的。这两个图像被称为立体对。一个世纪以来，立体对广泛地用于立体观察者，今天已用于普通玩具观察大师（View-Master）。彩图II-19显示一个分子的立体对。你可以在两个图像之间放置一个固定的纸片并垂直于图像面，使你可借助于一只眼看一个图像，从而可以融合两个图像成为一个三维图像。有些人不需要这个纸片也可看到这种效果，而少数人却完全看不到。

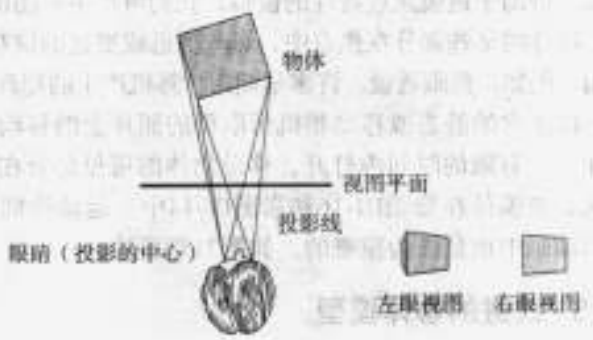


图14-7 双眼视差

存在着对每只眼提供不同图像的其他一些技术，包括具有偏振光滤波的眼镜和全息术。这些技术中，有些可使得占有空间的真三维图像成为可能，而不是被投射在单一的平面上。这些
616 显示可以提供一个附加的三维深度感：正如在真实生活中那样，靠近的目标实际上是更靠近，因而，根据每物体的接近情况，观察者的眼睛可有区别地聚焦在不同物体上。在18.11.5节，会更详细地研究产生和观察立体图像的方法；在习题6.27中描述了立体投影的数学基础。

14.8 改进的显示技术

除了改进用于设计和绘制物体的软件之外，显示技术本身的改善已提高了真实感的幻影。计算机图形学的历史，部分是显示设备所达到的视觉质量稳定提高的历史。还有，现代监视器的色域和动态灰度范围是我们所能看到的两个小子集。在显示器的图像要达到印刷精美的职业摄影照片的清晰度和对比度方面，我们还有一段很长的路要走。有限的显示分辨率使它不可能

复制出极为精美的细节。可见的磷光图像、来自屏幕的刺眼、几何畸变以及帧频闪烁的闪光效果等都是我们正观看着的显示器的各种各样的问题。与我们的视野相比，显示器相对较小的尺寸也提醒我们，显示器只是一个世界的窗口，而不是世界本身。

14.9 与其他感官的交互

走向真实感的最后一步，也许是将真实感图像与呈现给我们的其他感官的信息集成在一起。计算机图形学有一段很长的编程历史，它依赖于多样化的输入设备允许用户交互。飞行的模拟器是图形技术与真实的发动机声音和运动结合的当前实例，所有这些由一个驾驶员座舱的大模型提供，并建立起一个完整的环境。彩图I-16中的头带模拟器监视着头部的运动，使得另一个称为*头运动视差*的重要三维深度提示成为可能：当用户的头部从一边移到另一边时，也许试图去看更多的部分被隐藏的物体，那么视图的变化就像它在真实的生活中应有的那样。头盔显示器的另一个活跃的工作在于对*虚拟世界*的探索。例如，还没有构造好的分子或建筑物的内部 [CHUN89]。

许多现代游乐场的特色是让游戏者乘坐的小汽车或飞机随时间运动，并模拟综合的和数字化的图像、声音和力反馈等，如彩图I-7所示。这种附加的输出和输入形态的使用指出了通向未来的系统的道路，它将提供完全沉浸在包括听觉、触觉、味觉和嗅觉在内的各种感受。

14.10 走样与反走样

在3.17节中我们介绍了走样问题，并讨论了一些生成反走样的二维图元的基本技术。在这里，我们将更详细地研究走样问题，使我们能理解何时以及为何会产生走样，为在随后章节中将反走样合并入可视表面和明暗处理算法打下基础。另外一些材料在[CROW77b；CROW81]中可以找到。一组最好的实例包含在[BLIN89a；BLIN89b]中。

为了理解走样，我们必须介绍来自信号处理领域的一些基本概念。我们从*信号*的概念开始， 617
它是传达信息的一种函数。信号往往表示为时间的函数，但同样也可以是其他变量的函数。由于我们可以将图像想像成覆盖在空间的强度变化。我们可以认为信号是在空域中（作为空间坐标的函数）而不是在时域中（作为时间函数）。虽然图像是两个独立的空间变量（x和y）的二维函数，但是为了方便起见，我们常用单个空间变量x的一维情况作为例子。这种情况可以想像成用一个极薄的片子切割图像，从而表示出沿单一水平线的光照强度。图14-8a和图14-8b显示二维信号，图14-8c和图14-8d显示出沿水平线α的强度图。

信号可根据在空域中是否所有的点都有值来分类。在空间的连续位置上定义的信号称为*连续信号*[⊖]，而在空间中一组离散的点上定义的信号称为*离散信号*。在扫描变换之前，三维物体在视图平面上的投影可处理成一个连续的二维信号。该平面上每个无限小的点的值指示该点的光照强度。与此相反，在图形系统的帧缓存中，像素值的点阵是一个离散的二维信号，其值仅仅定义在该阵列的离散点上。我们的绘制算法必须确定在该阵列上有限数量像素点的光照强度，从而使它们最好地表示出由投影所定义的连续二维信号。然而，“最好地表示”的准确含义并不完全清晰，我们将进一步讨论这个问题。

一个连续信号，在其连续参数变化时，可包含有以非常快的（高频）速度变化的任意精密的细节。由于离散信号只能在离散的点上改变数值，显然，它有一个最大的变化速率。因此，将一个连续信号转换成一个有限的阵列值时，会造成信息的丢失。我们的目标是确保尽可能少地丢失信息，这样，最终的像素阵列可被用来显示一幅图片，这幅图片看起来和我们直接显示它时尽可能相像。从一个信号中选择数值的有限集合的过程就是*采样*，所选择的数值则称为*样本*。一旦我们选择了这些样本，我们必须从那些样本中重新建立原始的连续信号，这一过程称

⊖ 不要与积分中的连续性的定义相混淆。

为重构。帧缓存中的像素阵列是借助于图形系统的显示硬件来实现重构的，它们将这些离散的强度值转换成连续的模拟电压，应用于CRT的电子枪（见第4章）。这种流水线过程的一个理想化的版本见图14-9。信号处理理论[GONZ87]确立了最小的采样频率，这是从一个给定的信号重构该信号的一个精确副本时必须使用的采样频率，并说明了如何完成重构过程。可是，正如我们稍后将看到的，这个最小采样频率对我们所感兴趣的许多类型的信号而言是无限的，因此，理想的重构往往是不可能的。此外，正如14.10.5节中所述，显示硬件通常使用的重构方法往往不同于理论上的方法。因此，即使是适当的采样信号也不会被精确地重构。

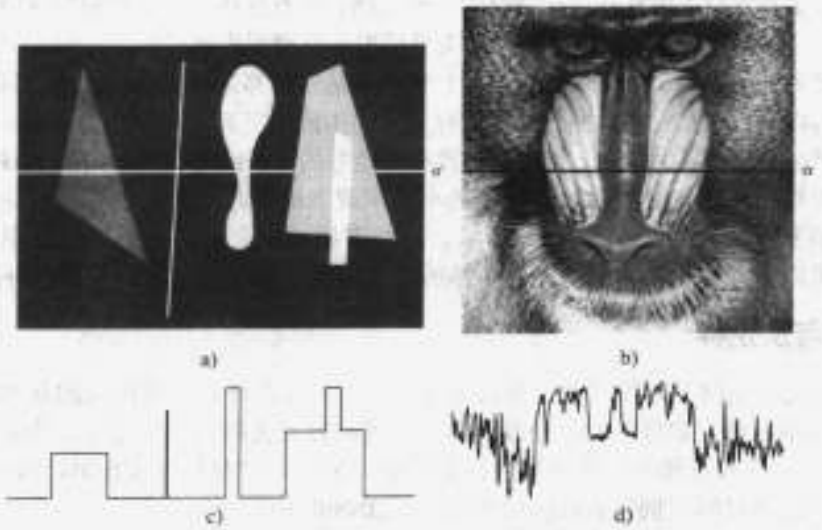


图14-8 图像：a) 图元，b) 山魈，c) 沿a)中扫描线α的强度图，d) 沿b)中扫描线α的强度图。（d部分是哥伦比亚大学George Wolberg提供的。）

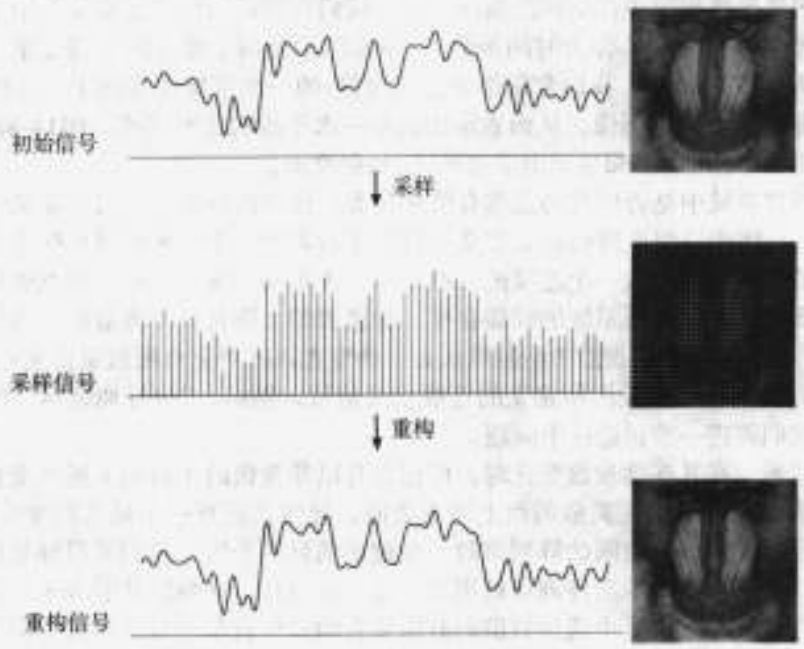


图14-9 对原始信号采样并用这些样本重构该信号（由于点没有面积，所以所采样的二维图像是近似值）。（由哥伦比亚大学George Wolberg提供的。）

14.10.1 点采样

选择每个像素值的最直接方法是所谓的点采样。在点采样中，对应每个像素，我们选择一个点，取这一点上原始信号的值，并将该值赋予那一像素。所选择的点通常是排列在一个矩形网格上，如图14-10所示。不同于第3章的扫描转换算法，投射的顶点不限制于整数的网格点。然而，由于是在一个有限的点集上采样信号的值，也许会丢失信号的重要特征。例如，图14-10中的物体*A*和*C*可由样本表示，然而，*B*和*D*却不能。若观察规范稍有变化或物体移动，事情就更糟了，物体可能从可视范围中跳出或跳入。若我们以更高频率采样又将如何呢？我们从信号中采取到的样本越多，我们对它的了解也越多。例如，可以很容易地看出，若充分增加图14-10中水平和垂直方向的样本数量，我们就能确保在该具体图片中没有丢失物体。对于足够的采样而言这是必要条件，但非充分条件。虽然如此，在更高频率上采样，可以在生成的图像中，用更多像素表示该图片上的每个部分。我们也可用较少的像素生成一幅图像，即把几个相邻的样本综合起来（例如，取平均值）以确定在较小的图像上的每个像素值。这意味着，所有将出现在较大图像上的特征，至少会对较小的图像有所贡献。

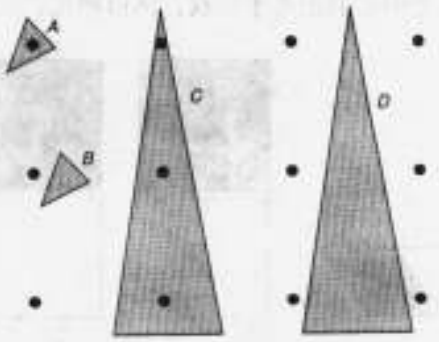


图14-10 点采样问题。样本用点(•)表示。物体*A*和*C*被采样了，但物体*B*和*D*未被采样

对每个像素采用超过一个采样点并将其综合的方法，被称为*超采样*。它能有效地适应重构该信号，并对所重构的信号重新采样。由于稍后将说明的原因，采样重构的信号往往比采样原始信号要好。由于这项技术是如此容易，并且往往有很好的结果，所以，它在计算机图形学中得到普遍采用，尽管在计算量上有明显地增加。但是，多少样本才足够呢？我们怎么知道我们的样本没有丢失特征呢？仅仅检测每个物体的投影是否被采样还是不够的。投影也许有一个复杂的形状或者明暗程度上的变化，而样本并没有反应它们。我们喜欢用某些方法去保证我们的样本对于重构原始信号而言，在空间上是足够密的。我们将会看到，采样理论告诉我们，在一个实际的信号特性的基础上，我们可计算出一个充分的最小采样频率。可惜，对包括图14-10所示的信号的特定类型的信号，所得出的该频率是无限的。稍后，我们将解释更详细的原因。现在，我们可以看到，取一个有限数量的样本不能捕获到图中明暗程度从一个值跳到另一个值剧烈变化处的x坐标。还有，如果我们找到一个有限的采样频率，在这频率上采样所有当前的物体，那么我们总可以做到，在样本之间的位置上增加一个物体，而该物体将完全丢失。

14.10.2 区域采样

“落在样本之间”的物体出现丢失的问题使人想起另一种算法：在以每个网格点为中心的一个正方形上将信息加以积分，除以正方形的面积，用这个平均强度作为该像素的强度。这种技术称为*未加权区域采样*，已在第3章中介绍过。不重迭的正方形阵列一般用来表示一个像素。每个物体的投影（不管多小）对包容它的这些像素的贡献严格地与投影所覆盖的像素总面积成正比，而与那个区域在像素中位置无关，在图14-11a中显示出权值相等的函数。这时，不会出现物体丢失，而在点采样时是会发生的。定积分的定义要求计算一个区间内的许多点的函数，并在点数增加时取其极限。这样，积分相当于一种无限采样的过程。

未加权区域采样具有因这种处理物体的平均性而产生的缺点。请考虑一个小的黑色物体，它整个落在一个像素之内并被白色背景所包围，如图14-11b所示。这个小物体可在像素内部自

由地移动，在每个位置上对该像素所计算的值（像素的明暗值）一样。然而，一旦物体跨入一个相邻的像素，原来像素的值及相邻像素的值同时受到影响。这样，该物体产生的图像只有当它跨越像素边界时才有变化。然而，我们希望当物体从一个像素的中心移动并接近另一个像素的中心时，这种变化能表现在该图像中。换而言之，我们希望物体对该像素强度的贡献能根据它与像素中心的距离来加权，离得越远，贡献越小。

618 - 621

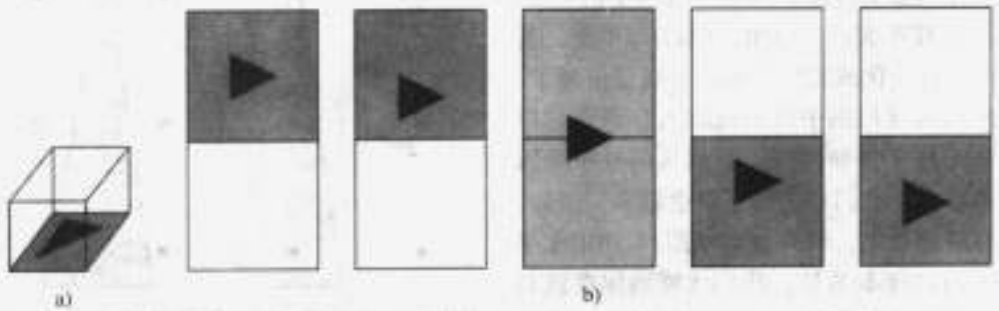


图14-11 未加权区域采样。a)在该像素内所有的点权值相等，b)当一个小物体在像素之间移动时所计算的强度的变化

在第3章，我们说明了加权区域采样允许我们对像素的不同部分赋予不同的权值，并且我们假定邻近像素的权函数应当重叠。为了看清为什么必须重叠，我们考虑一个由覆盖单一像素建立起来的竖直的金字塔构成的权函数，如图14-12a所示。在这种权函数之下，当一个物体离开该像素中心时，它对像素的贡献就减少。但是，未加权区域采样的缺点仍然存在：一个物体仅对包含它的单一像素有贡献。考虑一个子像素大小的黑色物体，覆盖在白色背景上，从一个像素中心移动到邻近像素的中心，如图14-12b所示。物体从第一个像素中心移出来时，随着它靠近其边界，它对第一个像素的贡献减少。仅当它跨越边界之后，才开始对它进入的像素有贡献，并且，当它到达新的像素中心时，其贡献达到最大值。这样，即使黑色的物体有恒定的亮度，在第二个像素亮度增加之前，第一个像素亮度将减少。其净效果是显示亮度的变化取决于物体的位置，即当物体在屏幕上移动时，这种变化将导致闪烁。很明显，为了改正这个问题，必须允许我们的权函数有重叠，这样一来，在物体上的一个点可以同时影响多于一个像素，如图14-13所示。这个图使用了一种径向对称的权函数。在此，需要回到采样理论去揭示增加权函数大小的基本原因，并准确找出为了采样和重构一个信号，我们需要做什么。

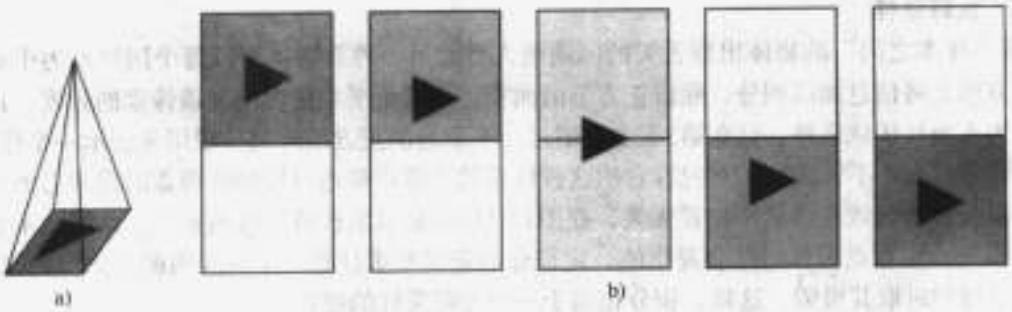


图14-12 加权区域采样。a)像素中的点有不同的权值，b) 当一个物体在像素之间移动所计算的亮度的变化

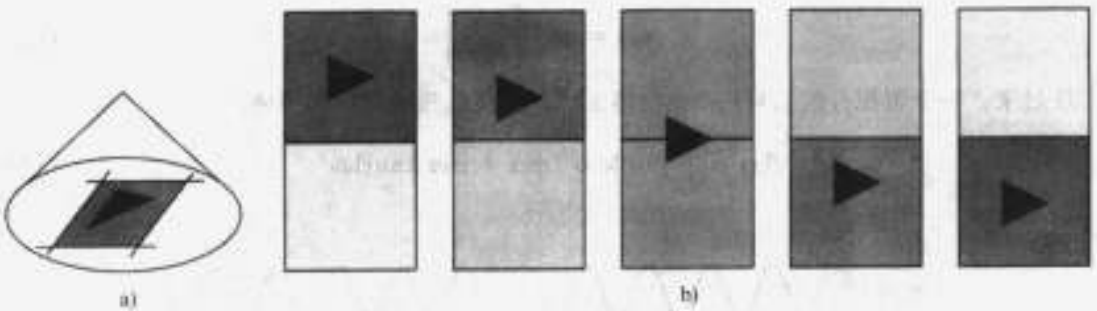


图14-13 重叠的加权区域采样。a)典型的权函数，b) 当一个目标在像素之间移动时所计算的亮度的变化

14.10.3 采样理论

采样理论提供一个精致的数学框架以描述连续信号与其样本之间的关系。到目前为此，我们已考虑了空域中的信号，也就是说，我们已将它们的每一个表达成相对于空间位置的幅值图。一个信号也可在频域中加以考虑，也就是，我们可将它们表达成正弦波的迭加，彼此之间也许有偏移（这种偏移称为相位移），并具有不同的频率和幅度。每个正弦波表示该信号频谱的一个成分。在空域中将每一点的值迭加起来，就可以将这些成份在空域中相加。

周期信号（如图14-14所示）可表达成有相位移的正弦波的迭加，正弦波的频率是该信号基频的整数倍（谐波）。但是对于图像这样的非周期信号，又将如何呢？由于一个图像的大小有限，我们可以将图像区域之外的信号定义为具有一个零值。这样一个信号，在有限区域中它是非零值，或更一般地说，任意一个衰减得足够快的信号$f(x)$（对于大的x值，快于$1/x$），也可以表达成有相位移的正弦波的迭加。然而，其频谱不是由某些基频的整数倍所组成的，可能会包含任意的其他频率。原始信号不能被表达成可数的许多正弦波的迭加，而必须表示为一段连续频谱的积分。但是通常将一幅图像（或许带上周围用来填充的零）处理成周期信号的一周期。图14-14b就是这样的，它显示出图14-8d的前10个成份。空域中的每个信号都有一个频域里的表示，反之亦然。稍后我们将会看到，对一个信号用两种表示是有好处的，因为在一个域中很难实现的某些有用的操作在另一个域中相对说来是容易实现的。

622
|
623

确定用什么样的正弦波来表达一个具体的信号是傅里叶分析的中心课题[GONZ87]。从一个原始信号$f(x)$开始，我们可以产生一个不同的函数，即f的傅里叶变换，称为$F(u)$，其参数u代表频率。对每个频率u，$F(u)$之值指示在原始信号$f(x)$中频率为u的成份有多大（即其幅度）。函数$F(u)$也称为f（或信号）在频域中的表达式；$f(x)$本身称为该信号在空域中的表达式。一个连续的可积分信号$f(x)$从空域到频域的傅里叶变换定义为

$$F(u) = \int_{-\infty}^{+\infty} f(x)[\cos 2\pi ux - i\sin 2\pi ux]dx \tag{14-1}$$

其中$i = \sqrt{-1}$，u表示一个正弦和余弦对的频率。（请注意，该式只适用于衰减足够快的函数。）要注意余弦正好是相位移动$\pi/2$后的正弦。它们在一起可用于确定该频率成份的幅度和相位移。对于每个u，$F(u)$的值都是一个复数。这是对信号频率为u的成份的相位移和幅度的一种很清楚的表示方法：$F(u)$的值也可写成为$R(u) + iI(u)$，其中$R(u)$和$I(u)$分别为实部和虚部。$F(u)$的幅值（或幅度）定义为：

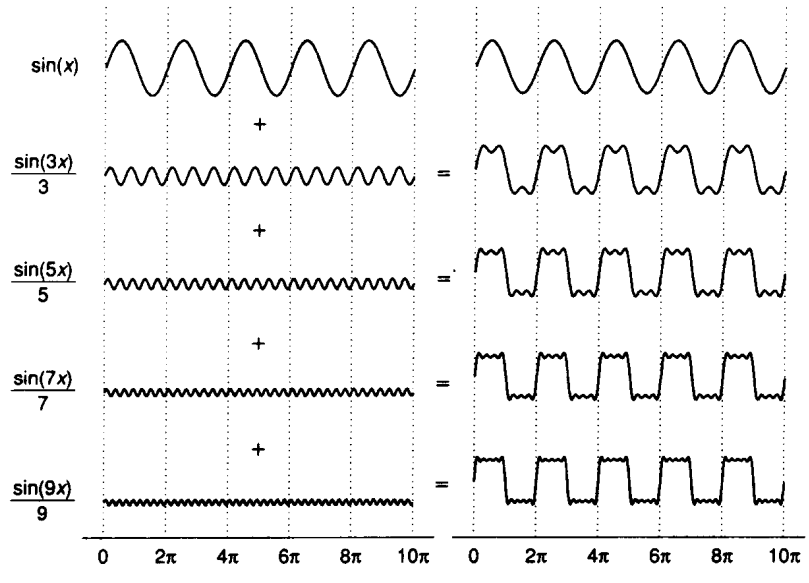
$$|F(u)| = \sqrt{R^2(u) + I^2(u)} \tag{14-2}$$

其相位移（也就是相角）为

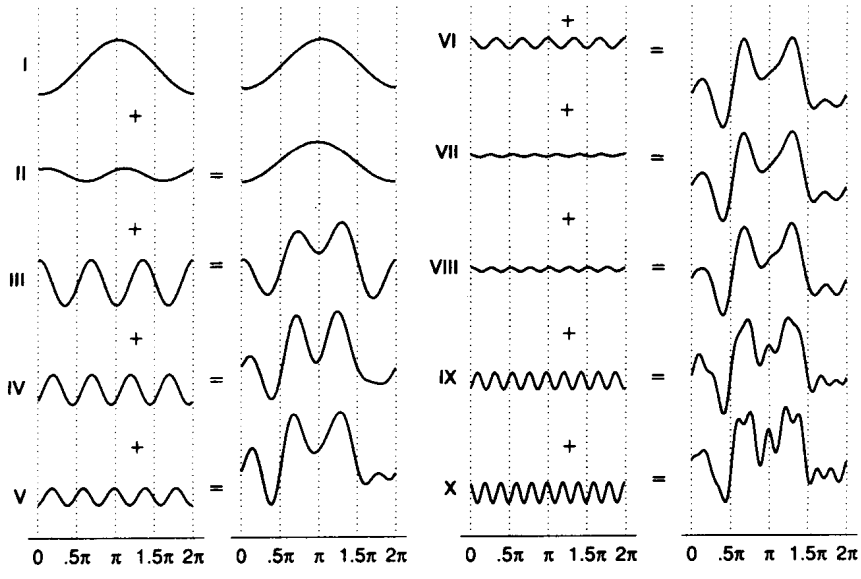
$$\phi(u) = \tan^{-1}\left[\frac{I(u)}{R(u)}\right] \tag{14-3}$$

反过来，一个可积分的信号$F(u)$也由傅里叶逆变换从频域变换到空域:

$$f(x) = \int_{-\infty}^{+\infty} F(u)[\cos 2\pi ux + i\sin 2\pi ux]du \tag{14-4}$$



a)



b)

图14-14 空域中的一个信号是带相位移的正弦波的迭加。显示每个成份，并同时在其右边显示它对该信号的影响。a) 一个方波的近似表示，b) 图14-8d的近似表示（哥伦比亚大学George Wolberg提供。）

一个信号的傅里叶变换往往画成幅值对频率的关系图，而忽略其相位角。图14-15给出了一些信号在两个域内的表示。在空域中，我们以像素中心的数目来标记横坐标；在频域中，我们

以每个像素的周期数（或者更准确地说，像素中心之间的每个区间的周期数）来标记横坐标。在每一种情况下，在$u = 0$处的峰值代表频谱的DC(直流)分量。将cos 0 = 1和sin 0 = 0代入式(14-1)中，显示出这相应于$f(x)$的积分。若从图14-15a或图14-15b中$f(x)$的每个值减去0.5，则信号的直流分量幅值应是0。

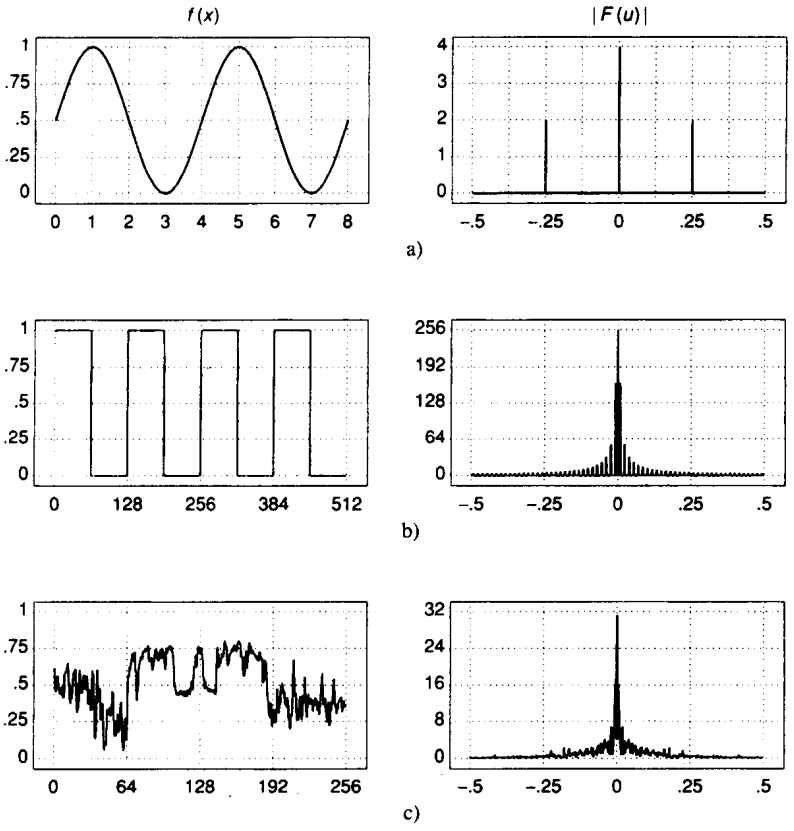


图14-15 空域和频域中的信号。a)正弦，b)方波，c)山魈。在频域中的DC值被削平了，使得其他的值易读，而且应该是129。（由哥伦比亚大学George Wolberg提供。）

在本章中，显示信号及其傅里叶变换的大多数图是用计算式(14-1)和式(14-4)的离散形式计算出来的，信号是以N个具有规则间隔的样本来表示的，离散傅里叶变换是：

$$F(u) = \sum_{0 \le x \le N-1} f(x)[\cos(2\pi ux/N) - i\sin(2\pi ux/N)],\ 0 \le u \le N-1 \tag{14-5}$$

离散傅里叶逆变换是：

$$f(x) = \frac{1}{N}\sum_{0 \le u \le N-1} F(u)[\cos(2\pi ux/N) + i\sin(2\pi ux/N)],\ 0 \le x \le N-1 \tag{14-6}$$

对于大多数信号，通过选择一个足够高的采样频率可以很好的近似连续傅里叶变换的行为。（借助于使用一种很灵巧的快速傅里叶变换公式[BRIG74]，离散傅里叶变换也可以比用式(14-5)和式(14-6)计算得更快。）这个离散傅里叶变换总是产生一个有限的频谱。请注意，若信号关于原点对称，则$I(u) = 0$。这是确实的，因为在原点一边的每个正弦项的贡献与另一边的贡献相等且相反。在这种情况下，按照[BLIN89a]，我们将画出带符号的函数$R(u)$，而不是$|F(u)|$的幅值。

624
~
626

采样理论告诉我们如果一个原始信号以大于其频谱中最高频率成份f_h的两倍的频率采样，那么，该信号可以很好地从其样本重构出来。采样频率的这一下限称为奈奎斯特频率。虽然，我们不给出高于奈奎斯特频率采样的合理性的形式化证明，但我们可以提供一个非正规的解释。考虑

一个最高频率成份为f_s的信号中的一个周期。如图14-16所示，这个分量是一个正弦波，具有f_s的最大值和最小值，因此，至少要求$2f_s$个样本，才能取得信号的最高频率成份的全部形状。请注意，实际上，严格的$2f_s$个样本是一种特定情况，只有当采样点是精确地位于最大和最小点上才是成功的（图14-16a）。如果它采在任何其他地方，那么，其幅值将不能被正确地表达（图14-16b），并且，若在过0处采样，则甚至可能认为幅值为0（图14-16c）。若我们以低于奈奎斯特频率采样，则所采得的样本也许和从较低频率信号采样得到的一样，其实例见图14-17。这种在重构信号时高频信号被伪装成低频信号的现象称为走样：高频成份看起来好像是低频的成份。另一种走样的例子示于图14-18。图14-18a显示一幅图像和一条扫描线上的亮度图，表达了一组其空间频率从左到右增加的亮度变化。图14-18b是从图14-18a的每一条线上每8个像素选择一个来建立的，并重复8次。它显示出当空间频率增加时，该条带走样。

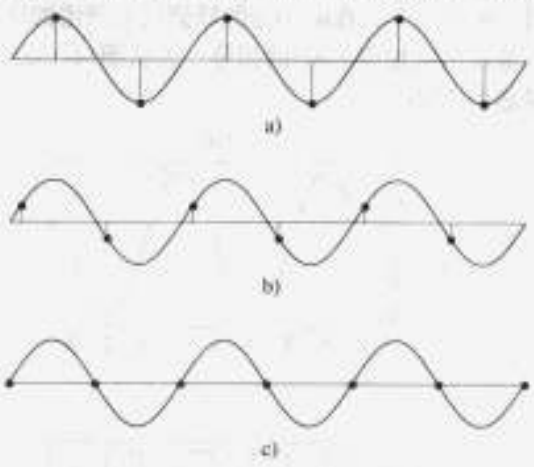


图14-16 以奈奎斯特频率采样，a)在顶点，b)在顶点之间，c)在过0处。（由哥伦比亚大学George Wolberg提供。）



图14-17 以低于奈奎斯特频率采样。（哥伦比亚大学George Wolberg提供。）

一个信号的形状是由其频谱决定的。波形越尖、角度越大，则其高频率成份越丰富；带有不连续性的信号具有一个无限频率的频谱。图14-10表示出我们的算法中试图去表示的物体投影的尖锐边界。由于图像的亮度在物体边界上有不连续的改变，该信号具有一个无限频谱。因而，不能根据有限的样本数量来适当地表达该信号。因此，计算机图形学生成的图像展示出两种主要类型的走样。第一，在物体的投影边上由于不连续造成的“锯齿状”：一个点的样本要么落在物体的投影内，要么不落在物体的投影内。在一个场景的投影中，即使出现一条这样的边，也意味着该投影有一个无限频谱。然而，就像图14-15b和14-15c一样，该频谱衰减得很快。第二，用透视看纹理和物体也可在场景的投影中产生许多不连续性和波动，当物体的投影太小或太接近时有可能交替地丢失或被采样，正如图14-18的右边所示的那样。其高频成份表示出这些投影跨过一根扫描线的频率，可以有高的幅值（例如，交替的黑白色棋盘格）。这往往比“锯齿状”对图片质量的影响更为严重。

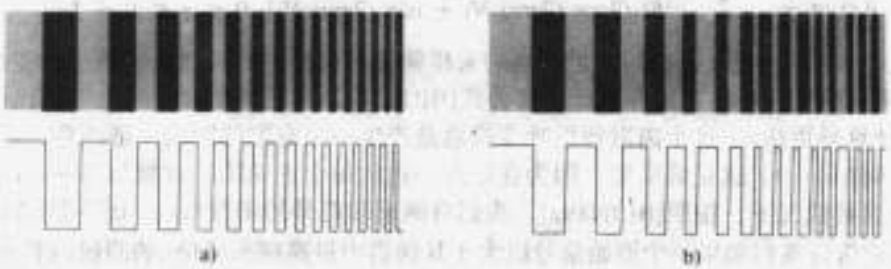


图14-18 走样。a)图像和一条扫描线的亮度图，b)所采样图像和一条扫描线的亮度图。（由哥伦比亚大学George Wolberg提供。）

14.10.4 滤波

627 ↓ 628

对前面一节所讨论的问题可有部分的答案。如果我们能够从原始信号中去掉不利的高频部分而产生新的信号，那么，我们便可以从一组有限数量的样本很好地重构新的信号。去掉的较高频率部分越多，需要的采样频率就越低，但是该信号与原始信号的相似性就越少。这种过程被称为信号频带宽度限制，也称之为低通滤波，因为一个信号滤波后，改变了它的频率谱，在这种情况下，高频被滤掉了，只允许低频通过。由于精密的视觉细节是用高频获取的，而它们被低通滤波器拿掉了，因而低通滤波在空域中产生了模糊，如图14-19所示。我们应该修正图14-19的流水线，加进一个供选择的滤波器，如图14-20所示。



图14-19 低通滤波后的图14-8b。（哥伦比亚大学George Wolberg提供。）

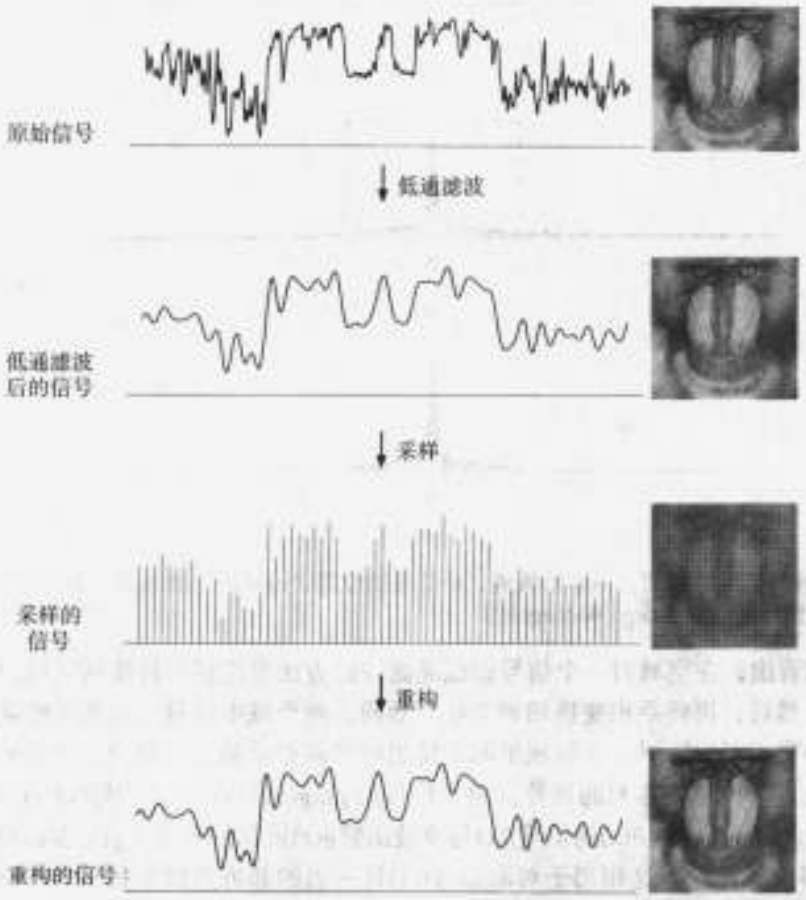


图14-20 带有滤波的采样流水线。（哥伦比亚大学George Wolberg提供。）

一个完美的低通滤波器可完全抑制所有高于某个规定的截止点的频率成份，并让低于该截止点的频率畅通无阻。用一个脉冲函数乘以信号频谱可以很容易地在频域中实现这样的滤波，如图14-21所示。我们可取一对信号在每一点的乘积来进行两个信号相乘。脉冲函数

$$S(u) = \begin{cases} 1, & -k \leq u \leq k \\ 0, & \text{其他} \end{cases} \tag{14-7}$$

可截止所有高于k的频率。因此，若低通滤波是将信号中所有的变化量都消除，则将只留下直流分量。

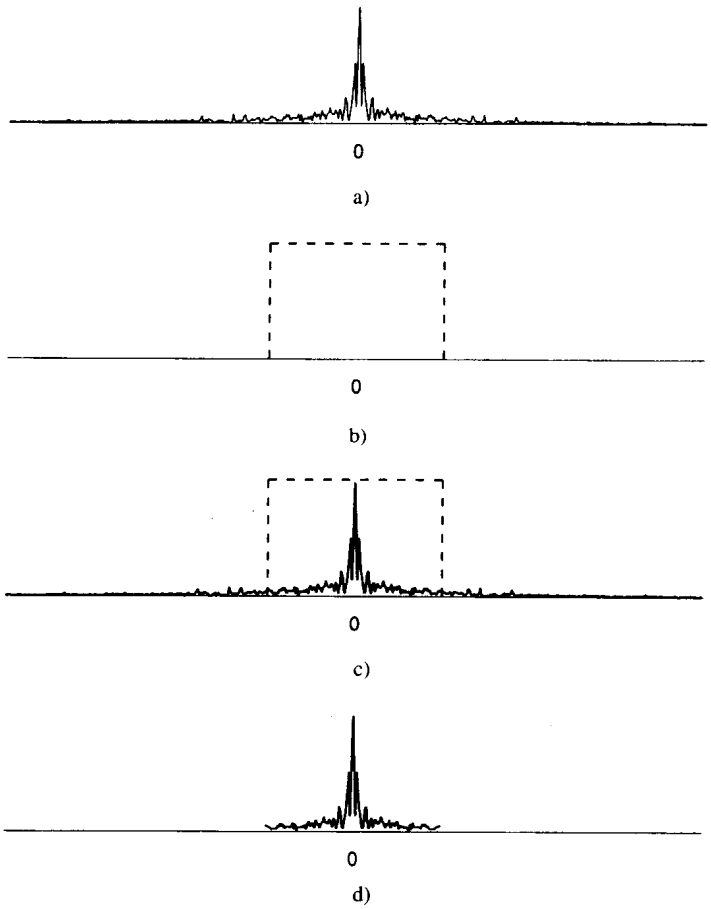


图14-21 频域的低通滤波器。a)原始频谱，b)低通滤波器，c)频谱与滤波器，d)滤波后的频谱（哥伦比亚大学George Wolberg提供。）

至此，可以看出，在空域对一个信号做低通滤波的方法是把信号转换到频域，将它乘以适当的脉冲函数，然后，再将乘积变换回到空域。然而，两个域中信号之间的某些重要关系使得这一过程没有必要。可以证明，在频域里两个傅里叶变换相乘精确地对应于在空域中对它们的傅里叶逆变换执行一种称为卷积的运算。两个信号$f(x)$和$g(x)$的卷积，写成$f(x)*g(x)$，被定义为如下的新信号$h(x)$。每一点上$h(x)$的值是$f(x)$与滤波函数$g(x)$乘积的积分，$g(x)$需相对于纵轴做反转并将其原点移位到该点。这相当于对取信号$f(x)$每一点的邻近点做加权平均——以位于该点的滤波函数$g(x)$的反转副本加权，并将它用于该点的$h(x)$的值。邻域的大小由滤波器非零域的大小确定。这就是所谓滤波器的支集（support），当一个滤波器的非零域有限时叫作有限支集。当定义卷积时，我们用τ作为一个积分变量。于是有：

$$h(x) = f(x) * g(x) \triangleq \int_{-\infty}^{+\infty} f(\tau)g(x - \tau)d\tau \tag{14-8}$$

629
:
631

与此相反，频域里两个傅里叶变换的卷积精确地对应于在空域里将它们的傅里叶逆变换相乘。滤波函数常称为卷积核或滤波器核。

可用图形来说明卷积。我们以函数$f(x)=1(0\leqslant x\leqslant 1)$与滤波核$g(x)=c(0\leqslant x\leqslant 1)$做卷积，如图14-22a和图14-22b所示。用τ的函数，我们可以改变x，从而相对滤波信号移动滤波器。为产生函数$g(x-\tau)$，我们首先相对于原点反转$g(\tau)$生成$g(-\tau)$，然后，将它按x做偏移成为$g(x-\tau)$，如图14-22c和图14-22d所示。乘积$f(\tau)g(x-\tau)$对τ的积分是图中阴影部分的面积，对于$-\infty\leqslant x<0$是0，对于$0\leqslant x\leqslant 1$是xc（图14-22e），对于$1\leqslant x\leqslant 2$是$(2-x)c$(图14-22f)，而对于$0<x\leqslant\infty$是0，卷积$f(x)*g(x)$的图示说明在图14-22g中，请注意，具有这样的核的卷积怎样平滑了$f(x)$的不连续性，它加宽了$f(x)$的非零部分所覆盖的面积。

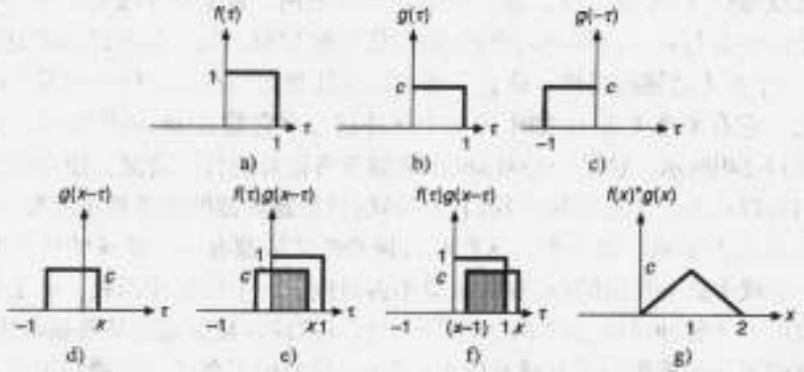


图14-22 用图形表示卷积。a)函数$f(\tau)=1,0\leqslant\tau\leqslant 1$，b)滤波核$g(\tau)=c,0\leqslant\tau\leqslant 1$，c) $g(-\tau)$，d) $g(x-\tau)$，e) $\int_{-\infty}^{\infty}f(\tau)g(x-\tau)d\tau=xc,0\leqslant x\leqslant 1$，f) $\int_{-\infty}^{\infty}f(\tau)g(x-\tau)d\tau=(2-x)c,1\leqslant x\leqslant 2$，g)$f(x)*g(x)$

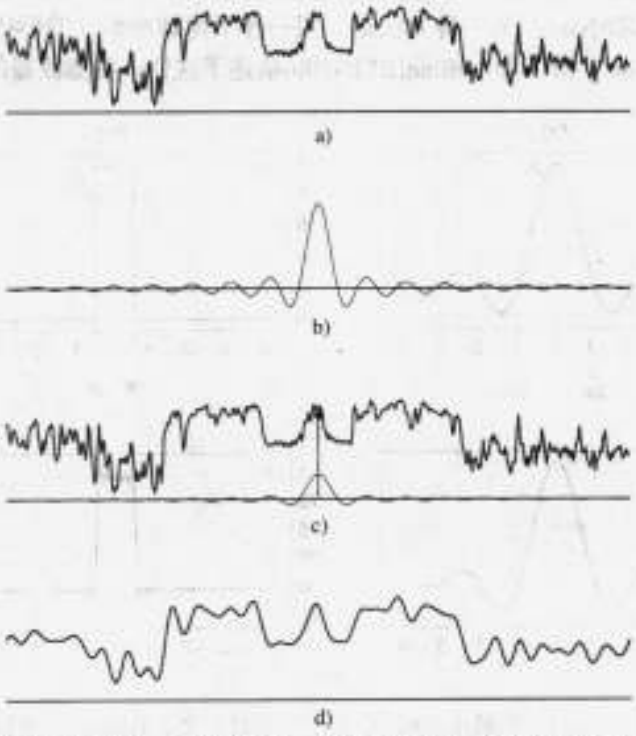


图14-23 空域中的低通滤波器。a)原始信号，b)Sinc滤波器，c)信号与滤波器，d)滤波后的信号。（由哥伦比亚大学George Wolberg提供。）

在频域中乘以一个脉冲函数与在空域中以对应于该脉冲的信号做卷积有相同的效果。这个信号是所谓的sinc函数，定义为$\sin(\pi x)/\pi x$。图14-23显示sinc函数以及它对另一个信号做卷积的结果。用一个sinc函数的卷积就是把该信号做了低通滤波。我们如何选择用于图14-23c中的sinc信号的高度和宽度呢？如图14-24所示，在空域和频域中，理想的低通滤波器的高度和宽度之间有一个关系（我们不进行证明）。在图14-24a中，若W是截止频率，A是幅度，那么，为使截止频率以下的所有频率都无阻地通过，应有$A/2W = 1$。因此，$A = 2W$。在图14-24a中的sinc的幅度和宽度随W而变化。当$W = 0.5$周期/像素（当每个像素采样一次，可达到的最高频率）时，A = 1，同时，跨在像素中心的sinc为0。当截止频率W取得更低或更高时，sinc变得更矮且更宽或更高且更窄。造成这种情况的原因是我们希望在空域中一个滤波器的积分为1。若滤波器用于维持图像的灰度等级（DC值）时，这是一个必要的限制，使它既不变亮，也不变暗。（我们考虑一下，一个滤波器与每一点都有同样的c值的信号做卷积，我们就可以知道这是真实的。）

sinc函数有一个令人遗憾的特性，这就是在离原点任意远的点上它仍不为零（也就是说，由于它是无限宽的，它有*无限支集*）。如果用一个脉冲信号乘它截去sinc函数顶端，我们可以限制这种支集，如图14-24b所示。这是一种对sinc函数做*开窗*运算的特殊情况，使它限制在一个有限的窗口中。我们可以认为，无论如何，我们丢弃掉的只是滤波器的值非常小的那些部分，所以，它不应当对结果有太大影响。很不幸，这种截去顶端的滤波器有一种*振荡*的傅里叶变换（也称为Gibbs现象）：空域中截去顶端的sinc在频域里不再对应于一个纯脉冲函数，而是对应于靠近截止频率处有波纹的一个脉冲函数，如图14-24b所示。这导致应该被抑制的某些频率成份得以通过，还对截止点附近的另一些成份做了衰减及放大；当sinc信号中更多的部分被利用时，这个效应所覆盖的区域在大小上减小，但只要是sinc被截去顶端，其振荡的幅度并未减弱。图14-14a中的一个近似方波展现出在空域中的振荡，它表现为不连续的小强度的“波纹”。由一个脉冲函数乘以sinc可得到一个截去顶端的sinc。另一种方法是，用一种不像脉冲那样的形状的连续函数相乘形成开窗的sinc，它允许sinc平滑下降。Blinn[BLIN89b]描述了这样一种滤波器的由来。

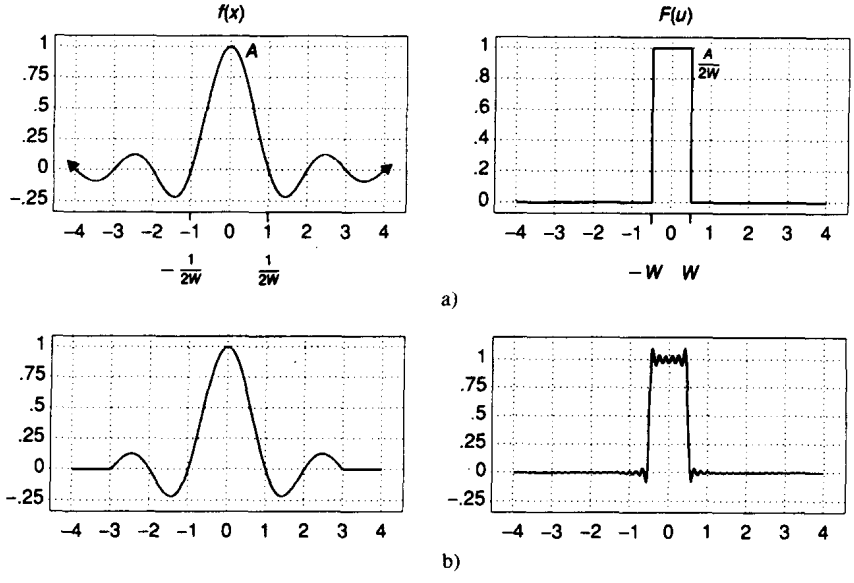


图14-24 a)空域中的sinc对应于频域里的脉冲，b)空域中截去顶端的sinc对应于频域里具有振荡的脉冲。（哥伦比亚大学George Wolberg提供。）

最后一个问题是，sinc以及由它导出的开窗滤波器，有一部分会下降到零值以下，所谓负波瓣。当一个信号与有负波瓣的滤波器做卷积时，其结果信号本身也会下降到零值以下。若信号表示的是强度值，这些值对应于非真实的负强度，因而最终必须强制在零。

虽然开窗的sinc函数是有用的，但由于窗口必须相当宽，相对说来开销较大。因此，常用多种其他函数来代替它。有限支集的滤波器就是所谓有限脉冲响应（FIR）滤波器，与顶端不截断的sinc滤波器相反，它是无限脉冲响应(IIR)滤波器。图14-25显示一些在空域和频域中常用的滤波器。

632
~
634

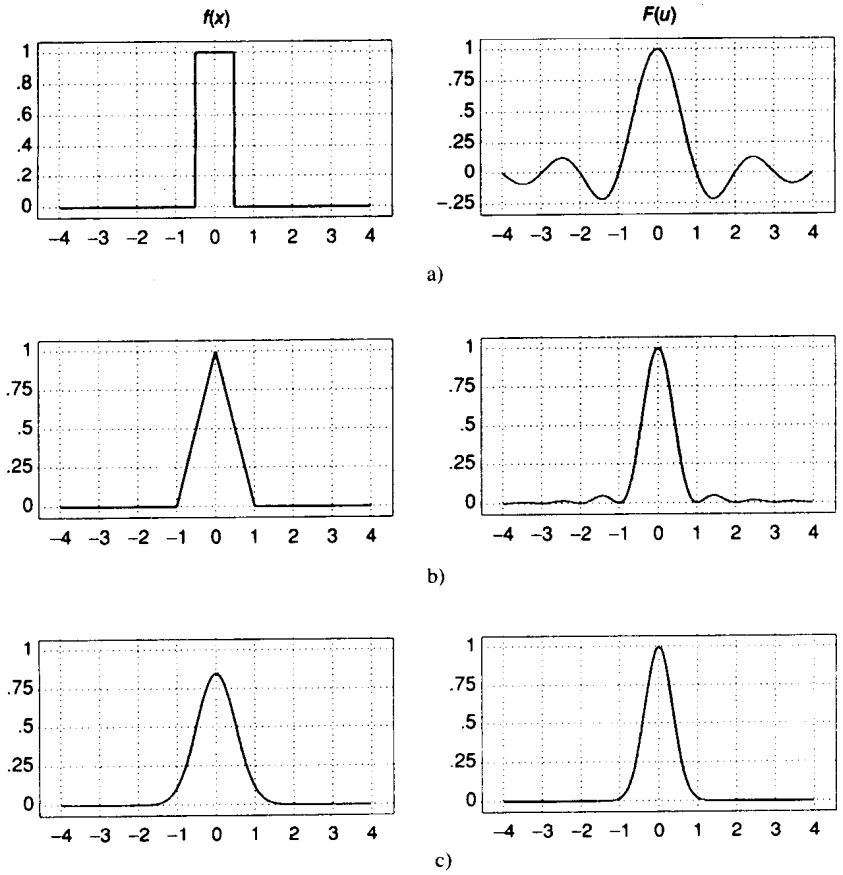


图14-25 空域和频域中的滤波器。a)脉冲——sinc，b)三角波——$sinc^2$，c)高斯——高斯。（哥伦比亚大学George Wolberg提供。）

现在，我们将采样问题变化为用一种合适的滤波器与信号卷积，然后，再对滤波后的信号进行采样的问题。然而，请注意，若我们只用滤波后的信号采样，那么，在任何地方完成的信号滤波（除了采样点处的滤波外）都是无用的。若我们事先更精确地知道在哪儿取样本，我们只需要在每个采样点处计算卷积积分（式(14-8)）以确定样本的值。使用区域采样中的加权操作来确定每一像素点的亮度就是这样的。构成覆盖每个像素中心的权值分布是一个滤波器。在实现不加权区域采样中与信号做卷积的脉冲函数，由于其外观原因，经常称为盒式滤波器。正如在频域中的脉冲函数对应于空域中的sinc函数一样，这个空域中的脉冲函数（盒式滤波器的一维等价物）对应于频域中的sinc函数（图14-25a）。这种对应性着重说明了，用一个盒式滤波器或脉冲滤波器去近似一个精确的低通滤波器是怎样的不好。在频域中，与一个sinc相乘不仅不能急剧地截止，还

635

会通过无限的高频分量。此外，脉冲滤波器使那些在要求范围内的频率衰减了，因为其傅里叶变换——sinc函数——在理想的低通滤波之前就开始减弱了。因此，它也会使图像过分模糊。

14.10.5 重构

在此，假定我们已用频率f_s采样了信号$f(x)$而得到被采样的信号，称之为$\hat{f}(x)$。采样理论表明$\hat{f}(x)$的频率特性看起来与$f(x)$的类似，在f_s的倍频位置重复出现。为看清这层关系，我们注意到采样一个信号相当于在空域中以一个梳状函数去乘它，之所以起这样的名称是由于该函数所显示的形态，如图14-26a所示。这个梳状函数除了在对应于采样点的规则间隔处值为1之外，其余的地方其值为0。这个梳状函数的(离散)傅里叶变换是在f_s的倍数处具有尖齿的另一个梳状函数，如图14-26b所示。梳状函数傅里叶变换的尖齿高度是f_s周期/像素，由于空域中的乘积对应于频域中的卷积，我们可由梳状函数的傅里叶变换与原信号傅里叶变换（图14-26c）的卷积来得到采样信号的傅里叶变换。经检查，该结果是复制的频谱（图14-26d）。试执行此梳状函数的卷积以检验这一结果，需注意的是应该用$F(u)$，而不是$|F(u)|$，足够高度的f_s产生的频谱将在彼此相离较远处出现重复，在极限情况下，当f_s接近无穷大时，将产生单一的频谱。

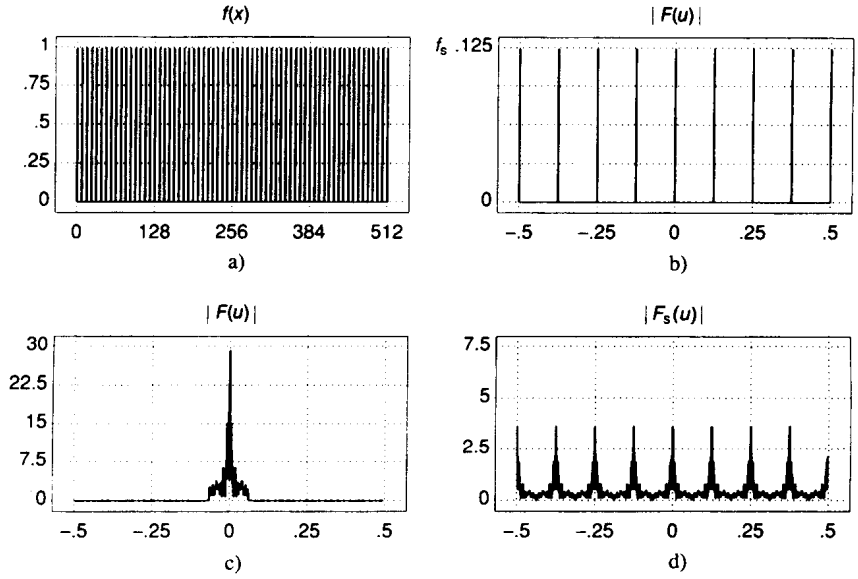


图14-26 a)梳状函数；b)傅里叶变换；由梳状函数的傅里叶变换与原始信号傅里叶变换c)做卷积生成采样信号的样本信号重复频谱d)。（由哥伦比亚大学George Wolberg提供。）

重构乃是从其样本重新产生原始信号的过程。在有限的采样频率下采样一个信号的结果（图14-27a）是一个具有无限频谱（图14-27b）的信号。若我们再一次在频域中对此信号做处理，以一个脉冲函数乘这个信号，可用于剔除这些重复的谱线（图14-27c），只留下原始频谱的一个副本（图14-27d）。这样，我们可以在频域中以一个脉冲函数乘以样本的傅里叶变换或是在空域中将该样本与$A=1$的sinc做卷积以重构该信号。

为了使信号和滤波器的傅里叶变换更容易在图14-27至图14-29中看清楚，我们做了以下假设：

- 每个图中(a)部分的傅里叶变换里的直流已去掉。这对应于空域中具有同样形状的信号，但具有负的直流偏移。（这样的一个信号不做进一步的处理是不能显示成一幅图像的，因为它包含负的亮度值。）
- 频域中的滤波器并没按正确的大小画出。为了将频谱的单个副本恢复成原始的幅值，其高度在图14-27中应是1，在图14-28至图14-29应是2。

图14-27e和14-27f显示出以三角滤波器（也就是所谓的Bartlett滤波器）重构该样本的结果。用这样的滤波器做卷积等同于对样本进行线性插值。

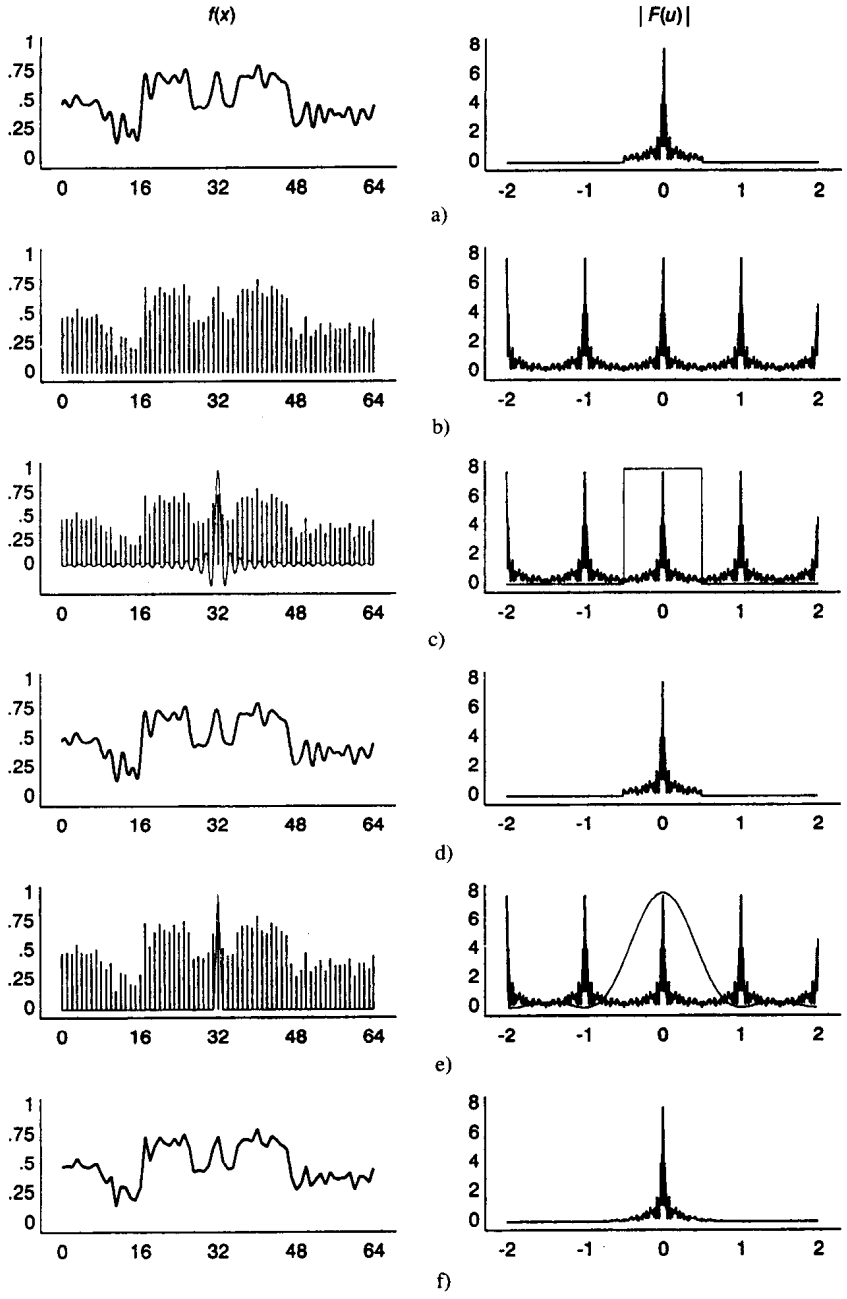


图14-27 采样和重构：足够的采样频率。a) 原始信号，b) 采样信号，c) 准备用sinc进行重构的采样信号，d) 用sinc重构的信号，e) 准备用三角波重构的采样信号，f) 用三角波重构的信号。（由哥伦比亚大学George Wolberg提供。）

若采样频率太低，频谱的副本会出现重选，如图14-28所示。在此情况下，重构过程将不能移去这些与原始信号的频谱重选的谱线。来自重复谱线的高频成份与来自原始频谱的低频成份混在一起，因而，在重构过程中，它将按低频处理。要注意的是，一个不充分的采样频率如

何在重构前后使较高频率看起来和较低频一样而引起走样。有两种方法解决这个问题：我们可选择足够高的频率去采样，但仅当信号不是无限频谱时，该方法才是足够的；另一种方法是，可在采样前对该信号进行滤波，以移去所有高于$f_s/2$的成份，如图14-29所示。

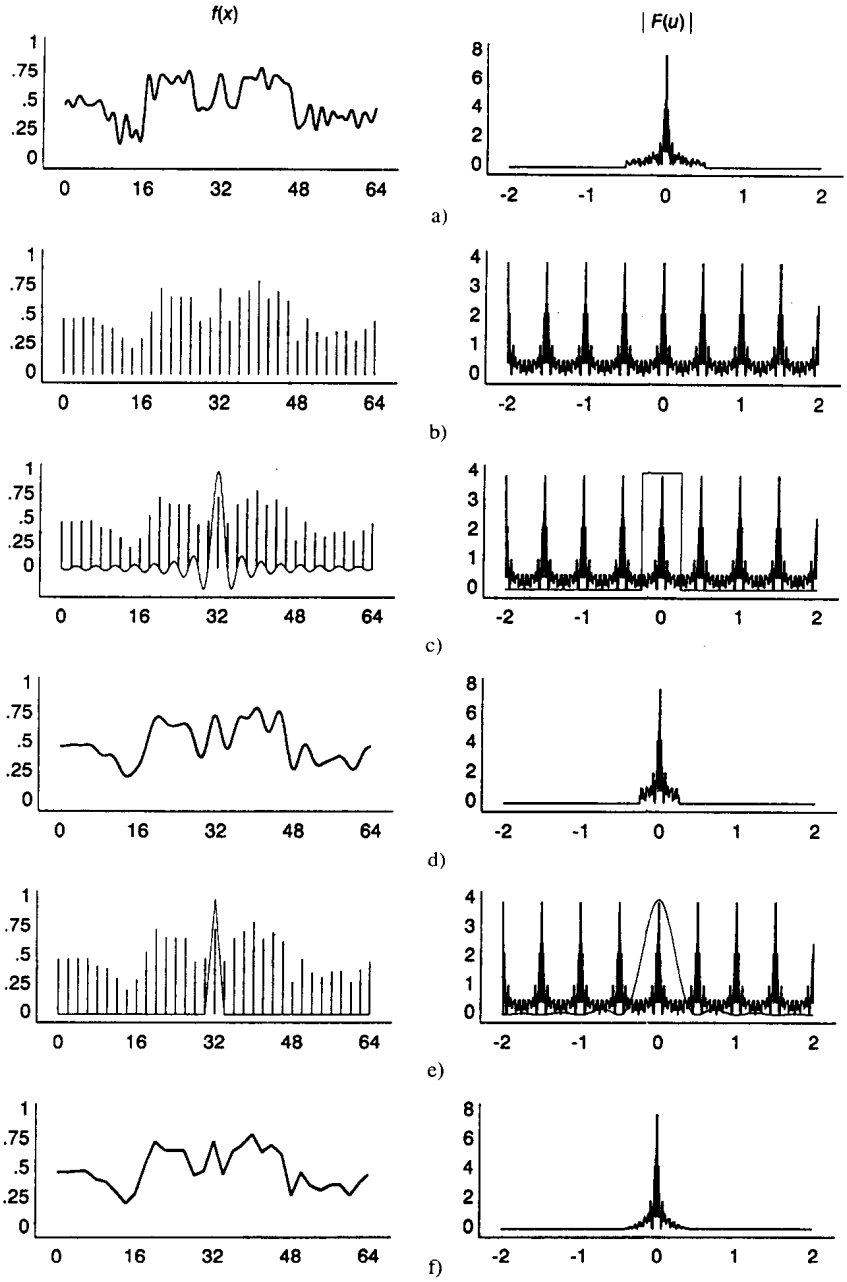


图14-28 采样和重构：采样频率不够。a) 原始信号，b) 采样信号，c) 准备用sinc进行重构的采样信号，d) 用sinc重构的信号，e) 准备用三角波重构的采样信号，f) 用三角波重构的信号。（由哥伦比亚大学George Wolberg提供。）

如果重构是由其他信号而不是一个sinc的卷积完成的，将会发生什么情况呢？借助于一种称为*采样和保持*的过程，在帧存中，样本被转换成连续的视频信号：对于被重构的信号，每个连续样本的值仅仅在一个像素的持续期间保存着。这一过程对应于一个样本与一个像素

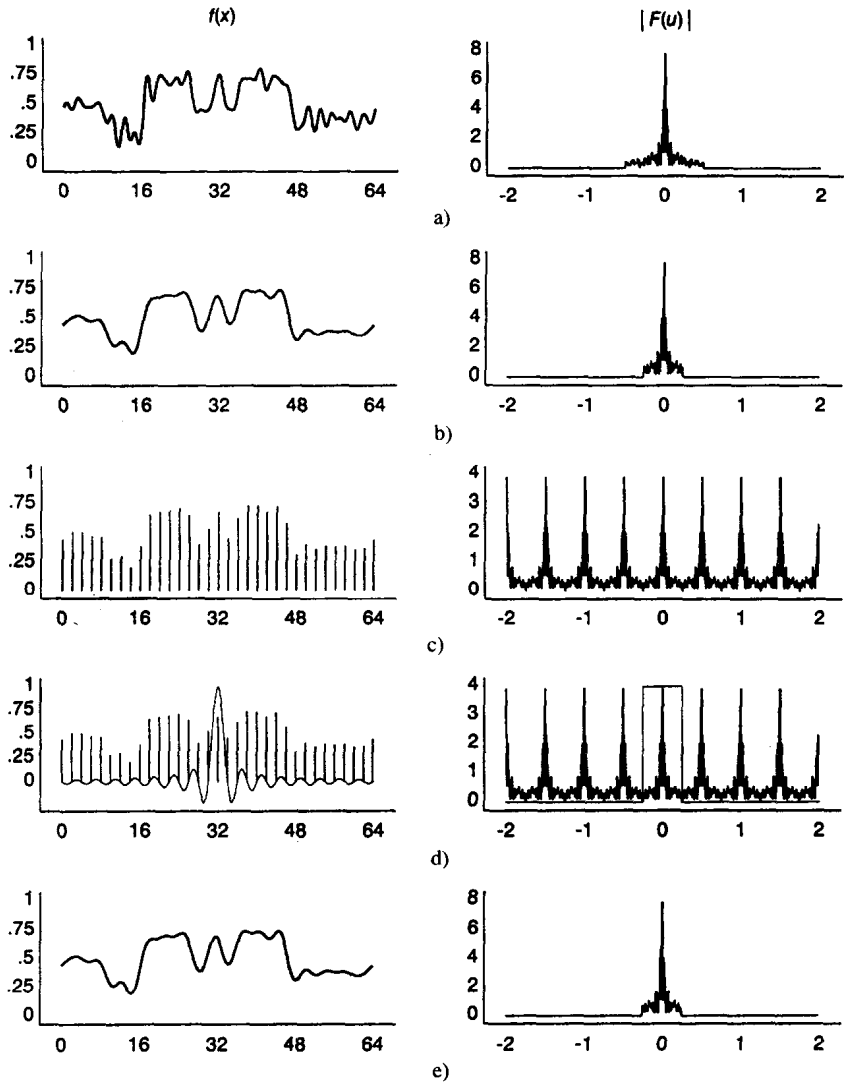


图14-29 滤波、采样和重构：滤波以后采样频率足够。a) 原始信号，b) 低通滤波后的信号，c) 采样信号，d) 准备用sinc重构的采样信号，e) 用sinc重构以后的信号。(由哥伦比亚大学George Wolberg提供。)

宽度的盒式滤波器求卷积，如图14-30所示，并给了我们一个概念，即一个像素就像一组方块瓦片之一铺着显示器。这最终的信号在像素之间有很陡的变化，对应于未被该样本表示的高频成份。这个效果是通常所谓的光栅化。虽然，视频硬件名义上采样和保持着每个像素的亮度，而产生用于CRT模拟电压的线路和CRT本身通常不能足够快地在像素的亮度间产生不连续的跳动。CRT光斑的高斯分布也减轻了这个问题。这样，借助于与盒式滤波器等价的卷积并伴随着高斯卷积，采样信号得以重构。然而，在光栅CRT显示中用像素重叠放缩以增加分配给一个像素的屏幕空间时，特别容易看到光栅。在打印机、数字录像机和LCD技术中，光栅也更为明显，其中像素到像素的转换是非常敏锐的，并产生出恒定亮度的具有相对明显边沿的方形像素。

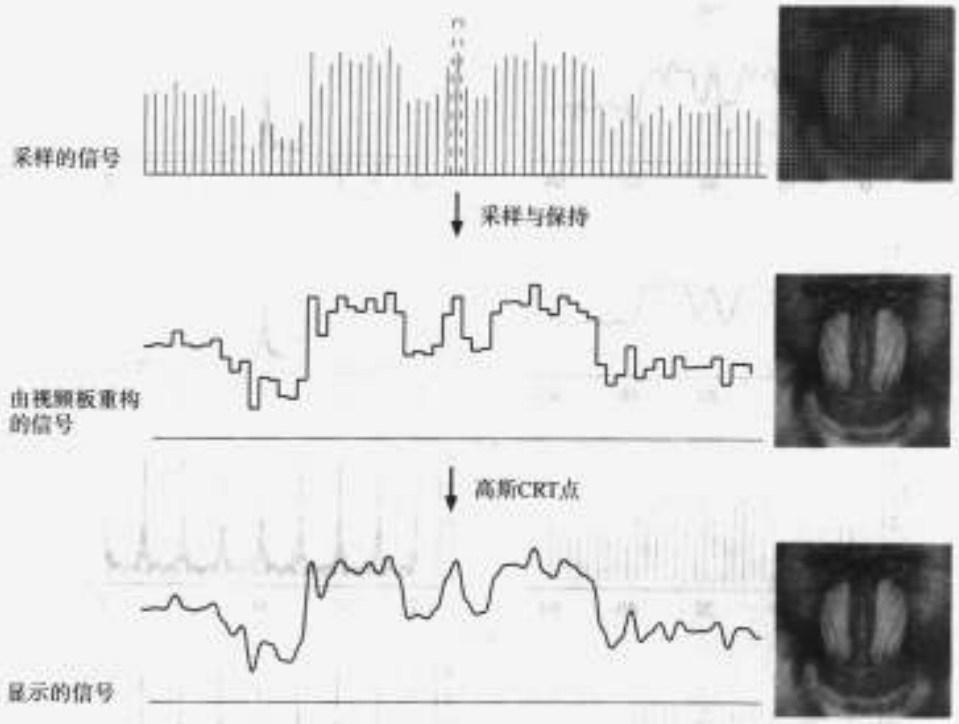


图14-30 由采样和保持实现的重构及高斯CRT图。（由哥伦比亚大学George Wolberg提供。）

我们早就注意到，必须以一个高于$2f_s$的频率来采样一个信号，才有可能做到更好的重构。如果用于样本重构的滤波器不是一个截去顶端的sinc，正如显示一个图像时总要发生的那样，那么，采样频率必须更高！例如，考虑采样频率稍微高于$2f_s$，最终样本会描绘出由一个低频正弦波调制（相乘）的原始信号，如图14-31所示。若用一个像素宽的盒式滤波器重构该信号，则混合光栅信号的低频幅度调制仍然保留。若卷积是由一个截去顶端的sinc来执行的，当然可以恢复原始信号。因此，采样前后不可避免地使用非理想的滤波器，将导致更高的采样频率。Mitchell和Netravali [MITC88]讨论了涉及到做一个好的重构工作的某些问题。

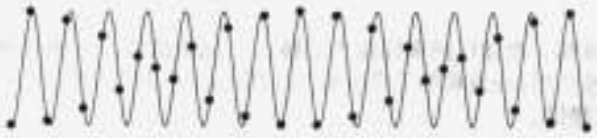


图14-31 一个信号以略高于奈奎斯特频率采样。（由哥伦比亚大学George Wolberg提供。）

14.10.6 实际的反走样

我们已经看到图像综合涉及采样和重构，也注意到我们很少用软件来做硬件采用的重构方法。实现反走样的绘制算法采用了点采样或一种分析方法，例如，区域采样。无论哪一种情况，对每个像素必须确定一个最终的值。在15.7.3节讨论的Catmull算法是一种使用不加权区域采样的分析（并且开销很大）方法的实例。它对应于在计算每个像素的样本值之前在物体的精确度上进行滤波。在采样之前的滤波通常称为前置滤波。当用超采样时，样本是按照我们前面讨论过的一种采样及加权滤波来组合的，权值是按连续卷积的离散方法确定的。该滤波器以一个阵列值来表示。如图14-32所示，滤波器阵列位于超采样值的阵列上，并且，相应位置的值所得乘积之和决定了一个取于滤波器中心点的信号样本。然后，该滤波阵列移到下一个采样的位置上，样本的数目对

应于被生成的滤波像素的像素分辨率。因为滤波是在点采样之后执行，这种方法通常称为后置滤波，它实际对应于只从所选的空间点上的样本重构该信号。然后，这些重构的值被作为新样本。这样，超级采样实现了一个加权区域采样的离散近似。

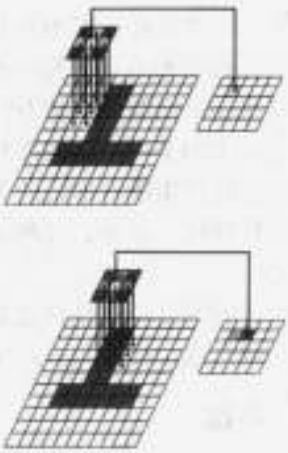


图14-32 数字滤波。滤波器用于组合样本并创建新样本

虽然使用一个像素宽度的盒式滤波器对所有子像素样本予以平均在计算上是很吸引人的，但较好的滤波器可产生更好的结果，如图14-33所示。请注意，不管用什么样的滤波器用于样本的后置滤波，由于不充分的初始采样频率所造成的损坏是不可修复的。一条经验规则是，在x和y方向进行四倍的超采样往往是令人满意的[WHIT85]。因为在大多数图形图像上，高频是由边界处的不连续性所造成的。这种不连续的边沿有一种衰减很快的傅里叶变换（类似于一个脉冲的傅里叶变换——sinc），与此相反，带有纹理的图像和透视图上较远的物体有一种在高频段更为丰富的傅里叶变换，它使得滤波更为困难。

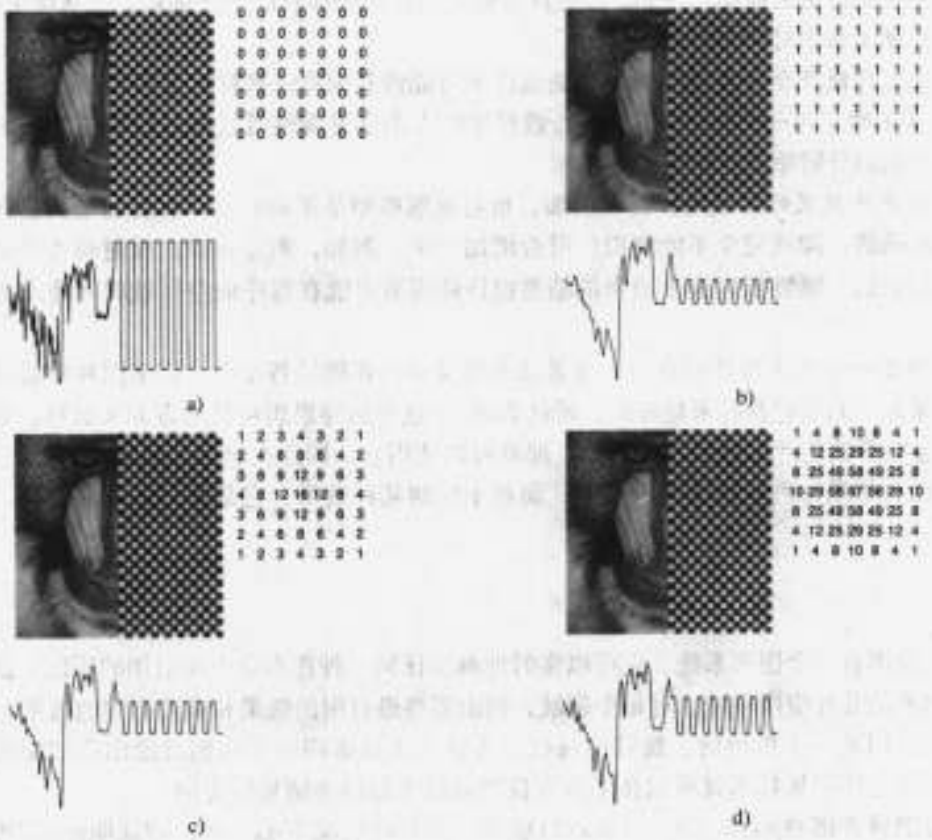


图14-33 滤波后的图像，中心扫描线图及滤波器核。a)原始图像，b)盒式滤波器，c)Bartlett滤波器，d)高斯滤波器。图像为512×512，滤波器为7×7。中心扫描线在棋盘格的底部。因为二维滤波器覆盖了扫描线上方和下方的白方格和黑方格，沿着棋盘格中心扫描线的滤波后信号大大地衰减了。（由哥伦比亚大学George Wolberg提供。）

虽然，增加采样频率很容易，但这种方法相应地增加了处理时间和内存，使其用途受限。点采样的多种变形可以处理这些问题而不牺牲点采样本身的简化机制。在*自适应级超采样*中，其采样频率在一幅图像中是不同的，当系统决定有需要时，可增加样本，其示例在15.10.4节中讨论。在16.12.4节中讨论的*随机超采样*中将样本放在随机确定的位置上，而不是放在矩形网格上。这种方法产生噪音形式的走样，这样，我们的视觉系统感到不舒服的程度要低于明确定义的规则走样的频率成份。这两种方法可结合起来，在已取得的统计特性的基础上决定新样本定位于何处。

636 ~ 645

当原始的源信号本身就是采样的图像时，伴随着重采样的后置滤波可用于产生一个经过缩放、旋转或是以一种不同方式变形的新的图像。这样的一些图像变化将在第17章讨论。

14.11 小结

本章中，我们提供了有关真实感图像技术的一个高层次的介绍。然后，我们研究了产生走样的原因和处理方法。在下面几章中，我们将详细讨论这些技术是如何实现的。当你读后面几章出现的算法时，你的脑子中应当记住五个关键问题：

1) *这个算法是一般的算法还是具有特殊目的的算法？*某些技术仅在特定环境中是好的，而另一些则设计得更为通用。例如，某些算法假定所有的物体是凸的多面体，并从这个假定出发，导出它的速度和相对简单的部分。

2) *反走样能否合并在内？*某些算法也许不可能像其他算法那样容易进行反走样。

3) *算法的空间-时间特性是什么？*数据库的大小和复杂性等这样的因素如何影响算法？或者图片绘制的分辨率又如何影响算法？

4) *所产生效果的可信度如何？*例如，折射光照模型是正确的吗？它看起来只是在某些特定情况是正确的，抑或完全不能模拟？可否增加效果？例如，阴影和镜面反射是否可加上去。如何能确信它们？牺牲据以表现效果的精度也许使得有可能在程序的空间或时间要求方面有重要改善。

5) *给定一个生成图片的目的，该算法是否适当？*在随后各章中，许多图片背后的基本原理可以归结为“如果它看起来是好的，就这么做。”这个指导思想可从两方面来解释。如果一种简单或快速的算法能产生吸引人的效果，那就可以使用它，即使在物理法则上找不到正当的证明。另一方面，如果一种算法昂贵得可怕，而对于绘制某种效果，它是惟一已知的方法，你就可以使用它。

习题

14.1 假定你有一个图形系统，它可以实时地画出任何一种在本章中所引用的彩图，试考虑你所熟悉的几种应用领域。对每个领域，列出那些最有用的效果和最没有用的效果。

14.2 请说明从一个单色的、旋转的线框立方体的正投影图中不可能推论出它的旋转方向。请解释怎样附加技术就可以在不改变投影前提下帮助弄清旋转方向。

14.3 考虑脉冲函数$f(x)=1$,对$-1\leqslant x\leqslant 1$成立，而其他情况下$f(x)=0$；请证明$f(x)$的傅里叶变换是一个sinc函数的倍数。提示：$f(x)$的傅里叶变换可按下式计算：

646

$$F(u)=\int_{-1}^{+1}f(x)[\cos 2\pi ux-i\sin 2\pi ux]dx$$

因为在$f(x)=0$的区间对积分没什么贡献，而在余下区间里，$f(x)=1$（对你的答案做傅里

叶反变换，你将得到原始函数。）

14.4 请证明按一个宽度为2的三角波滤波器重构一个信号，相当于它的样本做线性插值，若滤波器更宽，又将发生什么情况？

14.5 写一个程序，它允许你对一个图像用一个滤波器核心进行卷积。用不同的滤波核在x和y方向以2, 4和8倍的像素从原始图像去建立同样大小的新图像。你可以得到由你曾经使用过的图形软件包所产生的保存在帧缓存中的原始图像。你的滤波后的图像看起来比同样分辨率的原始图像更好吗？你的经验能证实14.10.6节所提及的经验规则吗？ 647

第15章 可见面的判定

对于一组给定的三维物体与观察视见说明，我们常常需要确定物体上的哪些线或面是可见的，从而在绘制时能够只显示这些可见的部分。这种可见性在透视投影中是相对于投影中心的，在平行投影中则是相对于平行投影方向的。这一确定可见性的过程称为*可见线判定*或*可见面判定*，也可称为*隐藏线消除*或*隐藏面消除*。在可见线的判定过程中，线可以看成是不透明面片的边，这些面片对远离视点的面片的边产生遮挡。因此，我们可以将整个过程通称为*可见面判定*。

尽管这一基本思想的表述非常简单，实现起来却要求计算机具有很强的处理能力，并且在常规的机器上需要花费大量的运算时间。为解决这个问题，人们提出了许多构思精巧的可见面判定算法，这将在本章中进行介绍。另外，人们为此也设计了一些专用的体系结构，其中一些将在第18章中讨论。可见面的判定有两种最基本的解决方案，在两种方案中都将每一对象看成是由一个或多个多边形（或更复杂的面片）组成的。

第一种方案是在生成图像中的每一点时确定n个对象中的哪个点可见。该方案可用伪代码表示如下：

```
for （图像中的每个像素） {
    确定由投影点与像素连线穿过的距；
        观察点最近的对象；
    用适当的颜色绘制该像素；
}
```

最直接的一种方法是对每一像素考察全部n个对象，判定沿投影点穿过该像素的对象中哪一个离视点最近。对p个像素而言，其计算量正比于np，其中p在高分辨率显示中会超过10^6。 649

第二种方案是直接在对象之间互相比较，以除去完全不可见的对象或不可见的部分。下面的伪代码便体现了这种思想：

```
for(世界坐标中的每一个对象){
    判定对象中未被遮挡的部分，包括其自身的遮挡
        与其他对象的遮挡
    用适当的颜色绘出可见部分
}
```

我们可以简单地通过将对象与其自身以及与其他对象的比较来除去不可见的部分。这样做的运算量正比于n^2。表面上看，由于$n<p$，似乎第二种方案更为高效，但实际上，它的每一个基本的步骤都很复杂且费时，因此，第二种方案往往更慢且更难实现。

人们常常称这两种方案为*图像精度算法*与*对象精度算法*。图像精度算法是基于显示设备的分辨率来判定每一个像素可见性的算法，而对象精度算法则是基于对象定义的精度来判定每一对象可见性的算法。[⊖] 由于对象精度的遮挡计算与显示分辨率无关，所以最终少不了用特定的

⊖ 术语图像空间和对象空间（由Sutherland、Sproull和Schumaker[SUTH74a]推广）常用来表示相同的区分。但是，这些术语在计算机图形学中还常常代表不同的意义。例如，图像空间用来表示透视变换后[CATM75]或投影到视图平面后[GILO78]在其原有表示精度下的对象。为了避免混淆，我们稍加修改了我们所用的术语。我们对于对象的透视变换或投影将以明显的方式表达出来，并用术语图像精度和对象精度来指示完成计算所采用的精度。例如，如果原有对象定义的精度不变，两个对象不在一个投影平面上相交的操作则是对象精度的操作。

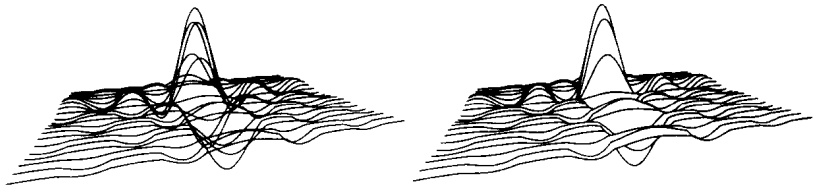
分辨率来显示这一步。如果生成图像的大小发生改变，只有最后显示这一步需要重新计算，比如在光栅显示器上显示相应的不同数目的像素。这是由于每个可见对象投影的几何图形全部是用对象数据库的精度表示的。而图像精度算法则与此形成对比。例如，用图像精度算法将一幅图像进行放大时，原来用低分辨率生成的可见面图像必须重新计算以表现更多的细节。在第14章中我们讨论过采样算法，这里我们可以认为对象精度算法是在连续的对象数据上操作，而图像精度算法则作用于离散的采样数据。这也就是图像精度算法在计算可见性时存在走样问题而对象精度算法不存在这个问题的原因。

650 对象精度算法最初是为向量图形系统开发出来的。在这些设备上，隐藏线的消除是非常自然的事情：完全被其他面遮挡的线被去除；部分被遮挡的线被裁剪成一条或多条可见的线段。所有的处理都在原始的线条集合中完成，并生成相同格式的新的集合。相比较而言，图像精度算法则是为光栅显示设备编写的，其目的是减少需要进行可见性判定的点的数量。这是一种很自然的选择。向量显示具有较大的地址空间（在早期的系统中就有4096 × 4096），但能够显示的线条与对象的数目有很大的限制；而光栅显示设备就不同，它的地址空间非常有限（早期系统为256 × 256），却具有显示无限多个对象的潜力。后续的算法常常结合了对象精度算法和图像精度算法这两种算法的思想，用对象精度算法来获得精确度，而用图像精度算法来获取绘制速度。

本章的内容是这样安排的：首先我们介绍一个可见面问题的子集，即显示一个双变量的单值函数；然后讨论提高可见面算法效率的一些方法；最后，我们给出判定可见面的主要算法。

15.1 双变量函数

在计算机图形学中，常常会用到双变量的单值连续函数，可记作$y=f(x, z)$。这些函数定义出如图15-1a中所示的曲面。这一类问题的隐藏面消除比较特殊，能够找到一种较快的方法加以解决。



a) b)

图15-1 双变量的单值函数：a)未消除隐藏线，b)消除隐藏线。（由哥伦比亚大学的Michael Hatzitheodorou提供。）

x与z的函数可以近似看作矩阵Y$m \times n$矩阵的值。每一矩阵元素可代表规则网格采样点上的高度。假定矩阵的每一列对应于x坐标，每一行对应于z坐标。换句话说，长方形矩阵与x与z轴对齐。矩阵元素的下标与元素的值共同构成一个三维点的坐标。曲面的线框图可以通过小的平

651 面片来线性逼近，其中，组成平面片的曲线段分别由矩阵Y的行与列的点连接而得。隐藏线算法要求消除线上所有相对于视点不可见的部分，如图15-1b所示。

要解决这个问题，我们首先考虑只绘制z为常量的曲线段的情况，假定离视点最近的曲线段为曲面的边界。那么，根据这样一个事实：z值固定的曲线段都位于由z定义的一个平面中，且这些平面互相平行，可以产生一个有效的算法[WILL72；WRIG73；BUTL79]。既然这些平面两两不能相交，从而这些曲线段之间也两两不能相交，那么，每一个曲线段就不可能被距视点画远的z所定义的平面上的曲线段遮挡。因此，我们按照距离视点由近到远的顺序来绘制这些曲线段，就可以建立一个从前到后的排序。依据这样的序列，要正确地绘出每一曲线段，只

需注意每个z平面上的被已绘出的折线所遮挡的部分不必绘出即可。

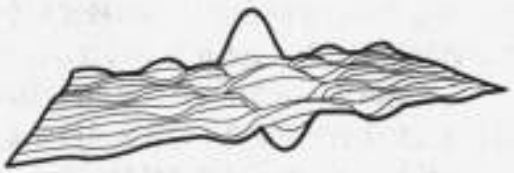
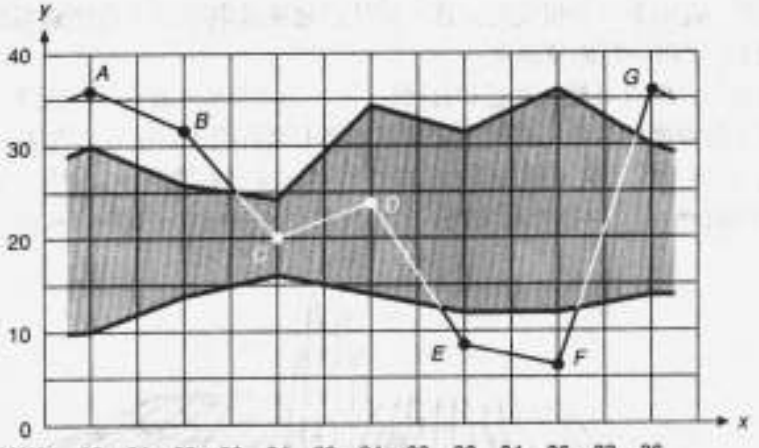


图15-2 绘出的线的轮廓。（由哥伦比亚大学的Michael Hatzitheodorou提供。）

考虑绘出位于视图平面上的折线段的轮廓边界，如图15-2中粗线所示。每添加一条新的折线时，只有当它的投影超出了该轮廓的上界或下界时，才是可见的。由于新绘出的折线段的z值总是大于已绘出的折线，它不可能穿过已绘出的曲面。这样，要确定哪一部分可见，只需要将当前绘出的折线在视图平面上投影的y值与曲面轮廓投影的y值相比较即可。对每一x保存其最大与最小的轮廓y值的算法称为水平线算法（horizon line algorithm）。

Wright[WRIG73]采用的表示轮廓的方法使用两个一维数组*YMIN*与*YMAX*来分别记录最小与最大的投影y值（相对于投影x的有限集）。因为仅含有有限项，这是一个图像精度的数据结构。*YMIN*与*YMAX*分别被初始化为平面投影可取的最大与最小的y值。绘制一条新的折线时，每一对相邻顶点的投影的y值与相应位置的轮廓数组的y值相比较。如图15-3所示，顶点值在相应的*YMAX*值以上（*A*，*B*，*G*）与*YMIN*以下（*E*，*F*）的点是可见的；反之是不可见的（*C*，*D*）。652 如果两个顶点都不可见，则该线段不可见，且轮廓数组保持不变。若两个顶点均可见（*AB*，*EF*），则该线段完全可见且应该被完整绘出，轮廓数组也要进行相应修改。常量z的折线上两个相邻顶点的x坐标常常映射到轮廓数组中不相邻的位置，这时，中间受影响的轮廓数组的y值由这两个相邻顶点的投影y值做线性插值获得。



新的 *YMAX*	**36**	34	**32**	26	**24**	29	34	33	32	34	36	33	36
新的 *YMIN*	10	12	14	15	16	15	14	13	**8**	7	6	13	14
线的 *Y*	**36**	34	**32**	26	**20**	22	**24**	16	**8**	7	6	21	**36**
旧的 *YMAX*	30	28	26	25	24	29	34	33	32	34	36	33	30
旧的 *YMIN*	10	12	14	15	16	15	14	13	12	12	12	13	14

图15-3 两端点之间的线段交点的y值由插值获得并与轮廓数组中的值相比较。端点值用黑体表示

最后需要考虑的是线段部分可见的情况。这时线段的两个顶点相对于同一轮廓数组并不一定都可见。这种情况在一般意义上指的是一个顶点可见，而另一顶点不可见（*BC*，*DE*），也包括两个顶点都可见的特殊情况，即一个在*YMAX*以上而另一个在*YMIN*以下（*FG*）。此时，在线段与轮廓数组的交点处将相邻顶点y的插值与数组中所保存的y值相比较，当插值点完全落在轮廓内部时，这一部分线段是不可见的。只有线段落在轮廓外的部分需要被绘出，同时轮廓数组

也进行相应的修改（图15-3）。当线段在两个相邻的数组元素之间改变可见性时，该线段一定与轮廓边界相交，这一交点可作为画线算法中的一个端点。

令人遗憾的是，图像精度算法采用的*YMIN*与*YMAX*轮廓数组很容易产生走样问题。如果采用的基本画线算法的分辨率高于轮廓数组的分辨率，那么，常常会出现走样的情况，即一些不可见的线段被绘制，而一些可见的线段却变得不可见。如图15-4所示，三条折线按从前到后的顺序依
653 次绘出。图15-4a和图15-4b中的两条折线相交时中间形成一段空隙。通过计算插值可以求出两条折线的交点，从而正确地判断出第二条折线相对于第一条折线的可见性。但是，绘出第二条折线后，*YMAX*变成了相同值。此时，再在图15-4c中绘出第三条折线，该折线在经过相交处的间隙时也变成了不可见部分。用更高分辨率的轮廓数组可以减轻这一问题，但运算量相应增大。

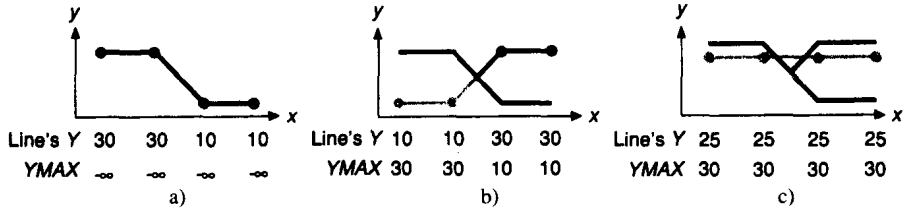


图15-4 使用图像精度的轮廓数组产生的走样问题

另一种方法是采用对象精度算法[WILL72]，其思想是用两个对象精度的折线代替*YMIN*与*YMAX*数组来表示轮廓边界。当考虑常量z的折线的每一部分时，只绘出投影到YMAX折线以上或YMIN折线以下的部分。可见的投影线从它们与轮廓线相交处起，连接成新的轮廓线，取代原有的轮廓线，直至下一个相交处。这一算法的精度选取必须考虑搜索与维护轮廓线所需要的附加运算量以及代码编写的复杂度。

与常量z相似，假设要绘制常量x的折线，如图15-5所示。在这种情况下，离观察者最近的常量x的折线并不是曲面的边界。在图中它是左数第七根折线（最接近垂直的一根）。要正确地绘制出这个曲面，必须对位于这一最近线右边的折线按从左到右的顺序，对位于其左边的曲线段按从右到左的顺序进行绘制。在这两种情况下的折线都是按照相对于观察者从前到后的顺序绘制的。

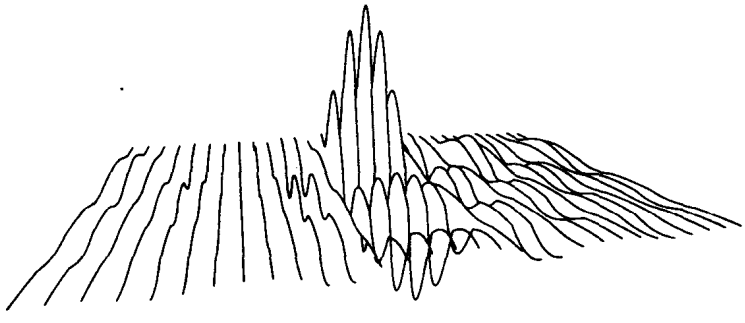


图15-5 使用常量x的折线而不是常量z的折线绘出的曲面。（由哥伦比亚大学Michael Hatzitheodorou提供。）

如图15-6所示，图像精度算法能够很方便地扩充到对常量x的曲线的绘制。将常量x与常量z的折线叠加在一起便构成了整个曲面，但也存在一个问题，即两类折线之间不能相互遮挡（图15-7）。Wright[WRIG73]提出通过对两类曲线交叉进行处理能够解决这个问题。那些最大程度上平行于视图平面的折线（如常量z的折线）按与前面相同的顺序进行处理。每处理一条常量z的折线后，立即绘出常量x的折线中位于刚处理过的z折线与下一条z折线之间的部分。x

的线段的绘出必须使用与绘出z线段相同的轮廓数据结构，同时，它们也必须依照从前到后的顺序处理。图15-8a给出了用正确的顺序（本例中是从左至右）绘出常量x的折线所得的图形，图15-8b则是用相反的错误的顺序（从右到左）处理的常量x的线段，也导致了错误的结果，这是由于在*YMAX*数组中后绘出的线段被前面绘出的线段所遮挡而导致的。

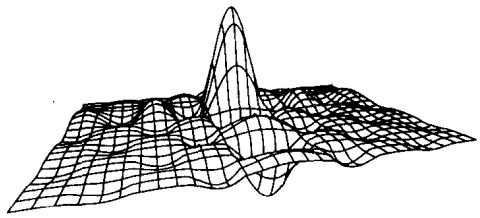


图15-6 由常量x与常量z的曲线合成的图15-5的曲面。（由哥伦比亚大学Michael Hatzitheodorou提供。）

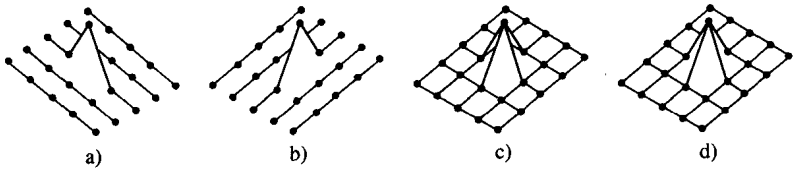


图15-7 a)常量z的线段，b)常量x的线段，c) a)与b)的简单叠加，d)正确的结果（基于[WRIG73]。）

654
~
655

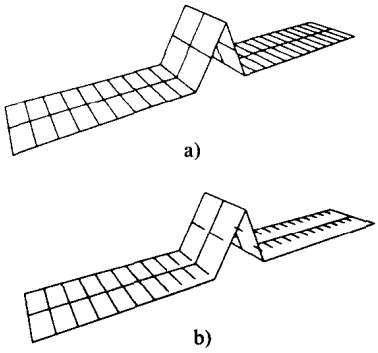


图15-8 常量x的折线与常量z的折线一样，必须按正确的顺序处理。a)正确处理的线，b)不正确的线。（由哥伦比亚大学Michael Hatzitheodorou提供。）

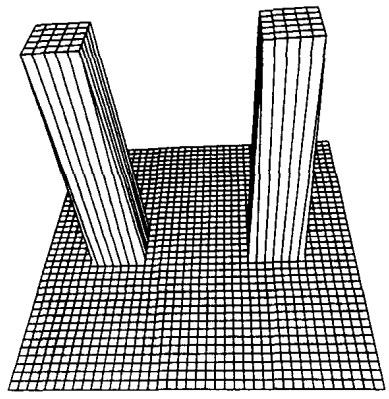


图15-9 用其他算法生成的投影图。（经加利福尼亚大学伯克利分校David P. Anderson许可。）

尽管本节所介绍的算法很有效，但这些算法都与观察（视见）参数说明密切相关。当对象的轮廓在视图平面上的投影不是x的函数时，这些算法不起作用。图15-9给出了用其他观察（视见）说明生成的一个例子。，该图是用由Anderson[ANDE82]提出的算法绘制的，它使用更复杂的数据结构来处理任意的观察说明。

15.2 可见面判定算法中的常用技术

由前面的讨论可以看出，通过选取适当的数据结构，可以有效地解决双变量函数的可见线的判定问题。那么，针对一般性的可见面判定问题，有什么好的解决方法呢？在本章的开始曾给出过图像精度算法与对象精度算法的简单表述，其中包含许多运算量较大的操作。比如，确定对象和投影线，求两个对象的投影，判断两个投影是否相交以及交于何处。对每一次相交运算，都必须计算离观察者最近的可见对象。为了尽可能减少创建图片所需的时间，我们希望使 656
这些费时的操作以尽可能少的时间与尽可能少的次数执行。以下各节便给出了几种相关的技术。

15.2.1 相关性

Sutherland、Sproull与Schumacker[SUTH74a]提出可见面判定算法能够利用相关性——环境或它的投影所表现出来的局部相似程度。在我们所需要绘制的环境中常常存在这样的对象，它们的属性从一部分到另一部分发生着平滑的变化；或者说，在一幅图像中我们怎样区分不同的对象，依赖于对象的属性值（包括深度、颜色及纹理等），这些属性很少有不连续点。这种相关性使我们在完成环境或图像的某一部分的计算时，对其相邻部分的计算可以重用这一部分的计算结果，包括完全照搬的重用和在其上进行适量增减的变化。与重新计算每一部分相比，这样做大大提高了运算效率。这种相关性可以概括为以下几类[SUTH74a]：

- *对象相关性*。如果一个对象与另一个对象完全分离，那么就只需要对这两个对象之间进行比较，而不必对组成对象的面与边进行比较。例如，对象A的每一部分都比对象B远离视点，那么就不需要判断A的面是否遮挡B的面。
- *面相关性*。指一个面上的曲面属性发生连续的变化。这样，对某一部分的计算只要经过适量的修改便同样适用于其相邻部分。在有些模型中，面与面之间能够保证不互相穿插。
- *边相关性*。指一条边只有在从一条可见边后面穿过或穿过一可见面时才会改变其可见性。
- *隐含边相关性*。两平面相交时，它们的交线（隐含边）可由相交的两点确定。
- *扫描线相关性*。指图像中一条扫描线上的可见对象跨度与其相邻的扫描线上的可见对象跨度相差甚小。
- *区域相关性*。指一组相邻的像素被同一可见面所覆盖。一种特例是*跨度相关性*，即面在扫描线上一组相邻像素的跨度上的可见性。
- *深度相关性*。指同一面的相邻部分深度值相近，而不同面在屏幕同一位置的深度值相距甚远。在某一面上，一旦确定了一点的深度值，其余点的深度值常常可以通过一个简单的差分方程求出。
- *帧相关性*。同一环境在两个相邻的时刻生成的图像有可能非常相似，尽管其对象与视点发生了小的改变。因此，一幅图像的计算可以重用于下一幅图像中。

15.2.2 透视变换

657 可见面判定显然应该在投影到二维图像之前的三维空间完成，因为二维变换破坏了深度比较所需要的深度信息。撇开选择的投影方式不谈，在一个点处的基本的深度比较可以简化为这样的问题：给定点$P_1=(x_1,y_1,z_1)$与$P_2=(x_2,y_2,z_2)$，它们是否互相遮挡？也可以这样问，P_1与P_2是否在同一投影线上（见图15-10）？如果回答是肯定的，那么需要对z_1与z_2进行比较以判定哪一点离视点更近；反之，则两点之间不存在遮挡关系。

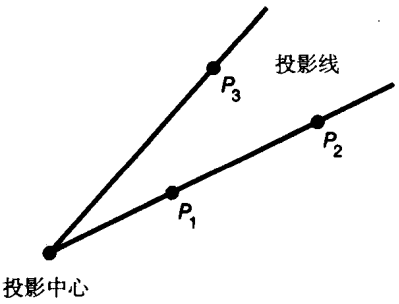


图15-10 P_1与P_2在同一条投影线上，则较近的一点遮挡较远的一点；否则，不形成遮挡关系（如P_1不遮挡P_3）

深度比较通常在规格化变换（见第6章）之后进行，这样，在平行投影中，投影线方向平行于z轴，而在透视投影中则从原点向外发散。对平行投影而言，如果两点的$x_1=x_2$且$y_1=y_2$，则两点在同一投影线上。而对透视投影而言，我们必须做四次除法，即判断$x_1/z_1=x_2/z_2$且$y_1/z_1=y_2/z_2$，才能确定两点在同一投影线上（见图15-10）。更进一步，如果P_1要与另外的一点P_3进行比较，还需要再多做两次除法。

为了免去这些不必要的除法运算，可以先将三维的对象变换到三维的屏幕坐标系。这样，变换后对象的平行投影就与未变换对象的透视投影一致了。从而使点与点之间的遮挡测试也与平行投影中相同。这种透视变换将对象进行变形，同时将投影中心移动到z轴的正无穷远处，使投影线互相平行（见图6-56）。图15-11表示的是该变换的透视投影视见体的效果。图15-12是一个立方体的变形效果图。

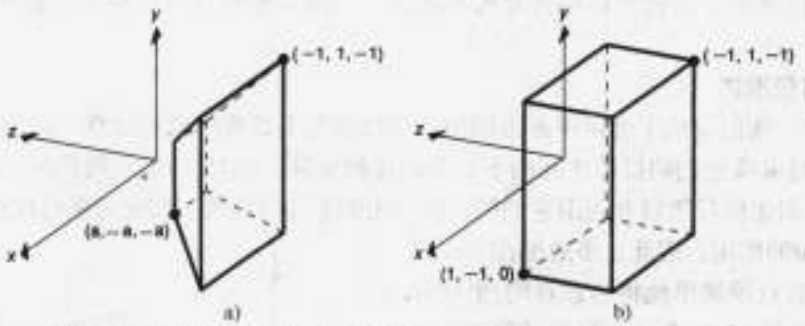


图15-11 规格化的透视投影视见体。a)透视变换前，b)透视变换后

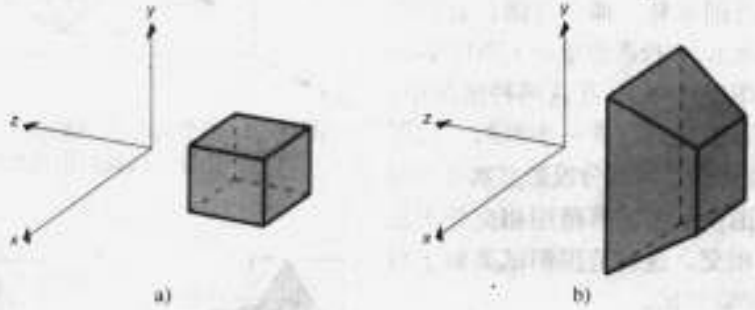


图15-12 立方体。a)透视变换前，b)透视变换后

这一变换的实质是保存了相对的深度、直线和平面，同时也执行了透视缩小效应。在第6章中已经讨论过，完成透视缩小效应的除法运算仅对每一点执行一次，而不必在两点之间进行比较时执行。式(6-48)的矩阵可表示如下：

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1+z_{\min}} & \dfrac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad z_{\min} \neq -1 \tag{15-1}$$

该矩阵将规格化的透视投影视见体变换为长方体，其边界为

$$-1 \leqslant x \leqslant 1, \quad -1 \leqslant y \leqslant 1, \quad -1 \leqslant z \leqslant 0 \tag{15-2}$$

规格化后的棱台视见体在进行M矩阵变换之前可以进行一定的裁剪，但裁剪之后的结果必须被矩阵M乘。另一个有效的方法是将M矩阵合成到透视规格化变换矩阵N_{pr}中（见第6章），这样就只需要执行一次矩阵乘法。然后在做除法之前对齐次坐标进行裁剪。假设相乘后的结果是(X, Y, Z, W)，那么，当$W>0$时，裁剪的边界为

$$-W \leqslant X \leqslant W, \quad -W \leqslant Y \leqslant W, \quad -W \leqslant Z \leqslant 0 \tag{15-3}$$

658
↕
659

上式由（15-2）中用X/W，Y/W，Z/W分别替换x，y，z而得，表明式（15-2）中的x，y与z是除以W后所得的值。裁剪之后再除以W便可得（x_p，y_p，z_p）。（对于W<0的情况请参见6.5.4节。）要注意的是，矩阵M假定视见体位于z轴为负的半空间中。为了表述上的方便，我们的例子常常用减少正的z值来表示远离视点。在另外一些采用将右手坐标系转换到左手坐标系的系统中，则正好相反。

现在，我们可以从图15-10所显示的复杂性中解脱出来，继续进行可见面的判定。当然，一旦确定了平行投影，透视变换矩阵M就不需要了，因为平行投影规格化变换N_{par}能够使投影线平行于z轴。

15.2.3 范围与包围体

在第3章中，我们介绍了确定屏幕范围的方法以避免不必要的裁剪运算。这里，我们用同样的方法来消除对象或它们的投影之间的不必要的比较运算。图15-13所示的是两个对象（此处是三维多边形）、对象的投影以及包围它们的投影的正矩形屏幕范围。假定对象已经过15.2.2节中的透视变换矩阵M的作用，因此，多边形在（x，y）平面上的正投影只须简单地将各顶点的z坐标分量忽略即得。在图15-13中，它们的屏幕范围并未重叠，因此它们的投影不需要进行相交测试。如果它们的屏幕范围重叠，那么可能会有两种情况（如图15-14所示）：投影也重叠（图15-14a）或投影不重叠（图15-14b）。在这两种情况中，都必须对投影是否重叠进行进一步判断。对图15-14b，通过判断得知，两部分投影其实并不相交，这表明，包围投影的屏幕范围相交并不总代表着投影本身相交。这种范围测试类似于裁剪算法中的简易消除测试。

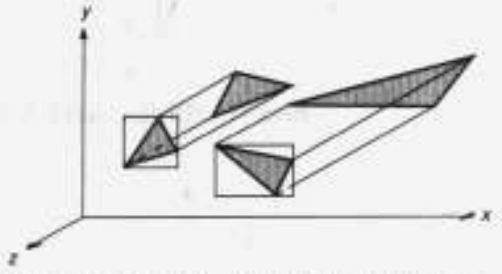


图15-13 两个对象、对象在（x，y）平面上的投影以及包围投影的区域

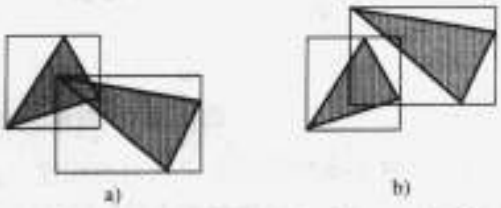


图15-14 包围对象投影的区域。a)包围域与投影均重叠，b)包围域重叠而投影不重叠

矩形区域测试也称为包围盒测试。这种区域也可用于像第7章中那样包围对象自身而不是它们的投影。这时的包围范围成为立体的，也称为包围体。另外，也有用于包围一维对象的，比如当我们需要判定两个对象在z轴上是否重叠。图15-15所示的便是这种情况：每一对象的最小与最大的z值之间的范围构成一个无限的包围域。满足下列条件时，两个对象在z轴上便无重叠：

$$z_{max2} < z_{min1} \quad 或 \quad z_{max1} < z_{min2} \tag{15-4}$$

在一维或多维度上利用最小与最大的边界进行比较也称为*极大极小界测试*（minmax testing）。极大极小界比较时，最复杂的部分在于确定范围本身。对于多边形（或其他完全被包含在由若干点定义的凸面体当中的对象）而言，重复计算所有点的坐标并记录坐标的最大与最小的值，便可计算出范围。

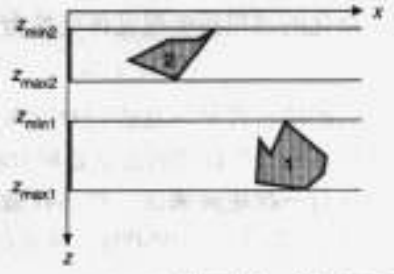


图15-15 用一维范围判定对象是否重叠

范围和包围体不仅可用于比较两个对象或

它们的投影，也可用于判定投影线是否与对象相交。包括计算点和二维投影的交或向量和三维对象的交（见15.10节）。

到目前为止我们只讨论了极大极小范围，当然还存在其他的包围体形式。那么，什么样的包围体最好呢？毫无疑问，答案取决于用包围体本身进行测试所需的代价以及包围体防止其包含对象测试不产生相交的能力。Weghorst、Hooper与Greenberg[WEGH84]把包围体的选择作为一个对象相交测试的总代价函数T的极小化来处理。这可以表示为

$$T = bB + oO \tag{15-5}$$

660 ↓ 661

其中b为包围体做相交测试的次数，B是执行包围体相交测试的代价，o是对象相交测试的次数（包围体实际相交的次数），O是对象做相交测试的代价。

由于只有当包围体实际相交时才会执行对象的相交测试，可知$o \leqslant b$。对特定对象与测试集合而言，O与b是常量，但B与o则随包围体的形状与尺寸大小发生变化。一个“较紧”的包围体，o会很小，但相应的B可能较大。包围体的有效性也可能受其包含对象的方位以及与其相交的对象的种类的影响。我们可以比较一下图15-16中所示的马车轮的两个包围体。如果投影方面垂直于(x, y)平面，则最紧致的包围体是球体。如果考虑投影方向垂直于(x, z)或(y, z)平面，则长方体为较紧致的包围体。因此，可能有多个包围体同一个对象相关，而根据不同情况选择恰当的一个使用。

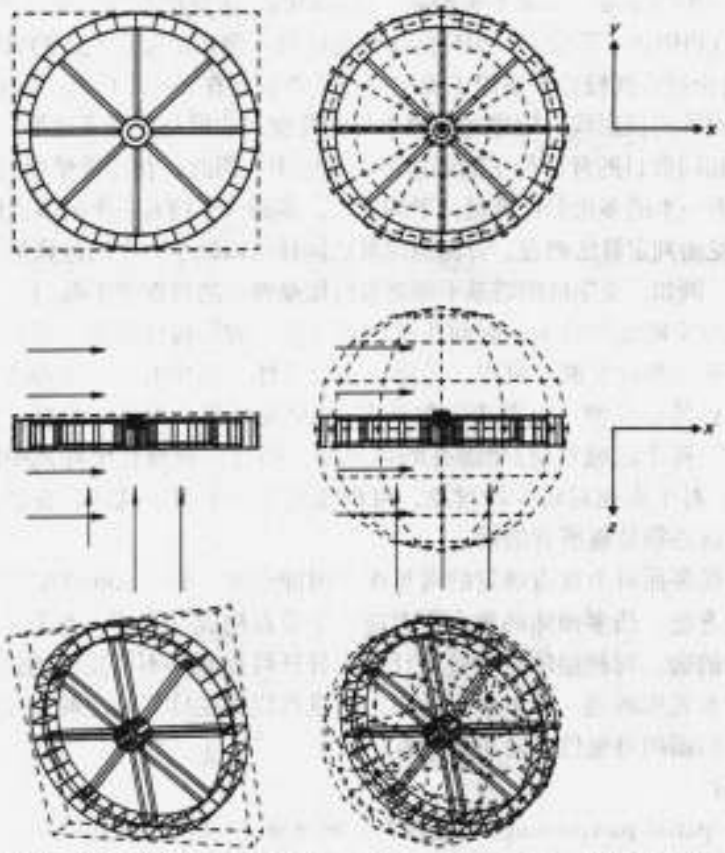


图15-16 选择包围体。（经允许引自Hank Weghorst, Gary Hooper, Donald P. Greenberg, Program of Computer Graphics, Cornell University, 1984。）

15.2.4 背面消除

如果一个对象能够近似为一实多面体，那么它的多边形表面可以完全包围住它的体积。假

定表面多边形的法向量指向多面体外部。如果多面体的内部完全位于前裁剪平面之前，那么那些表面法向量指向远离观察者一边的多边形将完全不可见（图15-17）。这些不可见的背面多边形应该在进一步的处理中被去掉，这种技术称为背面消除。类似地，非背面也可称为正面。

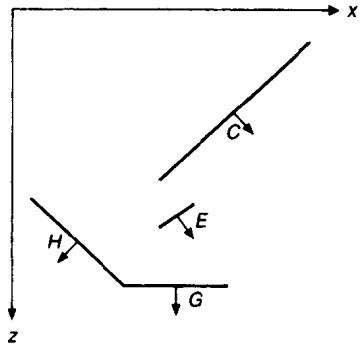


图15-17 背面消除。用灰色表示的背面多边形（A, B, D, F）被删除，而正面多边形（C, E, G, H）则保留下来。

在眼睛坐标系中，判别背面的方法很简单。只需求出投影中心到多边形上任一点组成的向量与该多边形的表面法向量的点积。若为非负，则是背面。（严格地说，点积为正表明是背面。点积为零时表明多边形被视见为边）。假设已通过透视变换或需要在（x, y）平面上做正交投影，投影方向为（0, 0, -1）。这时，点积测试可简化为判断曲面法向量的z值，若为负，则为背面。如果环境中仅含有一个凸多面体，那么可见面判定工作也就是进行背面的剔除，否则可能存在一些正面，如图15-17中的C与E，它们是部分可见或完全不可见的。

如果在多面体中存在丢失的或裁剪过的正面，或多边形根本就不是多面体的一部分时，背面多边形需要做进一步的处理。如果不想删除，最简单的方法是把背面多边形当成正面，即翻转其法向量的方向。在PHIGS + 系统中，用户能够为面的每一侧指定完全独立的属性集合。

662
~
663

7.12.2节中讨论过奇偶校验算法用于确定一点是否包含在多边形内部，从这个算法可推导出，一条穿过一个多面体的投影线一定穿过相同数目的背面多边形与正面多边形。于是，多面体投影中的点一定位于相同数目的背面与正面多边形的投影中。因此，在图像精度的可见面算法中，用背面消除能够省去一半的多边形计算量。平均来看，多面体中约有一半的多边形是背面的，因此，对对象精度的可见面判定算法而言，背面消除算法同样可以减少一半的运算量。（需要注意的是这只是指平均情况。例如，金字塔的塔基平面就有可能是惟一的背面或正面。）

背面消除是对象精度的技术，时间花费与多边形个数呈线性关系，通过对要显示的对象进行预处理可得到亚线性的性能。例如，考虑一个立方体，其中心为坐标原点，每个面都垂直于坐标轴。从立方体外的任何一点看去，都最多有3个面可见。更进一步，立方体坐标系可划分为8个45°的区域，每个区域对应3个潜在的可见面。因此，视点在坐标系中的位置可用于选择这8个集合之一。对于面相对较少的对象，可预先建立一个表，这样，在进行可见面判定时不需要一改变视点就必须处理所有的面。

在对象具有很多面时由视点确定的可见面表可能很大。Tanimoto[TANI77]提出另一种基于帧相关性的图论方法。凸多面体的每个面对应一个节点构成一个图，有公共边的两个面对应的节点连接形成图的边。对初始的视点位置计算出分开可见面与不可见面的边的列表。该列表包括了所有形成对象轮廓的边。Tanimoto证明，当视点位置在帧与帧之间变化时，只有位于旧轮廓与新轮廓之间的面的可见性需要重新计算。

15.2.5 空间划分

空间划分（spatial partitioning）（也称为空间细分（spatial subdivision））的思想是将一个大的问题分解成一些小的问题，基本方法是在预处理阶段将对象或它们的投影分成空间上具有相关性的若干组。例如，我们可以将投影平面划分成规则的二维矩形网格，并求出每个对象的投影位于哪些网格空间。只有投影位于相同网格的多边形需要判断其是否重叠遮挡。这一技术应用于[ENCA72；MAHN73；FRAN80；HEDG82]中。空间划分技术也可用于对空间对象添加

规则的三维网格。这样，判定对象与投影线相交的过程可通过判断投影线与网格分区的交，再判断对象是否位于分区中来提高速度（15.10节）。

如果对象的空间分布非常不均匀，采用适应性划分会更有效，在适应性划分中每个分区的大小可变。一种可行的方法是递归地细分空间直到达到某一终止条件为止，如分区中对象少于某一分区的最大对象个数时细分终止[TAMM82]。12.6节中的四叉树、八叉树以及BSP树数据结构非常适合于这一要求。 664

15.2.6 层次结构

在第7章中已经讨论过，层次对联系各种结构以及不同对象的运动起着非常重要的作用。一种嵌入式的层次模型是将每个子节点作为其父节点的一部分。该结构能有效地限制可见面算法中对象比较的次数[CLAR76；RUBI80；WEGH84]。层次结构中某一层次的对象包含了它的所有子对象，可看成是其子对象的范围（如图15-18所示）。这时，如果某一层中的两个对象不相交，那么这两个对象各自的较低层次的对象之间就不需要进行相交测试。类似地，只有当投影线穿过层次中的某一对象时，才需要对该对象的子对象进行测试。这种层次结构的使用是体现对象相关性的一个重要实例。层次的自动构建方法将在15.10.2节中讨论。

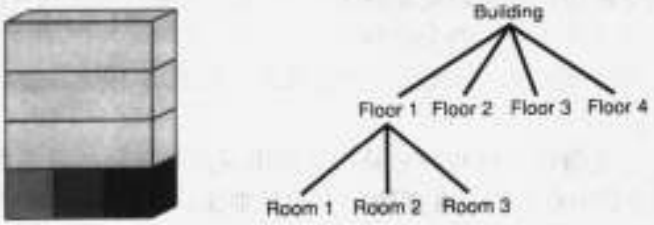


图15-18 层次结构可用于限制对象比较的次数。仅当光线与Building和Floor1相交时才需要同Room1到Room3进行相交测试

15.3 可见线判定算法

至此，我们已讨论了一些一般性的技术，下面介绍一些可见线与可见面的算法，看这些技术是如何得到应用的。首先从可见线算法谈起。这里给出的算法全都是对象精度的，且以一系列适于向量显示的可见线段作为输出结果。后面讨论的可见面算法也可用于可见线的判定，只须在绘制时用背景颜色绘制多边形的内部而用线的颜色绘制多边形的边。大多数可见面算法生成图像精度的点阵数据，而不是对象精度的边的列表。

15.3.1 Roberts算法

最早的可见线算法由Roberts[ROBE63]提出。该算法要求每一条边都是凸多面体的面的一部分。首先，用背面消除方法除去多面体所有背面对的公共边；然后，剩下的每一条边与每一个可能遮挡它的多面体相比较。许多多面体都能通过范围测试被去除：它们的投影范围在x或y上可能不重叠，或一个对象的边界在z轴上位于另一对象的后面。测试过的多面体依次与边进行比较。由于是凸多面体，对任何一条边而言，最多只有一组相邻点被某一多面体遮挡。因此，一个多面体要么完全遮挡这一边，要么留下边的一段或两段未被遮挡。这些剩下的未被遮挡的段继续与下一多面体进行比较。 665

Roberts的可见性测试采用的是从视点到被测边上一点的参数形式的投影线。当投影线穿过多面体导致端点不可见时，其投影线通过多面体的直线方程用线性规划方法求解。当投影线上的某点位于多面体的正面上时，该投影线穿过多面体。Rogers[ROGE85]给出了Roberts算法的

详细解释并提出了几种改进算法的方法。

15.3.2 Appel算法

一些更常用的可见线算法[APPE67；GALI69；LOUT70]只要求线是多边形的边，而不是多面体的边。这些算法也只考虑构成正面的边，并且利用到Appel算法中较为典型的边相关性。Appel[APPE67]定义了线上一点的不可见量（quantitative invisibility），即所有遮挡该点的正面多边形的个数。当线从后面每穿过一个正面多边形时，它的不可见量加1；而当它从后面穿出其前向多边形时，不可见量减1。一条线仅当它的不可见量为0时可见。图15-19中线段*AB*的每一段都标上了不可见量，如果不允许多边形互相贯通，线的不可见量只有在穿过

666 Appel称为轮廓线的部分时才发生改变。轮廓线可以是正面与背面共享的边，也可以是非封闭多面体的前向多边形的非共享边。两个前向多边形的共享边不改变线的可见性，因此不属于轮廓线。在图15-19中，边*AB*、*CD*、*DF*与*KL*是轮廓线，而*CE*、*EF*与*JK*不是。

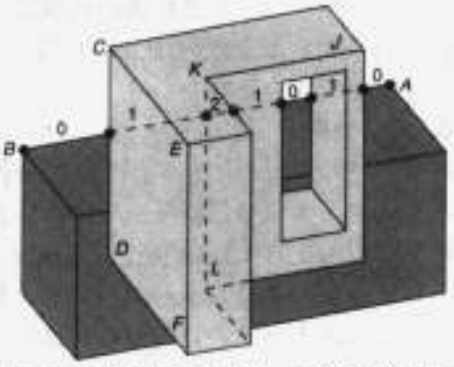


图15-19 线段的不可见量，虚线部分是不可见的。*AB*的投影与轮廓线的投影的交用黑点(•)表示。*AB*上每一段的不可见量也用数字标出

轮廓线穿过某一边之前可通过判断它是否穿过由视点与该边的两端点构成的三角形来确定。这需要用到7.12.2节中介绍的点在多边形中的判别法。由视点与轮廓线可确定一个平面，用该平面去裁剪要考虑的边，便可确定轮廓线在边上的投影。Appel算法要求所有多边形的边按与多边形一致的方向绘出。这样，不可见量改变的符号可由边与轮廓线的叉积的符号确定。

算法首先通过确认遮挡它的前向多边形的个数来确定对象的一个“种子”顶点的不可见量。方法是直接求其所有满足条件的正面，其条件是该正面与到“种子”点的投影线的交点比种子点本身近。然后利用边相关性，对由该点导出的边的不可见量进行修改，每经过一次轮廓线时增加或减少不可见量值。只绘出不可见量为0的部分。当到达线段的另一端点时，这一端点的不可见量再作为以该端点作为出发点的其他线段的初始不可见量值。

在轮廓线经过的顶点处，不可见量的变化需要进行一些修正。从该顶点导出的一条或多条线可能会被共享该顶点的一个或多个正面多边形所遮挡。例如，图15-19中，边*JK*的不可见量为0，而边*KL*的不可见量为1，因为对象的顶面遮挡了它。这一顶点处的不可见量的改变可通过测试边与共享顶点的正面多边形的遮挡关系来实现。

对于Appel算法这样的需要处理多边形相交的算法而言，需要计算边与正面多边形的交，并用这些相交结果来增减不可见量。由于可见线算法总是在边与边或边与对象之间进行比较，采用空间划分方法将会非常有效。每条边只需要与包含它的投影的网格单元中的边或对象进行比较。

15.3.3 光环线算法

任一种可见线算法都能很方便地实现将隐藏线绘制成用点、杠表示的虚线及低亮度的线或由其他显示设备支持的绘制风格。这样，隐藏线就能用合适的线型表现出来，而不是完全不显示。Appel、Rohlf与Stein[APPE79]另外提出一种绘制光环线的算法（图15-20）。每条线的两边

667 都有一层光环，这些光环能使其后面的线段部分虚化。与前面描述的算法不同的是，该算法并不要求每条线都是不透明多边形面的一部分。在视图平面上的线线相交处，位于后面的线的相

应部分被虚化。该算法对每条线与位于其前方的线求交，记录下被光环虚化的部分，然后在完成相交计算后绘出每条线的可见部分。如果光环比线与线之间的距离大，生成的效果便与普通的隐藏线消除算法相似，除非线的光环扩展到它所在多边形之外。

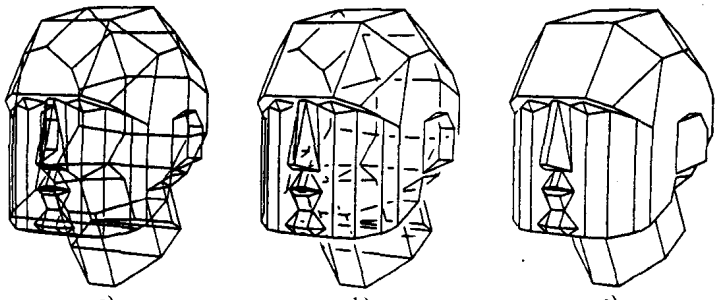


图15-20 三个头部模型。a)未消除隐藏线，b)隐藏线被光环虚化。c)隐藏线消除。（Arthur Appel, IBM T.J. Watson Research Center提供。）

在本章的其余部分，我们将讨论多种为可见面判定开发的算法。我们的重点是要计算出对象表面的哪些部分是可见的，至于表面颜色的判定需要留到第16章解决。在讲述每一个算法时，我们着重于其对多边形的应用，同时也指出何时能推广到处理其他对象。

15.4 z缓存算法

无论是从硬件还是软件的角度考虑，z缓存或深度缓存算法都是最简单的可见面判定算法。它最早由Catmull[CATM74b]提出，属于图像精度算法。该算法需要一个帧缓存F用于存储每一点颜色值；还需要一个同样大小的z缓存（z-buffer）Z用来存储每一点的z值。z缓存初始化为0值，代表后裁剪平面的z值；帧缓存初始化为背景颜色。z缓存中能存储的最大z值代表前裁剪平面。多边形按任意顺序扫描转换到帧缓存。在扫描转换过程中，如果多边形对应到(x, y)的点比缓存中现有的点离视点更近，那么用新点的颜色与深度值替代旧值。图15-21是z缓存算法的伪代码，读写缓存的过程在第3章中用的是WritePixel与ReadPixel过程，这里增加了ReadZ与WriteZ过程用于读写z缓存。 668

```
void zBuffer(void)
{
    int x, y;

    for (y = 0; y < YMAX; y++) {      /* 清除帧缓存和z缓存 */
        for (x = 0; x < XMAX; x++) {
            WritePixel (x, y, BACKGROUND_VALUE);
            WriteZ (x, y, 0);
        }
    }

    for (每个多边形) {                /* 画多边形 */
        for (多边形投影中的每个像素) {
            double pz = (在像素坐标(x, y)处的多边形的z值);
            if (pz >= ReadZ (x, y)) {   /* 新点并不是更远的 */
                WriteZ (x, y, pz);
```

图15-21 z缓存算法伪代码

```
                WritePixel ( x, y, 在像素坐标(x, y)处的多边形的颜色 );
            }
        }
    }
}  /* zBuffer */
```

图15-21（续）

不需要预排序，也不需要进行对象与对象之间的比较。对确定的x与y而言，整个过程的运算量仅相当于对集合$\{Z_i(x, y), F_i(x, y)\}$的搜索以找到最大的Z_i。z缓存与帧缓存总是记录每一(x, y)处当前最大z值所对应的信息。因此，多边形按它们被处理的顺序显示在屏幕上，扫描转换每一多边形时一次处理一条扫描线到缓存中（见3.6节）。图15-22是两个多边形的叠加，两个多边形分别用不同的明暗度表示；z值也显示在图中。

利用深度相关性及多边形是平面的事实，可以简化扫描线上每点z值的计算。一般而言，要计算z值，必须求解平面方程$Ax + By + Cz + D = 0$：

$$z = \frac{-D - Ax - By}{C} \tag{15-6}$$

如果在(x, y)处式(15-6)求得z_1，则在$(x + \Delta x, y)$处z值为：

$$z_1 - \frac{A}{C}(\Delta x) \tag{15-7}$$

式中A/C为常量，则给定$z(x, y)$时，求$z(x + 1, y)$只需要做一次减法，其中$\Delta x=1$。类似的增量计算可用
669 于求下一扫描线的第一个z值，只需要对每一Δy减去B/C。另外，如果面未确定或多边形非平面（见11.1.3节），$z(x, y)$可通过多边形沿扫描线上的一对边处z值的插值获得（图15-23）。此处也可使用递增计算方法。这里当一点的可见性判定条件未满足时，不需要计算该像素的颜色。因此，当像素的明暗计算很费时时，先将对象粗略地按从前到后的顺序进行排序，可以有效地先显示最近的对象。

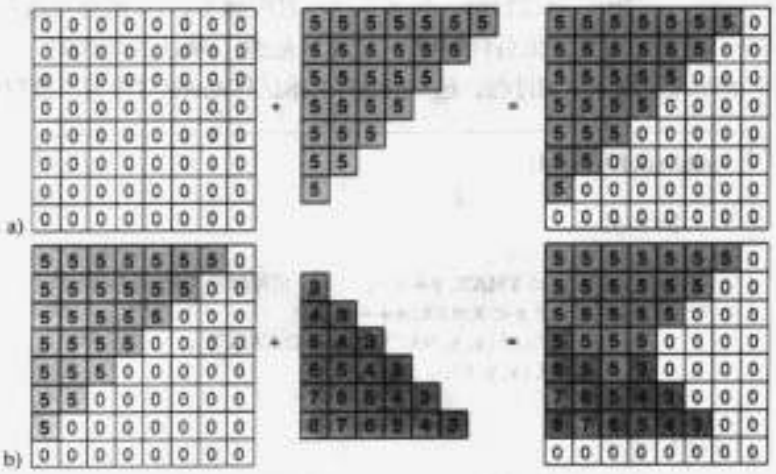


图15-22 z缓存。不同多边形覆盖的点用不同的灰度值表示，z值用数字标出。a)增加一个z为常量的多边形到空的缓存中，b)增加另一个多边形与第一个相交

z缓存算法并不要求对象由多边形构成。事实上，该算法最具吸引力的地方正是它能用于绘制任意的对象，只要该对象的投影的每一点处的明暗度及z值可以确定，从而不需要进行求交计算。

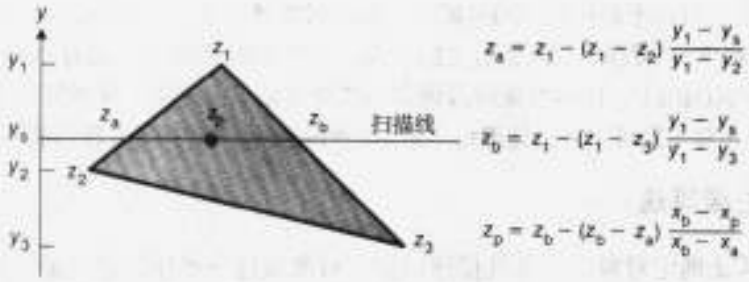


图15-23 多边形边与扫描线上z的插值。z_a由z_1与z_2之间插值获得；z_b由z_1与z_3之间插值获得；z_p由z_a与z_b之间插值获得

z缓存算法对x与y进行基本的排序，不需要进行比较运算。z排序在每一点处对包含该点的每个多边形只做一次比较。可见面计算需要的时间同对象中包含多边形的个数无关。因为，平均来看，当视见体中多边形个数增加时，每个多边形覆盖的点数却减少了。因此，搜索的每一集合的平均大小是固定的。当然，多边形增加带来的扫描转换量的增加也必须另外考虑。

尽管z缓存算法需要大量的存储空间，但实现起来却非常方便。当内存很紧张时，可以考虑将图像扫描转换成条状，只需要对处理的每一条图像准备足够的z缓存。缺点在于要对对象做多遍的处理。由于z缓存很简单且不需要附加的数据结构，随着内存造价的降低，许多基于z缓存的硬件和固件得到越来越广泛的应用，这在第18章中将详细讨论。因为z缓存算法在图像精度进行操作，很容易产生走样的问题。15.7节中的A缓存算法[CARP84]用未加权面积采样的离散逼近算法来解决这一问题。

z缓存算法的硬件实现常采用16位到32位的整数值，而软件（有些硬件）实现采用浮点值。尽管16位的z缓存对许多CAD/CAM应用已经足够了，但对于毫米级细节定义的对象分布在几公里范围的场景时就力不从心了。更糟的是，如果采用透视投影变换，变换除法带来的远距离z值的压缩效果会对深度排序和远距离对象的求交产生严重的影响。靠近视图平面时应该变换成不同的整数z值的两点，在距离很远时可能会变换成相同的z值（参见习题15.13和[HUGH89]）。

z缓存算法的精度有限，还会带来另一种走样问题。在绘制从不同端点出发的两共线边的公共部分时，扫描转换算法通常会作为两类不同的像素集来处理。这样，由于对z插值时可能会产生数值上的误差，这些边上相同的像素可能会得到略微不同的z值。这一点在绘制多面体的不同面的共享边时表现非常明显。在同一条边上，有些可见像素是某一多边形的一部分，而另一些像素却是另一相邻多边形的一部分。这个问题可以通过插入一些附加顶点以保证这些顶点位于公共部分的相同位置来解决。

在图像绘制出来以后，z缓存仍然有其存在的价值。因为可见面判定算法只用到这一种数据结构，可以将这种结构同图像保存在一起以便其他z值可计算的对象与其合并时使用。该算法也可以改写成当绘制选定的对象时z缓存内容不变。当z缓存用这种方式被屏蔽掉时，某一对象能够消除隐藏面并被绘制到一个独立的覆盖平面中（如果该对象是x与y的单值函数），并且能够在不影响z缓存内容的前提下被删除。这样，简单的对象，如规则网格等图形，能够在图像中的x，y与z方向移动，充当“3D指示”，并能与场景中的对象形成自然的遮挡与被遮挡关系。部分切除的效果也很容易实现，即根据z值是否位于某一切割平面之后来写z缓存与帧缓存。如果显示的对象对每一(x, y)有惟一的z值，那么z缓存的内容也可用于计算面积和体积。习题15.25解释了如何用z缓存来进行拾取。

Rossignac与Requicha[ROSS86]讨论了如何用z缓存算法处理CSG定义的对象。平面投影上的每一

670 ↓ 671

像素仅当它的z值更近且该平面构成CSG对象的一部分时才被写入。不同于在每一像素上只存储最近z值点，Atherton提出在z缓存中存入所有点的列表，这些点按z值排序，且存入每一点所属面的标识，构成对象缓存[ATHE81]。由后处理阶段确定图像如何显示。这样，许多特殊的效果（如透明、裁剪以及布尔集合运算）都可以通过处理每个像素的列表获得，而不用重新扫描转换这些对象。

15.5 列表优先级算法

列表优先级算法确定对象的可见性排序以保证对象按这一顺序绘制时能得到正确的图像结果。例如，如果对象之间在z方向上不存在重叠的情况，那么，我们只需要将对象按z值的增加进行排序，并按照这一顺序进行绘制。较近的多边形会覆盖较远的多边形，从而使近的对象遮挡住较远的对象。如果在z方向上存在重叠的现象，我们仍然有可能判断出正确的顺序，如图15-24a所示。如果对象之间循环重叠，如图15-24b和图15-24c所示，或互相穿透，那么就没有正确的顺序。这时，就有必要将一个对象分割成几部分以使得线性排序成为可能。



图15-24 z方向上多边形重叠的例子

列表优先级算法是对象精度算法与图像精度算法的混合。深度比较与对象分割都是对象精度的。只有扫描转换（它覆盖已绘制对象的像素依赖于图形设备的性能）是图像精度的算法。因为排序对象的列表是以对象精度生成，所以可以以任意的分辨率重复显示。正如我们将要看到的，列表优先级算法在以下几个方面有所不同：如何确定顺序，哪些对象需要被分割以及何时进行分割。具体而言，排序不一定是在z方向上，有些对象既没有循环重叠也没有互相穿透，
672 并且分割操作可能与观察者的位置无关。

15.5.1 深度排序算法

深度排序算法由Newell、Newell与Sancha[NEWE72]提出，其基本思想是按照逐步靠近视点的顺序绘制多边形到帧缓存。依次执行以下三步：

1) 按z坐标从小（远）到大（近）对所有的多边形排序。

2) 解决所有可能出现的排序上的歧义，这些歧义可能由z方向上的多边形重叠导致，必要时对多边形进行分割。

3) 按z坐标的升序（由远及近）扫描转换每一个多边形。

在SPHIGS中便使用了这种明确的优先级系统，它取代了最小z值的表示法，不再会有深度上的歧义，因为每一个优先级都对应于一个不同的常量z的平面。深度排序的简化形式常常称为画家算法，它很像是一个画家由远及近地绘画的过程，近的对象覆盖在远的上面。对象位于常量z的平面的环境，例如某些VLSI布局、制图以及窗口管理，称为是二维半，它们都可以正确地用画家算法处理。画家算法也可用于多边形并不位于常量z平面中的场景，此时的多边形按照最小的z值或其质心的最小的z值进行排序，上述的第二步可省略掉。尽管可以将场景构造成适于该算法的形式，但常常会出现排序不正确的情况。

图15-24给出了一些有歧义的情况，需要执行上述的第二步来加以解决。具体步骤如下：

假设当前排序列表的远端多边形为P。在该多边形扫描转换到帧缓存之前，必须与所有的在z方向上与P有重迭的多边形（设为Q）进行测试，以保证P不能遮挡Q从而使P能够在Q之前被写入。最多执行5次测试，一次比一次复杂度高。一遍成功后，即P不能遮挡Q时，下一个多边形Q继续进行测试。如果所有的多边形都通过了测试，那么P就开始扫描转换，列表中的下一多边形成为新的P。这5次测试是：

1) 多边形的x方向是否重叠？

2) 多边形的y方向是否重叠？

3) P是否完全位于Q平面的远离视点的一边？（图15-24a不成立，图15-25成立。）

4) Q是否完全位于P平面与视点相同的一边？（图15-24a不成立，图15-26成立。）

5) 多边形到(x,y)平面的投影是否不重叠？（这一点可以通过对多边形的边之间进行比较来判定。）

习题15.6提供了一种实现测试3和4的方式。

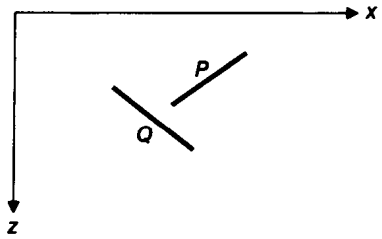


图15-25 测试3为真

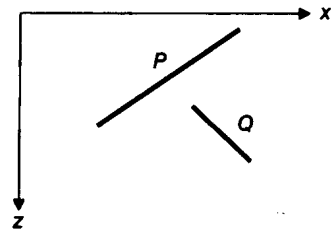


图15-26 测试3为假，而测试4为真

如果所有5项测试都失败，可假定此时P遮挡Q，因此需要测试Q是否能够在P之前扫描转换。测试1、2和5并不需要重复执行，但测试3和4中的多边形要做一次交换：

3') Q是否完全位于P平面的远离视点的一边？

4') P是否完全位于Q平面与视点相同的一边？

在图15-24a中，测试3'为真。因此，可以将Q移动到列表的顶端使之成为新的P。而在图15-24b中，测试仍然不能得出结论。事实上，P与Q仍然不能按照正确的顺序完成扫描转换。这时，P或Q中必须有一个被对方的平面所分割（见3.14节中有关多边形裁剪，将裁剪边作为裁剪平面）。初始的未分割多边形被舍弃，分割后的碎片按照正确的z排序插入列表，算法同前面一样继续进行。

图15-24c是一种更复杂的情形。P，Q与R中任一个多边形都可以移动到列表的顶端，使其与另一多边形确定正确的顺序，但不能保证同另两个都形成正确的顺序。这将产生一个无限的循环。为了避免循环处理，必须对移动到列表顶端的多边形进行标记。这样，当最初的5次测试失败且当前的多边形Q被标记时，测试3'与4'就不需要执行，而是将P或Q做分割（就好像测试3'与4'都失败了），再将分割后的碎片重新插入。

有没有可能出现两个多边形已经有了正确的排序但所有的测试都失败的情况呢？我们看看图15-27a中的P和Q。这里仅显示出了每个顶点的z坐标。P和Q位于当前的位置时，简单的画家算法与完全深度排序算法都首先扫描转换P。现在，顺时针旋转Q，旋转时保持Q在它的平面中，直到它开始遮挡P，但不允许P和Q自身相交（图15-27b）。（你也可以很容易地用两手模拟P和Q，掌心对着自己。）P和Q在z方向上有重叠部分，因此，必须进行比较运算。注意到测试1和2（x和y方向）均失败；测试3和4也失败，因为没有哪个多边形在另一个的半空间中；测试5失败，因为投影重叠。由于测试3'和4'也失败，即使P能够在Q之前被扫描转换，多边形也将被分割。

673 ~ 674

因为P有最小的z坐标，简单画家算法能首先正确地绘出P，但可以用另两个值再尝试一下：P的底端为$z=-0.5$，顶端为$z=0.5$。

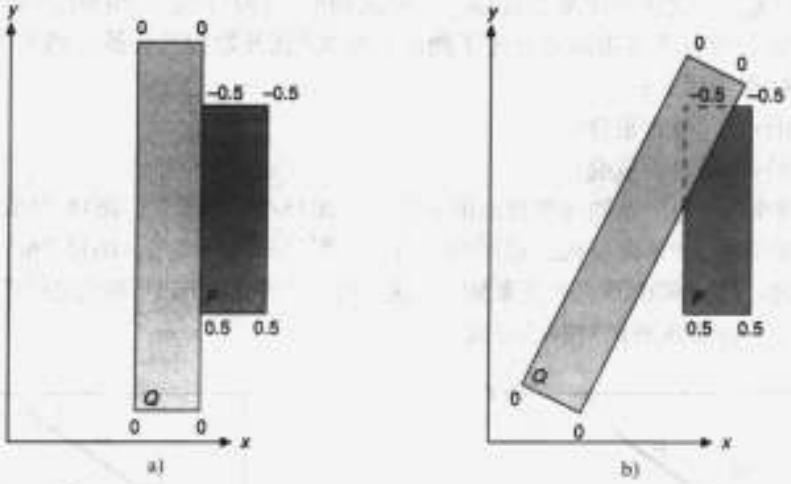


图15-27 正确排序的多边形可能被深度排序算法分割。多边形顶点处标出其z值。a)多边形P和Q未分割便直接扫描转换，b)多边形P和Q排序正确但5次测试均失败

15.5.2 二元空间划分树

二元空间划分树（BSP树）算法由Fuchs、Kedem和Naylor[FUCH80；FUCH83]提出，是一种非常有效的计算静态三维多边形集合之间可见性关系的算法，适用于视点任意变化的情形。该算法以费时费空间的预处理过程为代价，获得一个线性的显示算法，每次视点变化时执行一次该线性算法。该算法非常适用于视点变化，而场景对象不变的情形。

BSP树算法是基于Schumacker的工作[SCHU69]，基本思想是将环境看成是由很多的簇
675 （面的集合）组成，如图15-28a所示。如果能找到一个平面将一组簇与另一组簇完全分开，那么，与视点位于同一边的簇可能遮挡另一个簇，而不可能被另一个簇所遮挡。这些簇本身又能够找到一个平面递归地划分下去。如图15-28b所示，这样一种对环境的划分能够用一棵二叉树来表示，它的根是首次选取的划分平面。树的内部节点是划分平面，叶节点是空间区域。每一个区域都对应一个惟一的次序，使得当视点位于该区域时，簇之间能确定正确的遮挡关系。判定视点位于哪一区域的过程很简单，只需要将视点从根开始向下遍历整棵树，并通过与每个节点处的划分平面进行比较来选择左子树或右子树。

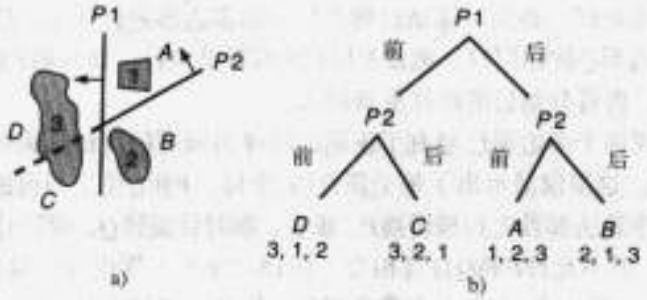


图15-28 簇优先级。a)簇1到3被划分平面$P1$和$P2$分割，得到A到D四个视点可能位于的区域。每个区域有不同的簇优先级；b) a)的二叉树表示[SUTH74a]

Schumacker在簇中选择面的原则是使每一个面都有一个与视点无关的优先级，如图15-29所

示。在相对于视点的背面被消除后，当面的投影相交时，具有较小优先级的面遮挡较大优先级的面。对任何一个像素而言，要显示的正确的面是投影覆盖该像素的最高优先级的簇中具有最高优先级（最小的数字编号）的面。Schumacker用专门的硬件来判定每一像素处位于最前面的面。另外，可以将簇按照其优先级递增的次序显示出来（基于视点），每一簇中的面也可以按照面的优先级递增的次序显示出来。BSP树方法并不需要对这两部分分别计算以求得正确的顺序，它采用扩展的Schumacker算法计算簇的优先级。基本思想是基于这样一个事实：对于一个多边形而言，如果位于它的远离视点一边的所有多边形最先被显示；然后显示它自身；最后显示它的位于视点一边的多边形，那么该多边形一定能够保证其扫描转换的正确性（即不会产生与其他多边形不正确的重叠情况）。我们需要保证这一点对每一个多边形都成立。 676

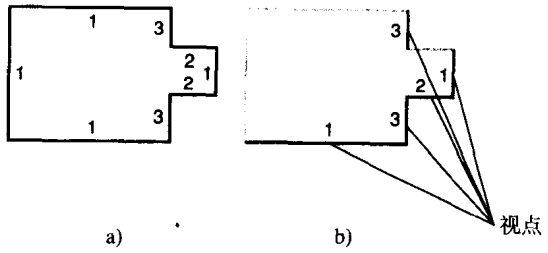


图15-29 面的优先级。a)簇中的面和它们的优先级，较小的数字表示较高的优先级；b)可见面的优先级[SUTH74a]

BSP树算法通过构造一棵多边形的二叉树（即BSP树）使多边形之间的顺序非常容易判定。BSP树的根是从所有待显示的多边形中选择出来的，无论选择哪一个多边形，算法都能正确运行。根多边形用于将整个环境划分成两个半空间。一个半空间包含所有位于根多边形之前的多边形，即与它的表面法向量对应的一边；另一半空间包含根多边形之后的多边形。同时位于前半空间与后半空间的多边形被根多边形所在平面分割，所得的两部分分别归入相应的半空间。在前后半空间中又可以分别选出一个多边形作为根节点的前后子节点，每一个子节点再递归地用于划分各自半空间中的剩余多边形，依此类推。当每个节点都只包含一个多边形时，算法结束。BSP树构造过程的伪代码见图15-30，图15-31是构造出的一棵树。

```
typedef struct {
    polygon root;
    BSP_tree *backChild, *frontChild;
} BSP_tree;

BSP_tree *BSP_makeTree (polygon *polyList)
{
    polygon root;
    polygon *backList, *frontList;
    polygon p, backPart, frontPart;      /* 假设每个多边形都是凸多边形 */

    if (polyList == NULL)
        return NULL;
    else {
        root = BSP_selectAndRemovePoly (&polyList);
        backList = NULL;
        frontList = NULL;
        for ( polyList中剩余的每个多边形 ) {
            if ( 多边形P在根的前面 )
                BSP_addToList (p, &frontList);
            else if ( 多边形P在根的后面 )
                BSP_addToList (p, &backList);
```

图15-30 构造BSP树的伪代码

```
            else {                          /* 多边形P必须被分割 */
                BSP_splitPoly (p, root, &frontPart, &backPart);
                BSP_addToList (frontPart, &frontList);
                BSP_addToList (backPart, &backList);
            }
        }
        return BSP_combineTree (BSP_makeTree (frontList),
                                root,
                                BSP_makeTree (backList));
    }
} /* BSP_makeTree */
```

图15-30（续）

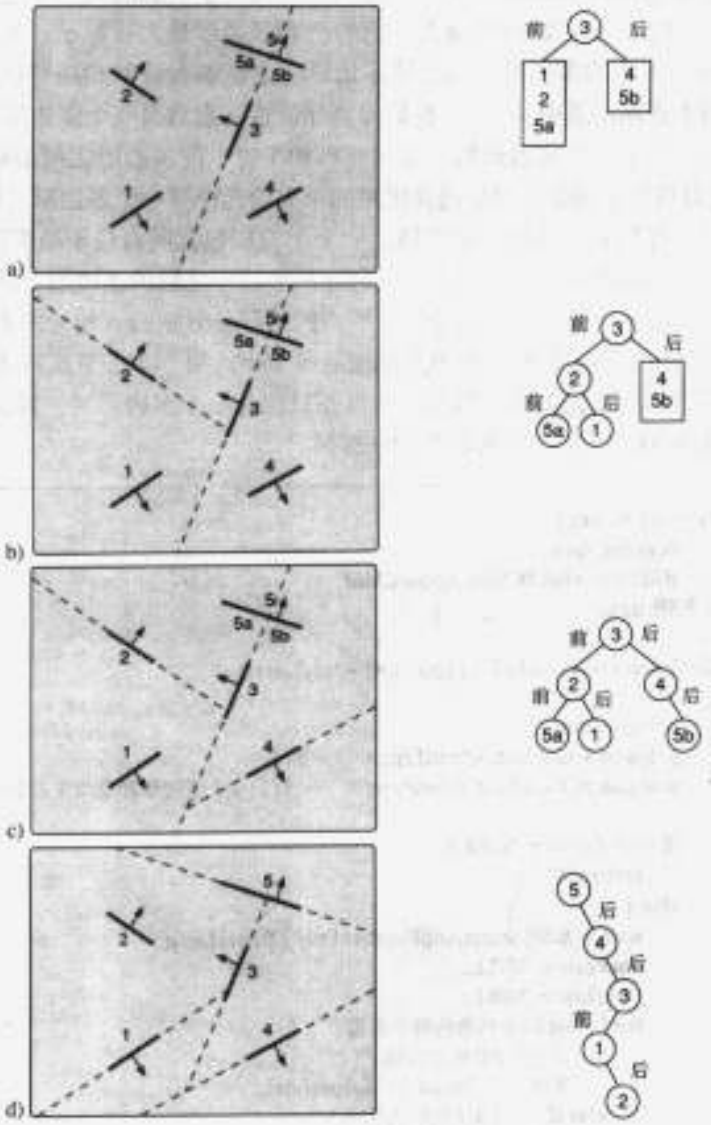


图15-31 BSP树。a)场景顶视图与未递归的BSP树，其中多边形3为根，b)构造左子树，c)完整的树，d)以多边形5为根的另一棵树（基于[FUCH83]）

显然，对于给定的任意视点，我们都可以通过按一定次序遍历BSP树的方法获得依优先级正确排序的多边形列表。我们可以从根多边形看起，它将其余的多边形分成两个集合，每一个都完全位于该多边形所在平面的一边。因此，算法只需要保证各集合之间显示的相对次序正确即可，即两个集合的多边形之间互不影响，且根多边形能够正确地显示并保持与其他集合之间的正确次序。如果视点位于根多边形的前半空间，算法必须先显示后半空间的多边形（那些可能被根遮挡的多边形），然后显示根，最后显示前半空间的多边形（可能遮挡根的多边形）。反之，如果视点位于根多边形的后半空间，算法必须先显示前半空间的多边形，然后显示根，最后显示后半空间的多边形。若多边形的法向量垂直于视线，则任一次序显示都可满足。当视点位于某一多边形的后半空间时，可以不显示这个多边形以实现背面的消除。根节点的每一个子节点也都递归地得到处理。显示一棵BSP树的伪代码如图15-32所示，图15-33则描述了图15-31c中的树在两种不同投影下的遍历方式。

```
void BSP_displayTree (BSP_tree *tree)
{
    if (tree != NULL) {
        if ( 视点在tree->root之前 ) {
            /* 显示后子节点、根和前子节点 */
            BSP_displayTree (tree->backChild);
            displayPolygon (tree->root);
            BSP_displayTree (tree->frontChild);
        } else {
            /* 显示前子节点、根和后子节点 */
            BSP_displayTree (tree->frontChild);
            displayPolygon (tree->root);      /* 仅当不需要背面消除时 */
            BSP_displayTree (tree->backChild);
        }
    }
}  /* BSP_displayTree */
```

图15-32 显示BSP树的伪代码

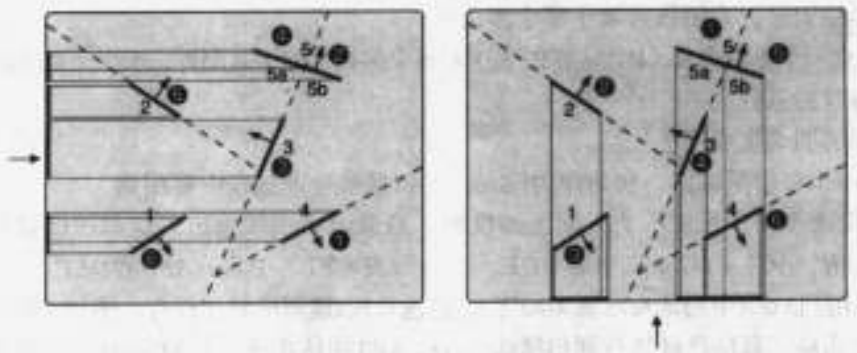


图15-33 BSP树的两次遍历，对应于两个不同的投影。投影线用细线表示，白色的数字表示画的顺序

每个多边形的平面方程在必要时可以进行变换，多边形顶点的变换由displayPolygon过程实现。BSP树对三维裁剪也有帮助。所在平面与视见体不相交的多边形必然有一子树完全位于视见体之外，因此在裁剪时不需要对其做进一步的考虑。

子树的根多边形的选择直接影响算法的效率。理想情况是选取的多边形能以最少的分割生

成BSP树。一个更容易实现的思路是选择一个多边形，使其对子节点多边形的分割最少。经验表明，从多边形集合中选取少量（5个到6个）多边形进行以上测试，选取其中的最优解，就能够很好地逼近全局的最优解[FUCH83]。

与深度排序算法类似，BSP树算法中的插入和排序完全是对象精度的，同时也依赖于光栅设备的图像精度的填充能力。不同的是，所有多边形的分割都在预处理阶段完成，并且只有当环境发生改变时才需要重新执行。值得注意的是多边形分割的次数可能比深度排序算法要多。

列表优先级算法允许使用硬件的多边形扫描转换器，从而在速度上大大优于在每点判断z值的软件算法。深度排序算法和BSP树算法按由后向前的顺序显示多边形，可能会遮挡住较远的对象，所以，和z缓存算法类似，每一个像素的明暗计算可能执行多次。反之，多边形也能按从前往后的顺序显示，这时，多边形上的像素只有在尚未写入时才被显示。

在用列表优先级算法实现隐藏线消除时，必须特别注意由细分过程带来的新边。如果这些边和初始的多边形边一样进行扫描转换的话，就会出现不希望看见的效果，因此必须对其做上标记，以便不参加扫描转换。

15.6 扫描线算法

677
~
680

*扫描线算法*最早由Wylie、Romney、Evans和Erdahl[WYLI67],Bouknight[BOUK70a；BOUK70b]与Watkins[WATK70]等人提出，属于图像精度算法，每次处理一条扫描线而生成一幅图像。它的基本思想是3.6节中的多边形扫描转换算法的扩展，同时运用了各种形式的相关性，如扫描线相关性和边的相关性。差别在于我们不只处理一个多边形，而是多边形的集合。算法的第一步是对投影到视平面上的所有多边形的非水平边生成一个*边表*（ET）。同前面一样，水平边不予考虑。ET中的项基于每一边的较小的y坐标进行桶排序，在每个桶中的边又按照较低端点的x坐标递增的顺序排序。每一项包括：

1) 具有较小y坐标一端的x坐标。

2) 边的另一端的y坐标。

3) 从一条扫描线到另一条扫描线的x增量，Δx（Δx是边斜率的倒数）。

4) 多边形标识号，表明该边属于哪个多边形。

还需要建立一个*多边形表*（PT）。在PT表中对每个多边形除了其ID外，至少还包含以下内容：

1) 平面方程的系数。

2) 多边形的明暗度或颜色信息。

3) 一个in-out布尔量标志，初始化为false，在扫描线处理过程中要用到。

图15-34是两个三角形在(x,y)平面上的投影，隐藏边用虚线画出。这幅图经过排序的边表（ET）中含有*AB*，*AC*，*FD*，*FE*，*CB*与*DE*。多边形表（PT）中含有*ABC*和*DEF*。

这里还要用到3.6节中的*活动边表*（AET）。它也是按x递增的顺序保存。图15-35中列出了ET、PT和AET项。当前，算法已经执行到扫描线$y=\alpha$，AET中依次序含有*AB*和*AC*。边从左到右进行处理。在处理*AB*时，首先转换多边形*ABC*的in-out标志。此时，标志变为true；于是，扫描是进入（“in”）多边形，要对多边形进行处理。现在，由于扫描是进入惟一的一个多边形*ABC*，它一定是可见的，因此，*ABC*的明暗度可以应用到AET表中从*AB*到*AC*的跨段上。这是一个利用跨段相关性的实例。在边*AC*上，*ABC*的标志变为false，因此扫描不再是进入（“in”）任何多边形。更进一步，因为*AC*是AET表中的最后一条边，扫描线处理完成。然后AET根据ET进行修改并再次按x排序（因为它的有些边可能已经交叉），继续处理下一条扫描线。

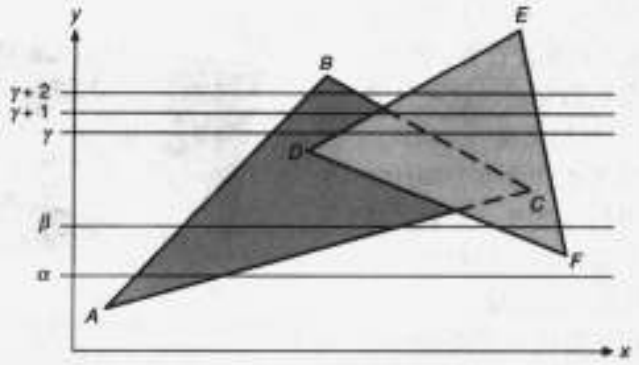


图15-34 用扫描线算法处理两个多边形



AET内容

扫描线	项			
α	AB	AC		
β	AB	AC	FD	FE
γ, γ+1	AB	DE	CB	FE
γ+2	AB	CB	DE	FE

图15-35 扫描线算法中的ET、PT和AET

当遇到扫描线$y=\beta$时，AET表的顺序是*AB*，*AC*，*FD*和*FE*。处理和前面大同小异。扫描线上有两个多边形，但扫描一次仅进入一个多边形。

扫描线$y=\gamma$的情况较为复杂。进入*ABC*时它的标志变为true。*ABC*的明暗度用于直到下一条边*DE*的跨段。这时，*DEF*的标志也变为true，因此，扫描进入（"in"）两个多边形。（有必要保存一个in-out标志为true的多边形的列表以及列表中含有多边形的个数。）现在我们必须判定*ABC*和*DEF*中哪一个离视点更近。方法是根据两个多边形所在平面的方程求出z值，其中$y=\gamma$，x等于直线$y=\gamma$和边*DE*的交点处的x值。该x值是AET表中*DE*项的值。在这个例子中，*DEF*的z值较大，因而是可见的。因此，在假定不存在多边形穿透的情况下，直至边*CB*的范围内应该用*DEF*的明暗度。到达边*CB*的这一点处，*ABC*的标志变为false，且扫描再次进入惟一的多边形*DEF*，继续使用它的明暗度直到边*FE*。图15-36显示两个多边形与$y=\gamma$平面的关系；两根粗线是多边形与平面的交。

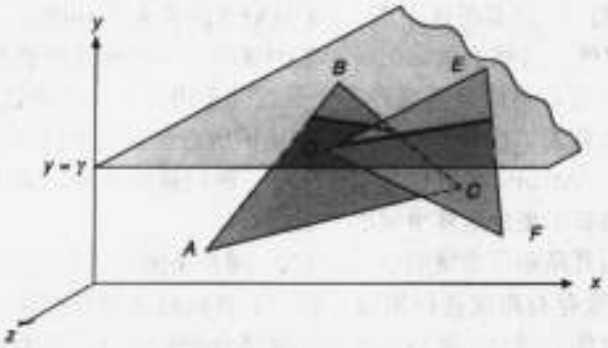


图15-36 多边形*ABC*与*DEF*和平面$y=\gamma$的交

681
↓
682

假设有一个大的多边形*GHIJ*位于*ABC*和*DEF*之后，如图15-37所示。那么当扫描线$y=\gamma$遇

到边*CB*时，扫描仍然是进入（"in"）多边形*DEF*和*GHIJ*，这时要再次完成深度计算。但是，如果我们假设多边形不会互相穿透，这种计算就可以避免。这意味着，当扫描离开*ABC*时，*DEF*与*GHIJ*之间的深度关系不会改变，即*DEF*继续位于前面。因此，当扫描离开一个被遮挡多边形时，深度计算是不必要的，只有当离开一个遮挡多边形时，才需要计算深度。

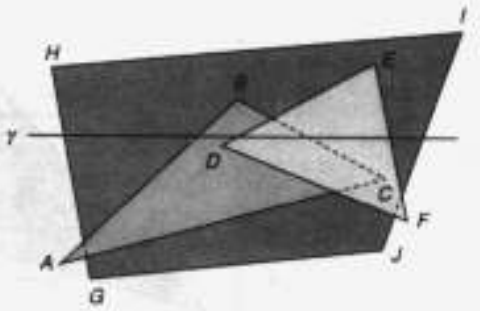


图15-37 三个非穿透性多边形。因为非穿透性多边形保持其相对的z的顺序，在扫描线γ离开被遮挡多边形*ABC*时不需要进行深度计算

要对穿透多边形使用该算法，需要做些变化。如图15-38所示，将*KLM*分割成*KLL'M'*与*L'MM'*。这里导入一条假边*M'L'*。另外，算法要经过适当的修改以找到扫描线上的穿透点。

这一算法的另一个修改是使用深度相关性。Romney注意到，假设多边形互不穿透时，如果一条扫描线上的位于AET表中的边与紧接着的前一条扫描线相同，且它们的顺序也相同，那么这条扫描线上各个部分的深度关系都没有发生改变，也就没有必要进行新的深度计算[ROMN68]。前一扫描线上可见跨段的记录也定义了当前扫描线的跨段。图15-34中的扫描线$y=\gamma$和$y=\gamma+1$就是这种情况。两者都是从*AB*跨段到*DE*和从*DE*跨段到*FE*的部分可见。然而从$y=\gamma+1$到$y=\gamma+2$就不再满足深度相关性，因为边*DE*和*CB*在AET表中改变了次序（算法中必须作处理）。可见跨段也因此改变成从*AB*到*CB*和从*DE*到*FE*。Hamlin与Gear[HAML77]讨论了在AET表中边的次序改变的情况下仍能保持深度相关性的情形。

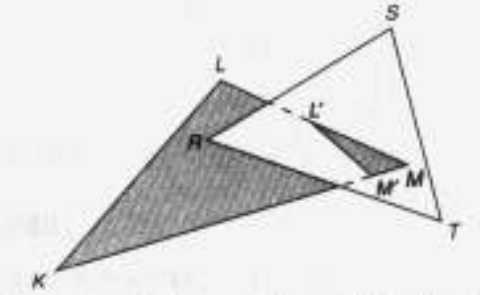


图15-38 多边形*KLM*在线*L'M'*处穿过多边形*RST*

683

我们还没有讨论如何处理背景。最简单的方法是将帧缓存初始化成背景颜色，这样算法就只需要处理与边相交的扫描线。另一种办法是在场景中定义一个足够大的多边形，且令其比场景中所有其他的多边形都远，并平行于投影平面，然后给这个多边形赋予预期的明暗度。最后的办法是当扫描未进入任何多边形时直接在帧缓存中写入背景色。

尽管到目前为止我们处理的都是多边形，扫描线算法的使用可以非常广泛，包括更加一般性的面（15.9节中讨论的）。为实现这一点，ET和AET由*面表*（surface table）和*活动面表*（active-surface table）取代，这些表按面的(x,y)范围排序。一个面从面表移动到活动面表需要执行附加的操作。例如，可以将面分解成许多个逼近的多边形，这些多边形在扫描离开面的y轴范围时将被丢弃，这就省去了在整个绘制过程中维护所有面数据的麻烦。通用的扫描线算法的伪代码如图15-39所示。Atherton[ATHE83]讨论了一种扫描线算法，用于绘制由结构实体几何（CSG）元素进行正则布尔集合运算生成的对象。

有一种扫描线算法因其简单而非常吸引人，它用z缓存来解决可见面问题[MYER75]。算法采用单扫描线帧缓存和z缓存对跨度进行累加，每一条新扫描线都要清除一次缓存。因为只需要一条扫描线的存储空间作为缓存，所以该算法非常适合于绘制高分辨率的图像。

Crocker[CROC84]用一个扫描线z缓存发现了他称之为*不可见相关性*的特性，即在一条扫描线上不可见的面满足一定条件时在下一条线上也不可见。给定扫描线，在为其构造一个

活动面表的同时，也创建一个独立的不可见面表。将一个面加入到不可见面表中的条件是：该面在当前扫描线上的最大z值要小于前一条扫描线在面的最小和最大x值处的z缓存中的z值。例如，给定一个立方体与包含的三角形（见图15-40a），三角形和前一条扫描线的z缓存的内容投影到(x,z)平面（见图15-40b）。三角形的z_{max}小于前一扫描线在三角形的x_{min}和x_{max}处的z缓存的值，于是，三角形被添加到不可见面表中。将面放入不可见面表能够在大多数的可见面处理中不考虑它。不可见面表中的有些面可能是不该属于该表的。为了修正这一点，在扫描线上处理每一点时，如果面的最大z值大于在该点被判定为可见面的z值，那么这些面要从不可见面表中删除，并加入到活动面表中。例如，尽管图15-40c中的三角形被放到了不可见面表中，它实际上是可见的，因为立方体被裁剪了。因此，要把它从不可见面表移到活动面表。

```
向面表中加入面；
初始化活动面表；

for（每条扫描线）{

    更新活动面表；

    for（扫描线上的每个像素）{
        判定活动面表中投影到像素的面；
        找出最近的这种面；
        确定在该像素处最近面的明暗度；
    }
}
```

图15-39 一般扫描线算法的伪代码

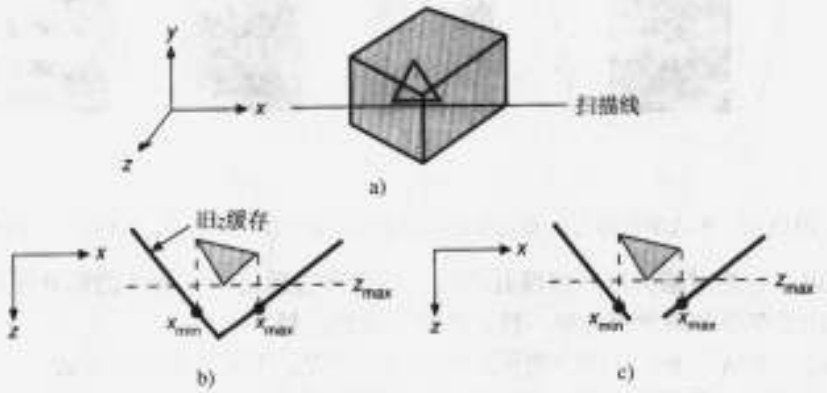


图15-40 不可见相关性。a)盒中的三角形，b)三角形正确放置到不可见表中，c)三角形被不正确地放置到不可见面表中。（基于[CROC84]。）

Sechrest与Greenberg[SECH82]提出了一种适于不相交多边形的对象精度算法，其基本思想仍然是扫描线算法。该算法基于这样一个事实：边的可见性只有在顶点处以及边的相交处才会发生改变。将顶点和边的交点按y排序，有效地把场景分成水平的带状区域，这些区域中的可见性关系是不变的（见图15-41）。在处理每一条区域时，区域中可见边部分的对象精度坐标计算出来，再附加上一些信息，通过这些信息，扫描转换时就能够重构出可见多边形的轮廓。开

始时，只有局部最小的顶点依次序被处理。先生成一个AET表，每当扫描中遇到顶点时便进行修改。边的交点是在测试活动边相交性的过程中同时求得的。

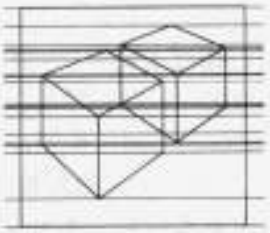


图15-41 Sechrest和Greenberg的对象精度算法在顶点和边的交点处将成像平面划分成水平带状。（经允许引自Stuart Sechrest and Donald P. Greenberg, Program of Computer Graphics, Cornell University, 1982。）

15.7 区域细分算法

区域细分算法与空间划分算法一样，都遵循分而治之的思想，只不过区域细分是针对投影平面的。投影图像的区域作为考察对象。如果很容易确定区域中的多边形可见，便显示这些多边形，否则，将该区域划分为更小的区域，递归地执行上述策略。区域变小后，会有较少的多边形覆盖每一个区域，直到最后能够作出决定。这种方法利用的是区域相关性，即一幅图像的足够小的区域最终会包含在至多一个可见多边形中。

15.7.1 Warnock算法

Warnock[WARN69]提出的区域细分算法将每一区域划分成四个相等的方块。在递归细分过程的每一步骤中，每个多边形的投影和感兴趣区域的关系可能为以下四种关系之一（见图15-42）：

1) 包围多边形完全包含感兴趣的（阴影）区域（图15-42a）。

2) 相交多边形与该区域相交（图15-42b）。

3) 内含多边形完全位于该区域中（图15-42c）。

4) 分离多边形完全位于该区域外（图15-42d）。

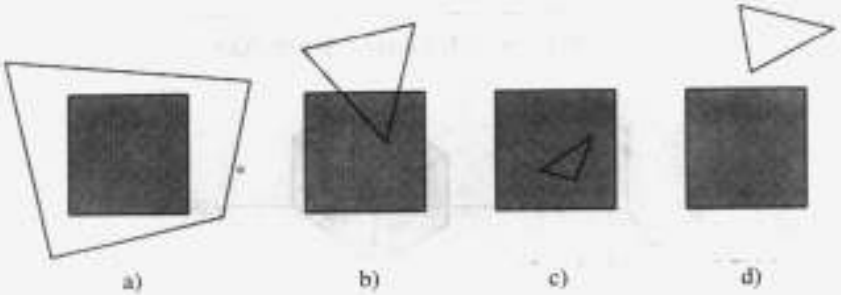


图15-42 多边形投影与区域元素的四种关系：a)包围，b)相交，c)内含，d)分离

分离多边形显然对感兴趣区域没有影响。相交多边形位于区域外的部分显然也没有影响，而位于区域内的部分和内含多边形一样，处理方法也一样。

在以下的四种情况下，区域中的可见性很容易判定，不需要再细分下去：

1) 所有的多边形都脱离区域，区域中只需要显示背景色。

2) 只有一个相交或内含多边形。该区域首先填充背景色，然后扫描转换多边形含在区域中的部分。

3) 含单个的包围多边形，但没有相交或内含多边形。该区域用包围多边形的颜色填充。

4) 多于一个多边形相交、内含于或包围该区域，但有一个包围多边形位于所有其他多边形之前。在所有多边形之前的判别方法是计算所有平面在区域四个角处的z坐标。如果存在一个包围多边形的z坐标比所有其他多边形大（近于视点），那么整个区域就可用该包围多边形的颜色填充。

情况1、2和3很容易理解。情况4的进一步说明见图15-43。在图15-43a中，包围多边形的四个交都比其余的交要近于视点（它处在 $+z$ 轴无穷远）。因此，整个区域都被填充上包围多边形的颜色。在图15-43b中，无法做出判定，尽管此时似乎包围多边形在相交多边形之前，但在左边相交多边形平面位于包围多边形之前。在深度排序算法中，如果相交多边形完全位于包围多边形的一边且包围多边形远离视点，就不需要再进一步的细分。而Warnock算法总是要对区域细分下去以简化问题的复杂度。细分之后，只有内含多边形和相交多边形需要重新考虑：原始区域的包围多边形和分离多边形仍然是其子区域的包围多边形和分离多边形。



图15-43 递归细分中情况4的两个例子。a)包围多边形在感兴趣区域的所有角都离视点最近；b)相交多边形平面在区域左边离视点最近，×标记出包围多边形平面的交，o标记出相交多边形平面的交，*标记出内含多边形平面的交

到目前为止，除了背景的扫描转换操作与四种情况下的多边形的分割之外，该算法都操作在对象精度。然而其图像精度的扫描转换操作也可由对象精度操作替代，从而输出可见面的精确表示：一块区域的大小（情况1、3和4）或裁剪到该区域的单个多边形以及它相对于区域的补集，代表背景的可见部分（情况2）。那么，如果不是这四种情况之一怎么办？一种方法是当达到显示设备的分辨率时便停止细分。这样，在一台1024×1024的光栅显示设备上，最多需要10次细分。如果已经执行了最大数目的细分，而四种情况都还未出现，那么需要计算在像素大小的不可见区域中心的所有相关多边形的深度。有最近 z 坐标的多边形用于确定该区域的明暗度。另外，出于反走样的考虑，可以继续对像素点进行细分，以分得子像素区域的大小作权值，从而求出该点的颜色。这些图像精度的操作只有在不满足简单情况时才执行，但正是这些图像精度的操作使得该算法成为图像精度算法。

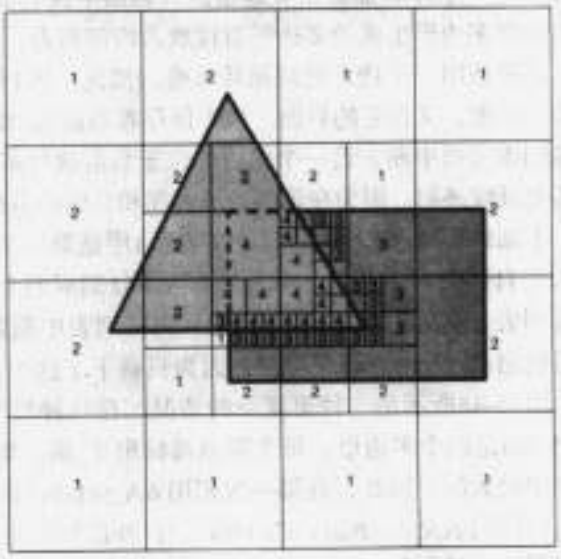


图15-44 区域细分为方块

图15-44是一个简单的场景和对场景的细分。每一细分区域中的数字代表四

种情况之一；在未标数字的区域，四种情况均不满足。可以将该算法同用四叉树做二维空间划分的算法（12.6.3节）进行比较。除了相等区域细分外，图15-45给出了另一种划分方法，即在多边形的顶点处（如果顶点位于区域中）做划分，以避免不必要的细分。这里为演示方便将深度限定为5。

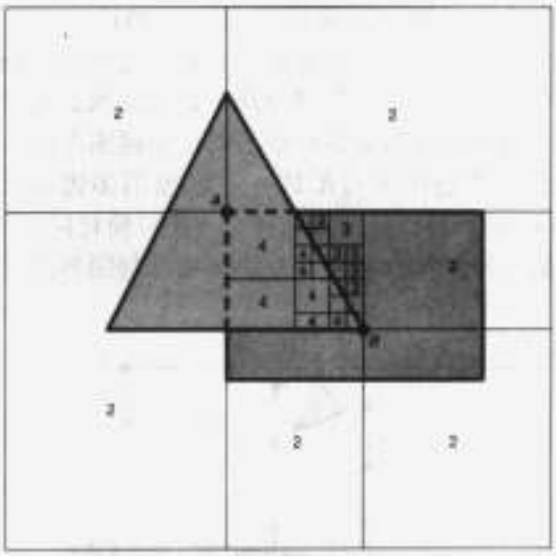


图15-45 在多边形顶点（圆点）处做区域细分。第一次在顶点A处，第二次在顶点B处

15.7.2 Weiler-Atherton算法

Warnock的算法不得不采用图像精度操作，其原因在于它只能将多边形裁剪成矩形区域。Weiler和Atherton[WEIL77]提出了另一种策略，不需要沿矩形边界细分，而是沿多边形的边界细分屏幕区域。这个算法需要一个十分有效的裁剪算法，能用一个多边形裁剪另一个有空洞的凹多边形（类似19.1.4节中的算法）。第一步，不是必须的但对提高效率非常有用，是将多边形按照某一z值排序，如按最近的z坐标排序。依这个标准离视点最近的多边形用于裁剪所有的多边形（包括该裁剪多边形），所得多边形分成裁剪多边形内部与外部两个列表。内部列表中的多边形位于裁剪多边形之后，是不可见的，因此要被删除。如果内部列表中存在一个多边形比裁剪多边形更近，说明初始排序不正确（事实上，当出现15.5节中的循环重叠情况时便不可能有正确排序）。每一个这样的多边形用于递归地裁剪内部多边形列表中的成员。递归细分结束后，显示内部列表。然后算法继续处理外部列表。算法总是用初始多边形作为裁剪多边形完成裁剪，而不采用裁剪后的片段，因为用初始多边形裁剪比片段裁剪要简单。这样，当一个多边形被裁剪时，每一个片断都要相关地指回到初始生成它的输入多边形。为进一步提高效率，裁剪算法将由裁剪多边形生成的多边形直接放入内部列表，不进行另外的测试。

684 ↕ 689

该算法用一个栈来处理循环重叠的情况（图15-24b）。循环重叠是指一个多边形既在其他多边形的前面，又在它的后面。栈中保存着当前用做裁剪多边形的多边形列表，这些多边形的使用因递归细分而中断。若一个多边形位于当前裁剪多边形之前，则需要确定它是否在栈中。若是，不需要继续递归，因为在该多边形内部和其后的多边形已被删除。该算法的伪代码见图15-46。

下面以图15-47为例说明。三角形A用做第一个裁剪多边形，因为它的最近的z值最大。将A放入它自己的内部列表；然后，矩形B分割成两个多边形：$B_{in}A$，加入内部列表；$B_{out}A$，加入外部列表。$B_{in}A$位于A之后，因此从内部列表中删除。现在，内部列表中已没有成员比A近，因此可输出A。下一步处理$B_{out}A$，因为只剩下了这个多边形，只需要简单地输出它即可。

图15-48所示是一种更复杂的情况，在这种情况中，初始排序（或任何其他排序）不正确。图15-48a是四个多边形，每个顶点都标出了z值。矩形A离视点最近，因为它的z最大值是所有多边形中最大的。因此，在第一次调用WA_subdivide时，A用于裁剪所有的多边形，如图15-48b所示。内部列表是A，$B_{in}A$，$C_{in}A$和$D_{in}A$；外部列表是$B_{out}A$，$C_{out}A$和$D_{out}A$。$B_{in}A$和$D_{in}A$在A之后，因而被删除，只剩下A和$C_{in}A$在内部列表中。$C_{in}A$在A平面的近端，但很明显该多边形排序不正确。因此，调用WA_subdivide完成递归细分，即用C分割当前的内部列表。C是出错多边形的源多边

```
void WA_visibleSurface (void)
{
    polygon *polyList = 所有多边形的副本的列表;
    按最大z值递减的顺序对polyList排序;
    清空栈;

    /* 处理其余的多边形区域 */
    while (polyList != NULL)
        WA_subdivide ( polyList中的第一个多边形, &polyList ) ;
}  /* WA_visibleSurface */

void WA_subdivide ( polygon clipPolygon, polygon**polyList )
{
    polygon *inList;          /* clipPolygon内的片段 */
    polygon *outList;         /* clipPolygon外的片段 */

    inList = NULL;
    outList = NULL;

    for ( *polyList中的每个多边形 )
        裁剪多边形为clipPolygon的祖先，将内部部分放在inList，
          外部部分放在 outlist;

    从inList中删除clip Polygon后面的多边形;

    /* 递归地处理未正确排序的片段 */
    for ( inList中不在栈中也不是clipPolygon一部分的每个多边形 ) {
        将clipPolygon压入栈中;
        WA_subdivide (polygon, &inList);
        弹出栈;
    }

    /* 显示clipPolygon内部的其余多边形 */
    for ( inList中的每个多边形 )
        显示多边形;

    *polyList = outList;      /* 从*polyList中减去inList */
}  /* WA_subdivide */
```

图15-46 Weiler-Atherton可见面算法的伪代码

形，如图15-48c所示。这一层递归的新内部列表是$A_{in}C$和$C_{in}A$；新的外部列表是$A_{out}C$。$A_{in}C$在C之后，故被删除。内部列表中只剩下$C_{in}A$；因为它是裁剪多边形的一部分，所以被显示出来。在递归调用WA_subdivide返回之前，*polyList*设成新的外部列表，其中仅包含$A_{out}C$。*polyList*是调用者的*inList*，因此下一步显示$A_{out}C$，如图15-48d所示，其中显示的片段用灰色标出。WA_subdivide的首次调用设置*polyList*为它的外部列表（$B_{out}A$、$C_{out}A$和$D_{out}A$）并返回。

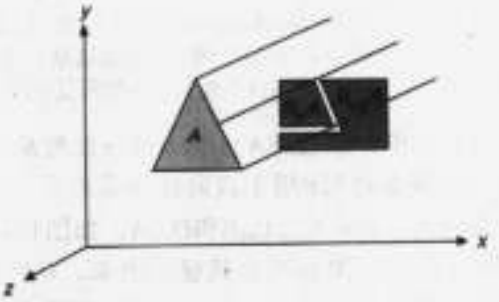


图15-47 用Weiler-Atherton算法实现简单场景的细分

690
|
691

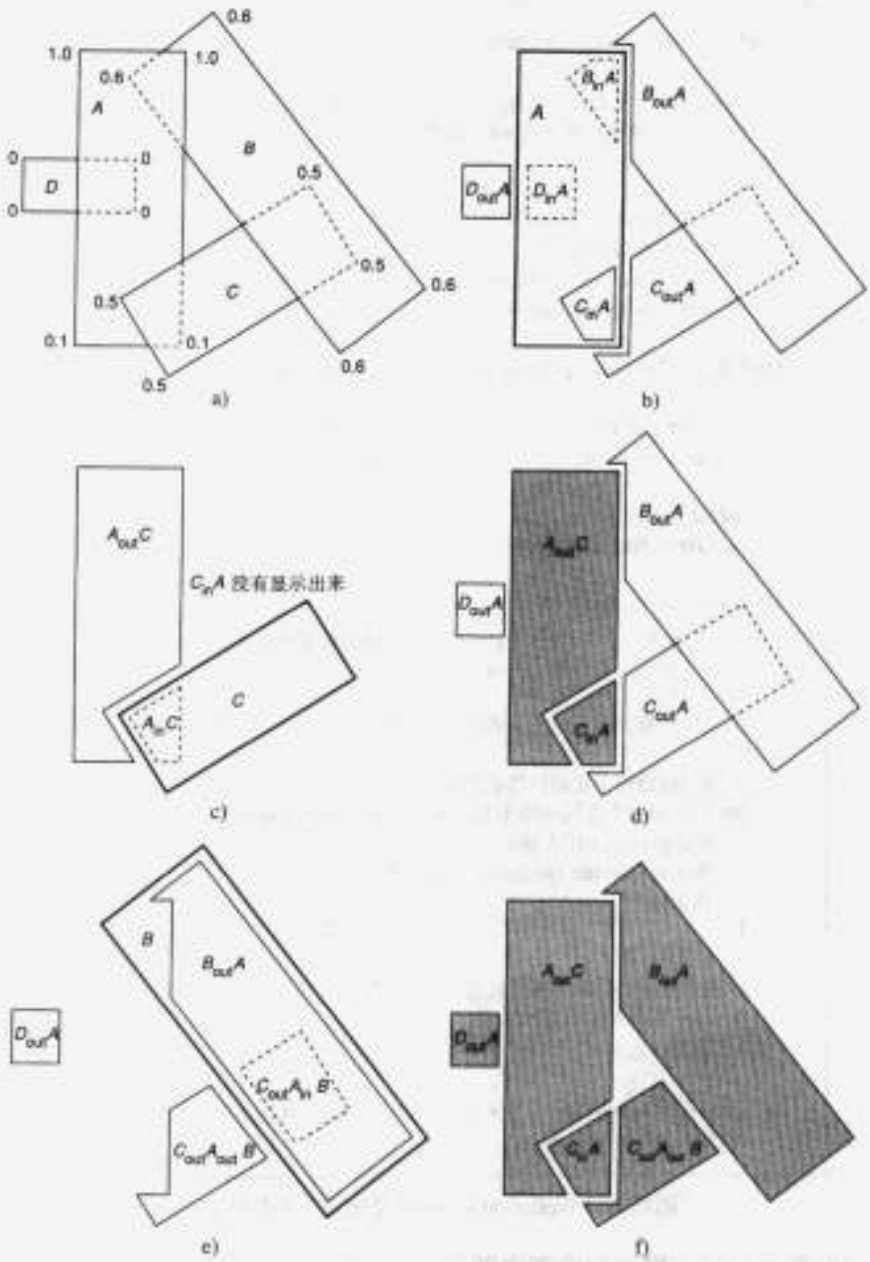


图15-48 用Weiler-Atherton算法实现递归。裁剪多边形用粗线标出。显示的多边形表示成灰色。数字是顶点的z值。a)初始场景，b)多边形被A裁剪，c)A的内部列表在递归细分过程中被C裁剪，d)A中显示出来的可见片段，e)多边形被B裁剪，f)最终显示出来的所有可见片段

692

下一步，执行WA_subdivide处理$B_{out}A$，新的*polyList*中仅含有$B_{out}A$，$C_{out}A$和$D_{out}A$。$B_{out}A$的源多边形B用于裁剪这些多边形，生成一个内部列表$B_{out}A$和$C_{out}A_{in}B$，同时生成一个外部列表，含有$C_{out}A_{out}B$和$D_{out}A$，如图15-48e所示。$C_{out}A_{in}B$位于B之后，被删除。内部列表中剩下$B_{out}A$，紧接着将其显示出来。在WA_subdivide返回之前将多边形列表设成新的外部列表。然后，调用WA_subdivide处理和显示$C_{out}A_{out}B$和$D_{out}A$，每个调用一次。完全的显示片段见图15-48f。

15.7.3 子像素区域细分算法

同所有的对象精度算法一样，Weiler-Atherton算法要求将每一个多边形与其他的所有多边形进行比较。15.2.5节的空间细分方法将屏幕划分成区域（或将环境分割成体），区域中的对象被单独进行处理[WEIL77]，从而可以减少比较的次数。即便如此，生成的多边形最终还是要绘制出来，这就产生了一个反走样的问题。但如果在子像素级别完成空间细分则它也可用于实现反走样。

1. Catmull的对象精度反走样算法

Catmull[CATM78b]提出了一种精确但很复杂的扫描线算法，他在每一个像素用对象精度的未加权区域采样实现反走样，所用的算法同Weiler-Atherton算法类似。算法的基本思想是对每一个像素执行可见面算法，即只比较投影到每一个像素的多边形片段。Catmull首先用3.14.1节中的Sutherland-Hodgman算法裁剪与扫描线交于像素点的多边形，如图15-49a所示，可以求出多边形投影到每一像素点的片段，再对它们用像素网格做空间划分。然后运用类似于Weiler-Atherton算法但更适于简单多边形几何的算法对每一个像素操作以判定每一片段覆盖该像素的量（图15-49b）。于是可以计算出可见部分颜色的加权和，作为每一像素点的颜色。因此，每一像素点的明暗度由投影到它的多边形片段的盒式过滤确定。

2. A缓存算法

对每一像素采用全对象精度的可见面算法代价太大！Carpenter的A缓存算法[CARP84]解决了这一问题。他的方法是用子像素网格上执行的图像精度操作逼近Catmull的每像素对象精度区域采样。这是对盒式过滤的区域采样的离散逼近。多边形首先按扫描线的顺序处理，把它们裁剪成一个个方形的像素覆盖块。对每一个像素生成一个裁剪多边形片段的列表。每一个片段都对应一个4 × 8位的掩码，记录它在像素中覆盖的部分（图15-49c）。片段的位掩码通过求每个片段边的掩码的异或而得。片段边的掩码确定原则是：边上为1，且边穿过的各行位于边右边的部分也为1（图15-49d）。所有与像素相交的多边形都处理完后，将像素中的片段按深度排序，并 693
用较近片段的位掩码裁剪较远片段，可确定出像素中的可见片段部分。然后对这些可见部分按其面积的大小求颜色的加权平均。位掩码用布尔量操作较为有效。例如，两个片段的位掩码的“与”操作能够判定它们之间的重叠部分。A缓存算法只为每个片段保留少量的附加信息。例如，它包含片段的z的范围，但并不保存片段哪一部分与这些z值相关。于是，当片段位掩码在z上重叠时，算法需要对子像素的几何形态做一个假定。这会产生一定的不精确性，尤其是当多个面交于一个像素时。

3. 用预计算的卷积表实现更有效的过滤

到目前为止我们讨论的子像素区域细分算法都采用未加权区域采样；这样，片段的影响就限制在它们裁剪的单个像素中。如果希望进行更大范围的过滤处理，必须考虑片段附近像素上的过滤器的影响（图15-49e）。Abram、Westover和Whitted[ABRA85]讨论了如何将更好的过滤器合成到这些算法中，他们将每一可见片段按几何类型分成了不同的类别。对每一个与片段相关的像素而言（图15-49e中的像素*A*到*I*），片段的类别和它相对于该像素的位置用做一个查询表的索引。该查询表中存有经过预计算的过滤内核与不同位置的原型片段的卷积。选中的项与片段的光强值相乘，并加到该像素的累加器中。不能归入任何一类的片段可用更简单片段的和或差以及位掩码来近似表示。

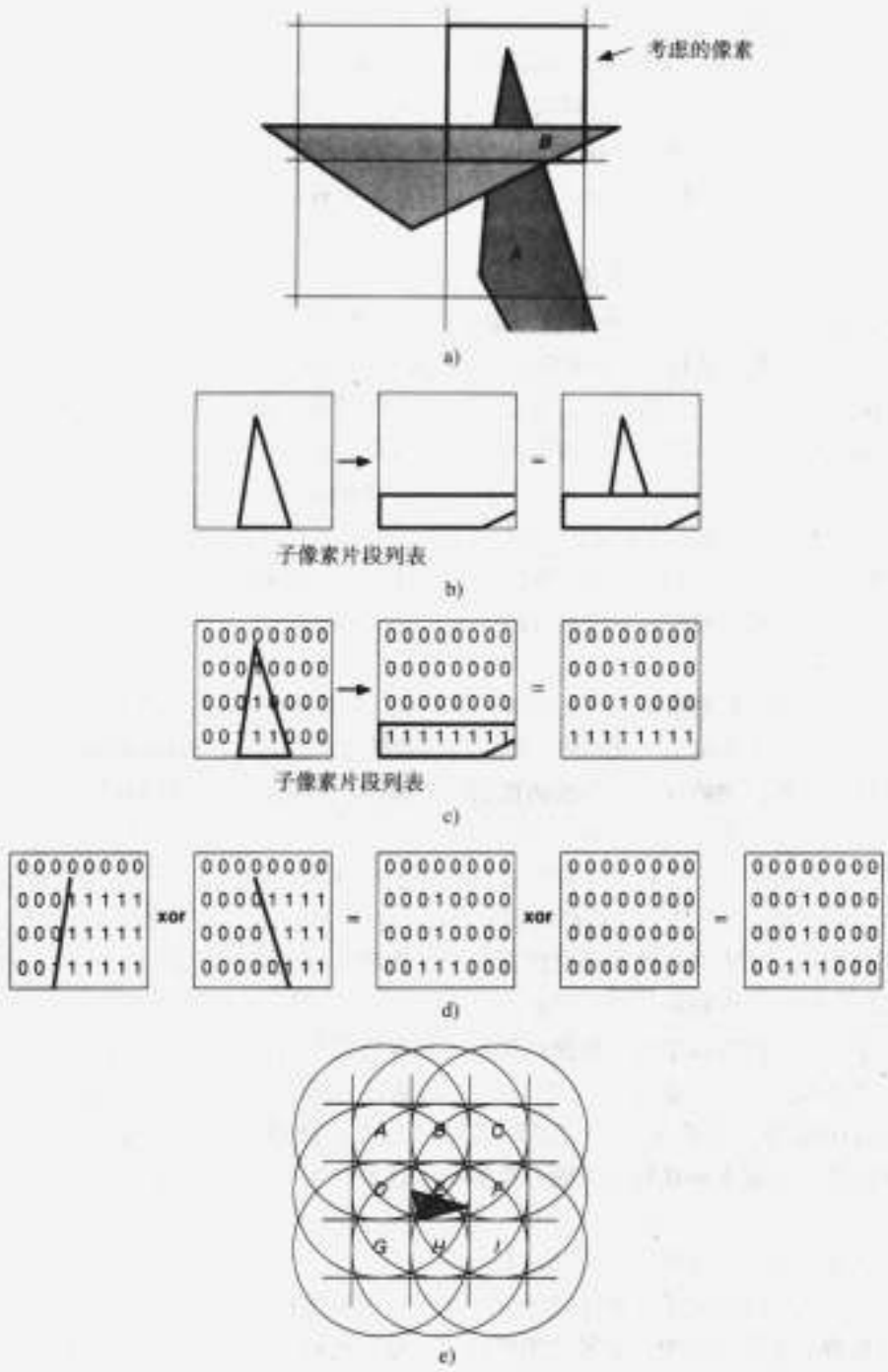


图15-49 子像素区域细分算法。a)样本像素内容，b)Catmull算法的子像素的几何形态，c)A缓存算法中子像素的几何形态，d)A缓存算法中片段的子像素掩码由边的掩码异或而得，e)Abram、Westover和Whitted算法增加多边形对所有受它影响的像素点的作用。(e图基于[ABRA85]。)

15.8 八叉树算法

显示八叉树编码的对象（见12.6.3节）的算法主要是利用八叉树中不相交立方体之间的规则

结构。因为八叉树算法已经进行了空间的预排序，所以现有的列表优先级算法很容易对平行投影给出正确的显示顺序[DOCT81；MEAG82a；GARG86]。从后向前排列时，排在前面的节点保证不会遮挡后面的节点。对正交投影而言，从后向前的正确排列仅由VPN便可确定。一种方法是先显示最远的八分体，然后是与它共享一个面的三个相邻八分体（顺序任意），然后显示最近八分体的三个相邻体（顺序任意），最后显示最近的八分体。在图15-50中，VPN为从0到V，相对于它的一个排列是0, 1, 2, 3, 4, 5, 6, 7。在这个排列中，任一节点都不会遮挡其后的节点。每一个八分体显示后，它的后继同样递归地按这个顺序显示。更进一步，因为每个叶节点都是一个立方体，最多只有三个面可见，它们的标志号也可由VPN确定。

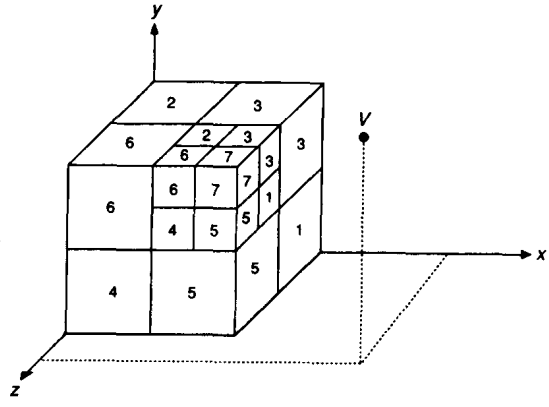


图15-50 从后向前显示的八叉树排列。（节点0位于左下后角落。）对由原点到V的VPN而言，节点可用几种不同的排序体系递归地显示出来

表15-1是8种不同的从后向前排序，由VPN三个坐标的符号和相应的可见八分体面确定。（注意每个排序中只有第一和最后一个八分体是固定的。）VPN的正的或负的x坐标分别表示右边（R）或左边（L）的面是可见的。类似地，y坐标对应着上（U）或下（D）面的可见性，z坐标对应着前（F）或后（B）面的可见性。VPN的任一坐标为0时，与其相关的面都是不可见的。只有非0坐标才对排序起作用。八叉树中所有节点的方向是一致的，节点的可见面和它们相应的多边形投影只须判别一次。处理任意的平行投影时可以用DOP代替VPN。

表15-1 从后向前排列与可见面

VPN				
z	y	x	由后向前顺序	可见面①
−	−	−	7,6,5,3,4,2,1,0	B,D,L
−	−	+	6,7,4,2,5,3,0,1	B,D,R
−	+	−	5,4,7,1,6,0,3,2	B,U,L
−	+	+	4,5,6,0,7,1,2,3	B,U,R
+	−	−	3,2,1,7,0,6,5,4	F,D,L
+	−	+	2,3,0,6,1,7,4,5	F,D,R
+	+	−	1,0,3,5,2,4,7,6	F,U,L
+	+	+	0,1,2,4,3,5,6,7	F,U,R

① R = right, L = left; U = up; D = down; F = front; B = back

另一种适于正交投影的从后向前排列的方法是对垂直于某一坐标轴的一个个面片依次处理，每一面片的内部再按行或列的顺序处理。VPN向量的每一分量的符号决定相应八叉树轴的处理方向。正的分量表示沿轴的增大方向，负的分量表示减小方向。依什么顺序使用这些轴是无关紧要的。例如，在图15-50中，分量全为正的VPN对应的一个排列是0, 4, 2, 6, 1, 5, 3, 7。首先是z变化，然后是y，然后是x。这一方法很容易推广到对体元（voxel）数组的操作[FRIE85]。

694 ~ 696

不需要显示一个对象的所有的体元，因为那些完全被其他体元包围的的体元是不可见的。因此，更有效的扫描转换算法是只绘制八叉树边界上的体元[GARG86]。边界体元集合的判定算法已在12.6.3节中讨论过。还可以做进一步的改进，因为即使是只显示边界体元，有一些面仍会画出后就被覆盖掉。Gargantini[GARG86]利用边界提取过程中获得的信息来确定每一个与

其他体元有相邻面的体元。这些面会被遮挡住，因此不需要画出来。绘制体元时，不一定要将其画成小的立方体，也可用垂直的矩形来近似表示每个体素（在某些限制情况下是一个像素）。

Meagher[MEAG82b]提出一种从前向后的算法，与前述的从后向前的算法顺序相反。他将要绘制的图像表示成初始为空的四叉树。每一个满的或部分满的八叉树节点按从前向后的顺序处理，并同与它的投影相交的四叉树节点相比较。那些投影只与满四叉树节点相交的八叉树节点是不可见的；这些节点和它们的后继节点都不再进一步处理。如果一个部分满的八叉树节点的投影与一个或多个部分满的四叉树节点相交，那么该八叉树节点的子节点再与四叉树节点的子节点比较。如果一个满八叉树节点的投影与部分满的四叉树节点相交，那么只有这些部分满的四叉树节点需要进一步细分以判定被投影覆盖的空节点。被一个满八叉树节点的投影包含的空四叉树节点被八叉树节点的值遮蔽。

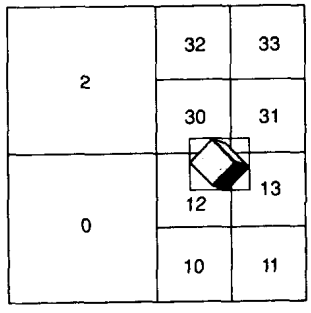


图15-51 从前向后八叉树扫描转换。每一个八叉树节点的投影和它的矩形框与四个四叉树节点相比较，这里是12, 13, 30和31

如图15-51所示，Meagher用正的矩形框包围每一个八叉树的投影。每一个框只和四个最低级别的四叉树节点进行比较，这些节点的边的大小不小于框的最大尺寸。它们是包含框的左下角的四叉树节点以及三个分别在北、东和东北方向的四叉树节点。在图15-51中，这四个节点是12, 13, 30和31。如果矩形框同一个矩形的四叉树节点相交，那么可以很容易地判定八叉树节点的投影（凸多边形）是否与四叉树节点相交[MEAG82b]。同列表优先级从后向前算法比较，这个从前向后算法是在图像精度上的操作，因为它依赖于一个图像精度的、在图像平面上投影的四叉树表示。

697

15.9 曲面算法

到目前为止，我们所讨论的算法，除了z缓存算法外，都只适用于多边形面定义的对象。第11章中的曲面对象只有用许多小的面片逼近处理后，才能使用适于多边形的算法。尽管利用多边形可以逼近曲面，但我们常常更希望直接用扫描转换去处理曲面，一方面可以消除多边形带来的视觉上的缺陷，另一方面可以避免逼近带来的额外存储开销。

11.4节中讨论的二次曲面是计算机图形学中常用的形式。二次曲面的可见面算法由Weiss[WEIS66]、Woon[WOON71]、Mahl[MAHL72]、Levin[LEVI76]和Sarraga[SARR83]开发。他们都找到了两个二次曲面求交的方法，给出了一个x,y,z的四阶方程，其解可通过数值方法求得。Levin用参数化相交曲线的方法将问题简化为二阶。球面是一种特殊的二次曲面，处理起来相对容易些，因而更多地得到应用。分子就常常显示成许多有色球体的集合（见彩图II-19）。人们开发了许多分子显示算法[KNOW77；STAU78；MAX79；PORT79；FRAN81；MAX84]。在15.10节中将讨论如何用光线跟踪绘制球面。

更具有灵活性的是参数样条曲面（第11章），因为它们更一般且在曲面片的边界处允许一阶导数连续。Catmull[CATM74b；CATM75]提出了第一个双三次曲面的显示算法。与Warnock算法的基本思想一致，对曲面片在s和t方向递归细分，生成四个小曲面片，直至投影覆盖不超过一个像素。用z缓存算法判定曲面片在该像素处是否可见。若是，则求出它的明暗并放入帧缓存中。算法的伪代码见图15-52。由于确定曲面片自身的大小十分费时，可以考虑用由面片

四个角顶点定义的四边形来代替操作。为进一步提高效率，可将曲面片（或它的包围盒）与裁剪窗口进行比较，若它完全位于窗口中，则从它生成的面片不需要裁剪的比较；若完全位于窗口外，则可将其丢弃；最后，若部分可见时，由它生成的每一面片都需要进行检测。

```
for（每个面片）{
    将面片压入栈;

    while（栈非空）{
        从栈中弹出面片;

        if（面片覆盖的像素个数≤1）{
            if（曲面片的像素在z方向上更近）
                确定明暗并画出
        } else {
            细分面片为4个子面片;
            将子面片压入栈;
        }
    }
}
```

图15-52 Catmull递归细分算法的伪代码

此后，Blinn和Whitted[LANE80b]各提出一个双三次曲面的扫描线算法，从一条扫描线到另一扫描线跟踪曲面的可见边。边的定义可以是真实的面片边界，也可以是轮廓边（如图15-53所示）。在轮廓边处，曲面法向量在三维屏幕坐标系统中的z分量为0。

Blinn直接处理曲面片的参数表达式。对扫描线$y=\alpha$，他发现，所有的s和t值都满足方程：

$$y(s,t)-\alpha=0 \tag{15-8}$$

这些s和t值用于计算$x(s,t)$和$z(s,t)$。不幸的是，方程(15-8)没有最终形式的解，而只能用Newton-Raphson迭代法（见附录）求解。求根算法需要一个初始值，可以用前一条扫描线的解作为当前扫描线的初始值。也会有无根的情况，算法便失败。类似地，Whitted也采用数值方法加上对（x,z）平面中曲线的逼近方法，其中曲线是$y=\alpha$平面与双三次曲面片的交。Whitted算法不能正确地处理某些轮廓边，能稳定地检测轮廓边的算法见[SCHW82]。

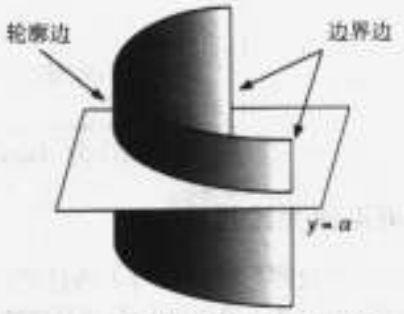


图15-53 面片的可见边的两种定义：由边界边定义和由轮廓边定义

698
↓
699

还有一种基于双三次曲面片自适应细分的方法非常成功，它确定了一个认为面足够平的容限，当曲面满足这个值时，划分中止。这个容限取决于显示设备的分辨率和被分区域相对于投影平面的方向。这就大大减少了不必要的细分。曲面片在一个方向足够平时，就只需要在另一个方向进行细分。细分程度足够细之后，曲面片可看成是四边形。用每个面片的四个角定义一个小的多边形区域，直接用扫描线算法处理，此时，允许多边形和双三次曲面很自然地混合在一起。

使用这一基本思想的算法分别由Lane和Carpenter[LANE80b]以及Clark[CLAR79]提出。他们的算法的差别在于曲面片的细分条件与判断曲面是否足够平的测试函数。Lane-Carpenter算法仅当处理的扫描线开始与曲面片相交时进行细分，而Clark算法在预处理阶段细分。Lane-Carpenter曲面细分算法在11.3.5节中讨论，伪代码见图15-54。

面片的控制点定义了它的凸包。将面片加入到活动面片表中等待处理，此时扫描线的y值是它的控制点的最小y值。这就节省了大量的内存。平整测试必须确定曲面片是否足够平以及边界曲线是否足够线性。如果同一个面片生成了两个面片，一个已足够平，而另一个相邻面还要继续细分下去，这时，会在面片上产生裂缝。两个面片之间的共享边在细分后，对第一个面片而言是一条线，而对第二个面片的子面片而言，是对曲线的分段线性逼近。解决这个问题的方法有两种：一是改变平整测试的容限，以使面片分得更细；二是采用Clark的方法，一旦确

定面片足够平，便将边当成一条直线来细分。

```
将面片加入面片表;
初始化活动面片表;

for (每条扫描线) {

    更新活动面片表;

    for (活动面片表中的每个面片) {
        if (面片可以用平面四边形逼近)
            将面片加入到多边形表;
        else {
            将面片分成子面片;
            for (对每个新的子面片) {
                if (子面片与扫描线相交)
                    加入到活动面片表中;
                else
                    加入到面片表中;
            }
        }
    }

    对当前扫描线处理多边形表;
}
```

图15-54 Lane-Carpenter算法的伪代码

15.10 可见面光线跟踪

光线跟踪（也称为光线投射）通过跟踪从观察者眼睛到场景中物体的想像中的光线来确定面的可见性[⊖]。这是一种典型的图像精度算法（本章开始时讨论过）。选定一个投影中心（观察者的眼睛）和给定视图平面上的窗口，该窗口被划分为规则网格，网格的每一元素对应于要求分辨率下的像素。于是，对窗口中的每一像素都有一条视线从投影中心出发，穿过像素中心到达场景中，如图15-55所示。像素的颜色设成最近交点处对象的颜色。简单光线跟踪的伪代码见图15-56。

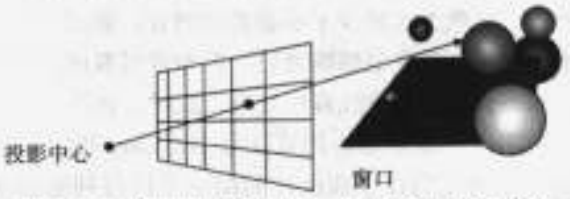


图15-55 光线从投影中心发射经过窗口映射到每一个像素，判定最近的相交对象

光线跟踪算法最早由Appel[APPE68]以及Goldstein与Nagel[MAGI68；GOLD71]提出。Appel用光线的一个稀疏网格判定场景明暗度，包括确定一点是否在阴影中。Goldstein与Nagel最初用他们的算法模拟弹道发射与核粒子的轨迹，后来才应用于图形学中。Appel最早研究光线跟踪计算阴影，而Goldstein与Nagel最先用光线跟踪求布尔集合运算。Whitted[WHIT80]与Kay[KAY79a]将光线跟踪算法扩展到计算高光反射与折射。我们将在16.12节中讨论阴影、反射与折射等人们所熟知的光线跟踪适用的领域，我们还将给出一个合成了可见面判定与明暗处理的完整的光线跟踪递归算法。在本节，我们只讨论光线跟踪

700 / 701

⊖ 尽管光线投射与光线跟踪常被混同使用，但有时光线投射用于专指本节的可见面算法，而光线跟踪指的是16.12节中的递归算法。

在可见面判定方面的应用。

```
在视图平面上选择投影中心和窗口;
for (图像中的每条扫描线) {
    for (扫描线上的每个像素) {
        确定从投影中心穿过像素的线;
        for (场景中的每个对象) {
            if (对象被交并且最近)
                记录交线和对象名;
        }
        设置像素的颜色为最近对象交线的颜色;
    }
}
```

图15-56 简单光线跟踪的伪代码

15.10.1 相交计算

光线跟踪算法的核心是计算光线与场景中对象的交点。为实现这一点，这里采用与第3章中相同的向量的参数表示。从 (x_0, y_0, z_0) 到 (x_1, y_1, z_1) 的光线上的每一点 (x, y, z) 由参数t定义如下：

$$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0), \quad z = z_0 + t(z_1 - z_0) \tag{15-9}$$

为方便起见，我们定义Δx，Δy与Δz为

$$\Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0, \quad \Delta z = z_1 - z_0 \tag{15-10}$$

因此，

$$x = x_0 + t\,\Delta x, \quad y = y_0 + t\,\Delta y, \quad z = z_0 + t\,\Delta z \tag{15-11}$$

如果(x_0, y_0, z_0)为投影中心，(x_1, y_1, z_1)为窗口中像素的中心，则t从0变化到1代表这两点之间的点；t为负值时代表投影中心之后的点，而t大于1时对应于窗口远离投影中心一边的点。对每一类对象我们需要找到一种表示方式，以方便地求出交点处的t值。最简单的对象是球体，这就是为什么在典型的光线跟踪图像中存在大量球体的原因！以(a, b, c)为球心，半径为r的球体用如下方程表示：

$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2 \tag{15-12}$$

702

展开式(15-12)，并将式（15-11）中的x,y,z值代入，可得：

$$x^2 - 2ax + a^2 + y^2 - 2by + b^2 + z^2 - 2cz + c^2 = r^2 \tag{15-13}$$

$$(x_0 + t\Delta x)^2 - 2a(x_0 + t\Delta x) + a^2 + (y_0 + t\Delta y)^2 - 2b(y_0 + t\Delta y) + b^2 + (z_0 + t\Delta z)^2 - 2c(z_0 + t\Delta z) + c^2 = r^2 \tag{15-14}$$

$$x_0^2 + 2x_0\Delta xt + \Delta x^2t^2 - 2ax_0 - 2a\Delta xt + a^2 + y_0^2 + 2y_0\,\Delta yt + \Delta y^2t^2 - 2by_0 - 2b\Delta yt + b^2 + z_0^2 + 2z_0\,\Delta zt + \Delta z^2t^2 - 2cz_0 - 2c\Delta zt + c^2 = r^2 \tag{15-15}$$

多次合并后可得：

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] + (x_0^2 - 2ax_0 + a^2 + y_0^2 - 2by_0 + b^2 + z_0^2 - 2cz_0 + c^2) - r^2 = 0 \tag{15-16}$$

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x\,(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] + (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0 \tag{15-17}$$

式(15-17)是关于t的二次方程，其系数全是可从球与光线方程中导出的常量。因此可用二次方程公式求解。如果没有实根，那么光线与球不相交；如果有一个实根，则光线与球相切于一点；如果有两个根，则相交于两点，其中得到最小正t值的是最近点。将光线规格化使得从(x_0, y_0, z_0)到(x_1, y_1, z_1)之间的距离为1将可简化计算。这样其t值度量的将是WC单位表示的距离。同时，相交运算亦得到了简化，因为式(15-17)中t^2的系数为1。我们可以用类似的方法求得光线与第11章中介绍的普通二次曲面的交点。

正如在第16章中我们将会看到的，必须确定交点处曲面的法向量以对曲面进行明暗处理。这一操作对球体而言非常简单，因为球面上未规格化的法向量是从球心指向交点的向量：设球心为(a, b, c)，则交点(x, y, z)处的表面法向量为$((x-a)/r, (y-b)/r, (z-c)/r)$。

求解光线与多边形的交点相对比较困难。一般的方法是先确定光线是否与多边形所在平面相交，然后判断交点是否位于多边形内部。平面的方程可表示为

$$Ax + By + Cz + D = 0 \tag{15-18}$$

代入式(15-11)，得

$$A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0 \tag{15-19}$$

$$t(A\Delta x + B\Delta y + C\Delta z) + (Ax_0 + By_0 + Cz_0 + D) = 0 \tag{15-20}$$

$$t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{(A\Delta x + B\Delta y + C\Delta z)} \tag{15-21}$$

703

如果式(15-21)中分母为0，那么光线与平面平行没有交点。一种判定交点是否位于多边形内部的简单方法是将多边形与交点正投影到定义坐标系的三个平面之一（如图15-57所示）。要获得最精确的结果，应该选取有最大投影的方向为轴，这对应于多边形的平面方程中系数绝对值最大的坐标。对多边形顶点和该点的坐标做正投影。于是，该点的多边形包含测试就可完全在二维中完成，算法参见7.12.2节。

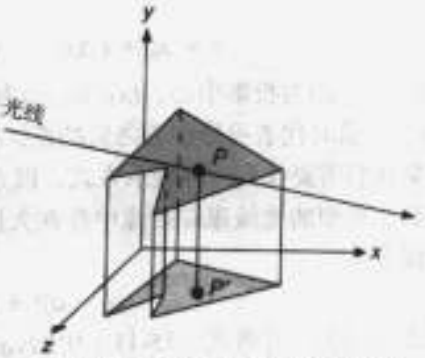


图15-57 光线与多边形是否相交的判定。光线与多边形平面的交点p被投影到其中一个坐标系平面上，再判断投影点p'是否位于投影多边形内部

同z缓存算法类似的是，光线跟踪也只在投影线和对象之间求交，而不需要直接判断场景中两个对象之间的交。z缓存算法将对象近似成与对象相交的投影线上的一系列z值；而光线跟踪算法将对象近似成与场景相交的投影线上的一系列的交点。对于一类新的对象，我们只需要为其写一个扫描转换和计算z值的过程便可实现z缓存算法的扩充；同样，我们只需要写一个光线相交的过程便可实现可见面光线跟踪算法中新对象的扩充。在两种情况下，都必须有一个计算曲面法向量的过程，用于进行明暗处理。相交和曲面法向量算法有很多，如为代数曲面[HANR83]、参数曲面[KAJI82；SEDE84；TOTH85；JOY86]和第20章中讨论的许多对象开发的算法。这些算法的概述参见[HAIN89；HANR89]。

15.10.2 可见面光线跟踪算法的效率

在每一像素，z缓存算法只计算投影到该像素的对象的信息，同时利用相关性。比较而言，我们讨论的可见面光线跟踪算法简单而运算量大，需要对从眼睛发出的每一条光线与场景中的每个对象求交。一幅含100个对象的1024 × 1024的图像就需要100M次的相交计算。无怪乎

704

Whitted发现75%到95%的系统时间都花在了求交计算上[WHIT80]。因此，我们有必要讨论一下，如何能够加速每一步的相交计算或尽可能地省略掉。正如我们将在16.12节中看到的，递归光线跟踪算法跟踪来自交点的附加光线以判定像素的明暗度。因此，15.2节中的许多技术，如透视投影和背面消除等，并不总是有用的，因为并非所有的光线都从同一个投影中心发出。在16.12节中，我们还将增加一些专门用于处理这些递归光线的技术。

1. 优化相交计算

在对象-光线求交的方程中有许多项中包含表达式，这些表达式要么对整幅图像是常量，要么对某一特定光线是常量。这些表达式可以预先计算出来，如多边形到平面的正投影等。经过认真的比较和数学上的研究，我们发现有可能提出快速的求交计算；即使是15.10.1节中的最简单的球体求交公式也有改进的余地[HAIN89]。如果光线变换到与z轴同方向时，相同变换也可作用到每个候选对象上，这样，任何的相交都将发生在$x = y = 0$处。这就简化了求交计算并允许最近的对象通过z排序得到。在明暗处理计算时可通过反向变换将交点再变换回来。

包围体提供了另一种加速求交运算的途径。一个求交相对复杂的对象，对其包围体求交就要相对简单得多，这些包围体可以是球体[WHIT80]、椭球体[BOUV85]或长方体[RUBI80; TOTH85]。光线不能与包围体相交时，包围体中的对象就不需要进行测试。

Kay和Kajiya[KAY86]提出一种凸多面体的包围体，由一组无限的*面对*（slab）的交组成，每一个面对由一对包围该对象的平行平面定义。图15-58a是二维的情况，对象被四个面对（平行线对）以及它们的交所包围。这样，每一个面对都可由方程(15-18)表示，其中A，B和C是常量，D是D_{min}或D_{max}。如果用相同的参数化面对集合包围所有对象，每一个包围都可简化成用每个面对的D_{min}和D_{max}代表。光线一次和一个面对求交。与每一个面对平面的交可由方程（15-21）求出，产生远和近两个t值。对所有的包围采用相同的参数化面对集合，可实现方程（15-21）的简化，产生

$$t = (S + D)T \tag{15-22}$$

其中，$S = Ax_0 + By_0 + Cz_0$，$T = -1/(A\Delta x + B\Delta y + C\Delta z)$。对给定的光线和参数化面对，$S$和$T$都只计算一次。

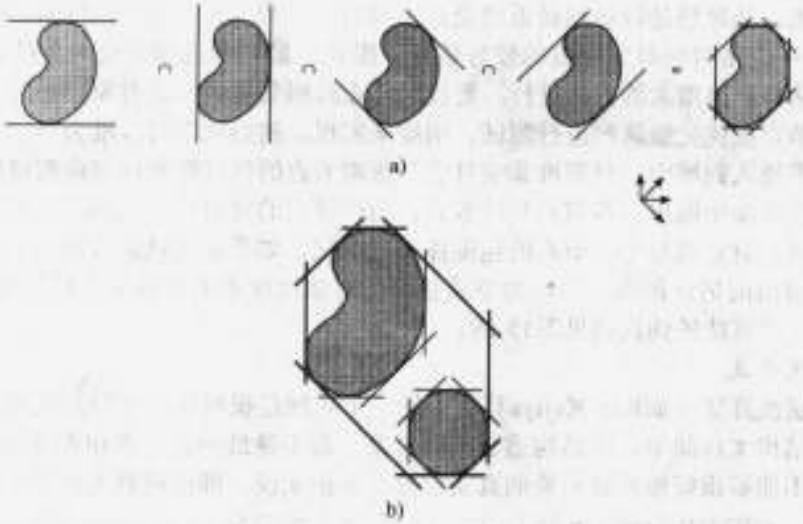


图15-58 面对的交构成的包围。a)由固定集合的参数化面对包围的对象，b)两个包围体的包围体

既然每一个包围都是面对的交，光线与整个包围的交就等于光线与每个面对的交的交集。可通过取t的最大近值与t的最小远值实现。为了快速地检测不相交的情况，某一包围的t的最大近值和最小远值在处理每一面对时可修改，当前者超过后者时对包围的处理便终止。

2. 减少相交计算

理想情况下，每一条光线只需要与它实际相交的对象做相交测试。更进一步，在许多情况下，我们希望每条光线只和交点离光线的原点最近的对象做测试。为实现这一目标，有许多技术采用预处理场景的方法将光线和对象分成同等的类，以限制相交测试的次数。这些技术包括在15.2节中已有介绍的两个补充方法：层次结构和空间划分。

3. 层次结构

尽管包围体本身并不能确定次序或相交测试的频率，包围体可以组织成嵌套的层次结构，对象位于叶节点与内部节点，每个节点又包围它的子节点[RUBI80；WEGH84；KAY86]。例如，一个针对Kay-Kajiya包围体集合的包围体可以这样计算：用每一个子包围体的最小值D_{min}和最大值D_{max}构造每一对平面，如图15-58b所示。

如果包围体的父节点不与光线相交，那么该包围体也不会与光线相交。根据这一点，当相交测试从根开始时，许多级别的分支（含有许多对象）都将被简单地拒绝。层次结构遍历的算法可简单表示如下：

705
~
706

```
void HIER_traverse (ray r, node n)
{
    if (r与n的包围体相交)
        if (n是一个节叶点)
            r与n的对象相交;
        else
            for (n的每个孩子节点c)
                HIER_traverse (r, c);
}  /* HIER_traverse */
```

4. 有效层次结构遍历

HIER_traverse按深度优先遍历层次结构。Kay和Kajiya[KAY86]提出了一种有效的遍历包围体层次的方法，其思想是以找到最近的交点为目的。光线与Kay-Kajiya包围体的交对应着两个t值，较小的一个是对到对象距离的较好估计。因此，最好的选择求交对象的顺序是从光线原点出发，沿估计距离增大的方向进行。要找到与光线相交的最近的对象，必须维护一个节点的列表，这些节点按优先级队列进行测试，用堆来实现。初始状态时，堆为空。若根包围体与光线相交，则根插入到堆中。只要堆非空且它的顶端节点的估计距离比当前测试的最近的对象要近，就将节点从堆中取出。若节点是叶节点，则计算它的对象与光线的交。否则，它就是一个包围体，此时，对它的每个子节点的包围体进行测试，如果相交就插入堆中，同时标记上包围相交计算时得出的估计距离。当堆为空或相交的对象比堆中的所有节点的估计距离都近时，选择过程终止。该算法的伪代码见图15-59。

5. 自动层次生成

包围体的层次算法（如Kay-Kajiya算法）的一个问题是很难生成好的层次结构。按建模过程生成的层次结构太过简单，该结构适于控制对象，而不是最小化对象和光线的交。另外，模型层次结构并不能够很好地反映对象的真实位置。举例来说，即使机器人两只手的手指相互接触，它们在层次结构上仍分属于相隔很远的不同层次上的对象。Goldsmith和Salmon[GOLD87]提出一种自动生成适于光线跟踪的好的层次结构方法，其基本思想是通过估计与光线相交的运算量来衡量层次结构的质量。

```
void KayKajiya (void)
{
    object *p = NULL;          /* 指向最近命中对象的指针 */
    double t = ∞;              /* 到最近命中对象的距离 */

    预先计算光线交;
    if ( 光线命中根的包围体 ) {
        将根插入堆中;
    }
    while ( 堆不为空且到顶端节点的距离<t ) {
        node *c = 从堆中删除顶端节点;

        if ( c是叶子节点 ) {
            光线与c的对象相交;
            if ( 光线命中它并且距离<t ) {
                t = 距离;
                p = 对象;
            }
        } else {       /* c是一个包围体 */
            for ( c的每个子节点 ) {
                光线与子节点的包围交;
                if ( 光线命中子节点的包围体 )
                    将子节点插入堆中;
            }
        }
    }
}  /* KayKajiya */
```

图15-59 用Kay-Kajiya包围体找到与光线相交的最近对象的算法

我们可以考虑单个包围体的情形。假定每一个包围体在计算是否与光线相交上运算量是一样的，则运算量直接正比于包围体与光线相交的次数。包围体被视线命中的概率是命中该包围体的视线占所有视线的百分比。这一概率正比于包围体投影到视图平面上的面积。平均来看，对凸包围体而言，这个值大致正比于包围体的表面积。由于每一个包围体都包含在根包围体中，在光线与根相交的前提下还与第i个包围体相交的条件概率可用A_i/A_r近似表示，其中，A_i是第i个包围体的表面积，A_r是根包围体的表面积。

如果光线与某一个包围体相交，假定我们必须对该包围体的k个子节点都做相交计算。这样，该包围体的总运算量用相交次数来表示为kA_i/A_r。根的运算量就是它的子节点的个数（因为$A_i/A_r=1$），且叶节点的运算量为0（因为k=0）。要计算某一层次的运算量，可以对其每一个包围体的运算量求和。例如，在图15-60中，每一个节点都标出了它的表面积。假设根A与光线相交，运算量是一次求交，根的估计运算量是4（它的子节点的数目）。它的两个子节点（C和E）是叶节点，运算量为0。B有两个子节点，且表面积为5。因此，它的估计运算量是2（5/10）=1.0。D有三个子节点，且表面积为4，所以它的估计运算量为3(4/10) = 1.2。剩下的惟一非叶节点是F，有两个子节点，表面积为3，估计运算量为2(3/10) = 0.6。总的运算量估计为1（与根求交）+4+1.0+1.2+0.6=7.8次可能的相交。

707 ~ 708

因为这里我们更关心的是相对值的大小，所以并不需要除以根节点的表面积。更进一步看，我们并不需要知道包围体的实际表面积，而只需要与它成比例的一个值即可。例如，对直棱柱而言，可以不用$2lw+2lh+2wh$，而只需要使用重新组合简化后的形式：$(w+h)l+wh$。

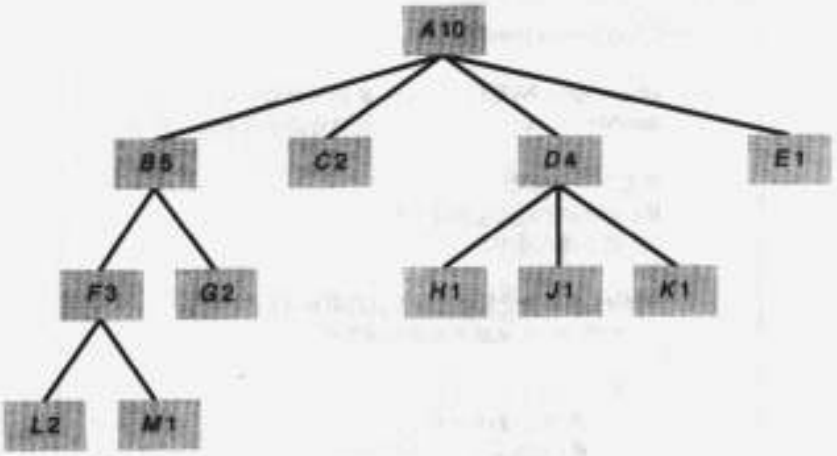


图15-60 层次结构的代价评估。字母为节点名字；数字是节点的表面积

Goldsmith和Salmon用递增的方法生成一个层次结构，即一次增加一个新节点。增加节点的顺序对算法有影响。也可以直接按照建模的顺序，但对大多数的场景而言，将节点打乱后用随机选取的顺序能够获得更好的结果。增加节点的方法是：使之成为现有节点的子节点或用一个包含初始节点和新节点的新的包围体节点来代替某一现有节点。在两种情况下，都不用从头计算生成新树的运算量，而只需确定增加一个节点所需的递增的代价。节点增加为子节点时，可能会增加其父节点的表面积，同时父节点的子节点个数也加1。于是，父节点的估计代价的差值为$k(A_{new}-A_{old})+A_{new}$，其中$A_{new}$和$A_{old}$分别是父节点新和旧的表面积，$k$是初始子节点的数目。如果节点增加是生成了一个新的父节点，以新节点和初始节点为子节点，那么新生成父节点的增加代价为$2A_{new}$。在两种情况下，新子节点的祖父节点和更上层的祖先节点增加的代价同样用$k(A_{new}-A_{old})$来计算，其中，k，A_{new}和A_{old}是祖先节点的值。这一方法的前提是假设节点放置的位置对根包围体的大小没有影响。

最直接了当的方法是对新节点在树中可能的位置都计算其增加的代价，然后选取使其代价
709 增加最小的一个位置。Goldsmith和Salmon采用的是启发式搜索的方法，从根开始计算其增加子节点的代价。然后选取一棵子树，选取的原则是使添加新的子节点后包围体表面积增加最小。接着算法继续对该子树操作，计算对其增加一个子节点的代价，然后根据表面积增加最小的原则继续选取它的子树。到达叶节点时，计算包含初始叶节点和新节点的新包围体节点的生成代价。搜索结束时，节点插入点的位置便是增加运算量最小的位置。因为对单个节点确定插入点的搜索代价为$O(\log n)$，所以整个层次结构生成的时间可表示为$O(n\log n)$。搜索与评估过程是基于启发式的，因此生成的层次结构并不是最优的，但这样生成的层次结构能大大地减少相交计算的代价。

6. 空间划分

包围体层次结构将对象自底向上地组织在一起；而空间划分方法则是将空间进行自顶向下的划分。首先计算场景的包围盒。有一种方法是将包围盒分成等距离的规则网格（如图15-61所示）。每一个分区都对应一个对象列表，这些对象要么完全位于该分区中，要么仅有一部分在里面。填充列表时将每个对象归入包含它的一个或多个分区中。如图15-62所示的2D情况中，光线只与它穿过的分区中的对象相交。另外，分区按照光线穿过的顺序进行处理，这样，只要找到一个有相交的分区，就不需要再处理剩余的分区。要注意的是，我们必须考虑该分区中所有其他的对象，

以确定哪个交点最近。由于分区是均匀网格，沿光线方向的每一个后续分区的计算可以采用3.2.2节中介绍的三维画线算法。略加修改以生成光线经过分区的列表[FUJI85；AMAN87]。

如果光线与某分区中的对象相交，还需要判断交点是否位于该分区中；有可能交点位于很远的另一个分区中，而另一个对象则有更近的一个交点。例如，在图15-63中，对象*B*尽管位于分区2中，却相交于分区3。必须继续遍历各个分区直到在当前分区中找到一个交点，在此例中是分区3中的*A*。为了避免光线与对象在多个分区中重复求交，可以在首次处理对象时将交点和光线的ID随对象缓存起来。

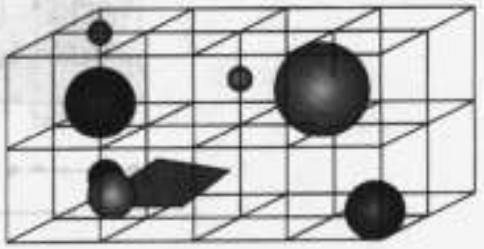


图15-61 场景划分成大小相等的均匀网格

Dippé与Swensen[DIPP84]研究了自适应划分算法，生成大小不等的分区。另一种自适应空间划分方法利用八叉树来划分场景[GLAS84]。算法中用12.6.3节中描述的八叉树寻邻区算法来确定沿光线的下一个分区[SAME89b]。八叉树和其他的层次空间划分方法都可以看成是层次划分的特例，在这种层次结构中，任一节点的子节点之间互不相交。因为这些方法允许采用自适应划分，划分策略的选择对分区中对象的数量和对象相交的运算量很敏感。这在不规范、不均匀分布的场景的处理中非常有利。

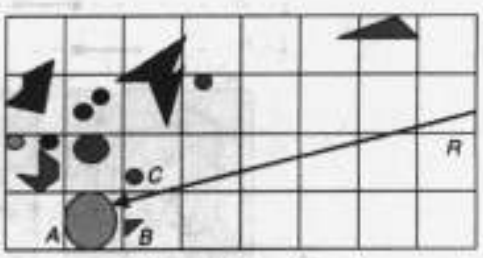


图15-62 空间划分。光线*R*只可能与*A*，*B*和*C*相交，因为它穿过的其他分区都是空的

空间划分与层次结构各有优点，可以将它们结合在一起。Snyder和Barr[SNYD87]采用手工设定的层次结构，其内部节点要么是列表，要么是规则的三维网格。这就允许人为设计环境，对分布稀疏的少量对象采用列表，对大量的规则分布的对象采用网格方式。彩图III-1是一个用Snyder与Barr系统实现光线跟踪的场景，含2×10^9次基本运算。

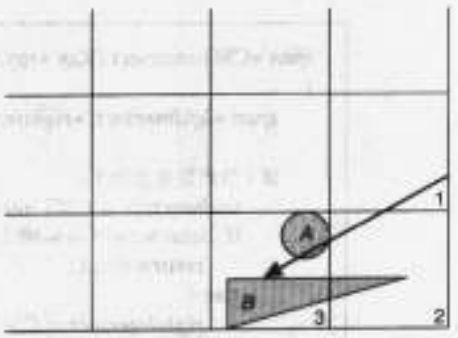


图15-63 对象可能在另一个体元中相交

710
↕
711

15.10.3 计算布尔集合运算

Goldstein和Nagel[GOLD71]最早开始研究对布尔集合运算生成的简单对象进行光线跟踪。用第12章中介绍的方法直接对实体间进行比较，从而确定两个3D实体的并、差或交是很困难的。相反，光线跟踪允许将三维问题简化为一维计算。每条光线与基本对象的交能得出一系列的t值，每一个t都对应一根光线进入或穿出对象的点。因此，每个t值也定义了一个跨段的起始点，而在跨段内光线或进入对象或离开对象。（当然，光线与对象相切，仅有一个交点的情况也必须考虑。）布尔集合运算一次处理一条光线，在一维上判断沿同一光线而分属两个对象的跨段之间的并、差或交。图15-64是光线穿过两个对象时生成的跨段，以及执行集合运算后所得的结果。对每条光线遍历CSG层次结构，在每个节点计算左相交列表和右相交列表，代码见图15-65。彩图III-2是用CSG定义的光线跟踪绘制的碗。

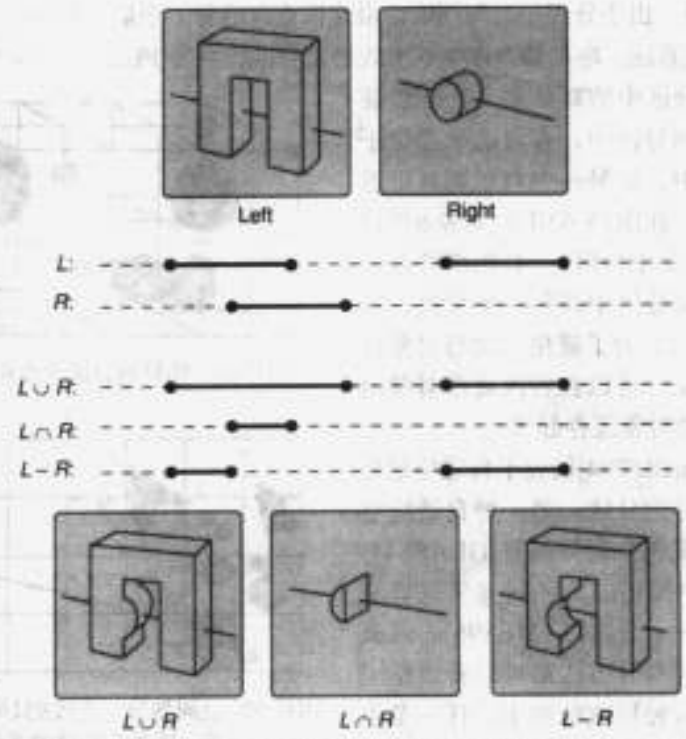


图15-64 合成光线-对象相交所得的跨度列表。(经许可摘自[ROTH82]。)

```
span *CSG_intersect (Ray *ray, CSG_node *node)
{
    span *leftIntersect, *rightIntersect;    /* 跨度列表 */

    if (节点是复合的) {
        leftIntersect = CSG_intersect (ray, node->leftChild);
        if (leftIntersect == NULL && node->op != UNION)
            return NULL;
        else {
            rightIntersect = CSG_intersect (ray, node->rightChild);
            return CSG_combine (node->op, leftIntersect, rightIntersect);
        }
    } else      /* 节点是图元 */
        return 对象与光线的交线;
}  /* CSG_intersect */
```

图15-65 计算光线与CSG层次结构相交的代码

表15-2 布尔集合运算合成的对象的点分类

左	右	∪	∩	−
in	in	in	in	out
in	out	in	out	in
out	in	in	out	out
out	out	out	out	out

Roth提出，在进行求差或求交运算时，如果与树的左边不相交，那么也没有必要同树的右边

求交[ROTH82]。只有并运算时其结果可为非空。事实上，如果我们只需要判定合成对象是否相交（而不是实际求相交的集合），那么如果左边相交且进行并运算时，右边就没必要计算。

CSG_combine函数处理两个相交记录的列表，每个列表都按t的升序排序，并将它们按不同的操作合并起来。列表的合并是通过删除具有下一个最大t值的相交记录完成的。删除了记录的列表中设置一个标志，用于记录光线是进入左列表还是右列表。至于该交点处起始的跨度是否位于合成对象中可通过查表得知，该表如表15-2所示，是基于操作符和两个“in”标志的组合。记录位于合成列表中的条件是仅当它是合成对象的一个跨度的开始或结尾，其他在跨度内部的情况都不考虑。当光线从对象内部发出时，标志必须正确地初始化。 712 ~ 713

15.10.4 反走样光线跟踪

我们讨论过的简单光线跟踪算法都采用规则网格上的点采样，所以会产生图像的走样现象。Whitted[WHIT80]提出一种自适应方法，对可能产生走样最严重的部分发出更多的检测光线。用这些附加的样本计算像素的更精确的值。他的*自适应超采样方法*向像素的边角而不是中心部分发射采样光线，如图15-66a和图15-66b所示。这样在开始时，图像只需要一个附加行和一个附加列的光线。光线发出穿过像素的四个角后，取平均求出其明暗值；如果像素的值和该平均值差别不大，就用该平均值作为像素值。如果差别很大，就将像素继续细分，方法是对像素边的中点和中心发射光线，形成四个子像素（图15-66c）。每个子像素的四个角处的光线按同样的原则进行比较。细分可以递归地进行下去，直到达到一个预先定义的最大划分深度值（如在Warnock算法中），或光线的明暗已足够相似。像素的明暗是它子像素明暗的面积加权平均。自适应超采样能够取得比未加权区域采样方法更好的显示效果，而不用增加提高采样率的花费。

我们来看图15-66a，光线穿过两个相邻像素的边角，最大的划分深度是2。在图15-66b中，如果不需要对光线A, B, D, E组成的像素作进一步细分，那么，像素的颜色可表示成 $(A+B+D+E)/4$，其中，用光线的名字代表它的明暗。相邻的那个像素则需要进一步细分，对其继续跟踪光线G, H, I, J和K，因此定义了图15-66c中的四个子像素的顶点。每一子像素递归地进行处理。这时，只有右下的子像素再次划分，跟踪的光线为L, M, N, O和P（图15-66d）。此时已达到最大划分深度，其像素的明暗确定为：

$$\frac{1}{4}\left[\frac{B+G+H+I}{4}+\frac{1}{4}\left[\frac{G+L+M+N}{4}+\frac{L+C+N+O}{4}+\frac{M+N+I+P}{4}+\frac{N+O+P+J}{4}\right]+\frac{H+I+E+K}{4}+\frac{I+J+K+F}{4}\right]$$

当穿过像素的光线错过一个小的对象时，也会产生走样问题。这会产生以下这样的情况：规则分布的对象中有些不可见的变成可见的了，或者在一个运动对象的图像序列中，它一会儿出现，一会儿消失，好象是被最近的光线一会儿射中一会儿错过。Whitted 用球形的包围体包围每一对象，包围体足够大，总是能够同至少能和一条从眼睛发出的光线相交，这种方法能够避免这些现象的发生。因为光线都会聚到眼睛，包围体的大小是到眼睛距离的函数。若光线与包围体相交却不与对象相交，则所有共享这一光线的像素都要进一步划分，直到与对象相交。16.12节中将讨论更多更新的反走样光线跟踪算法。 714

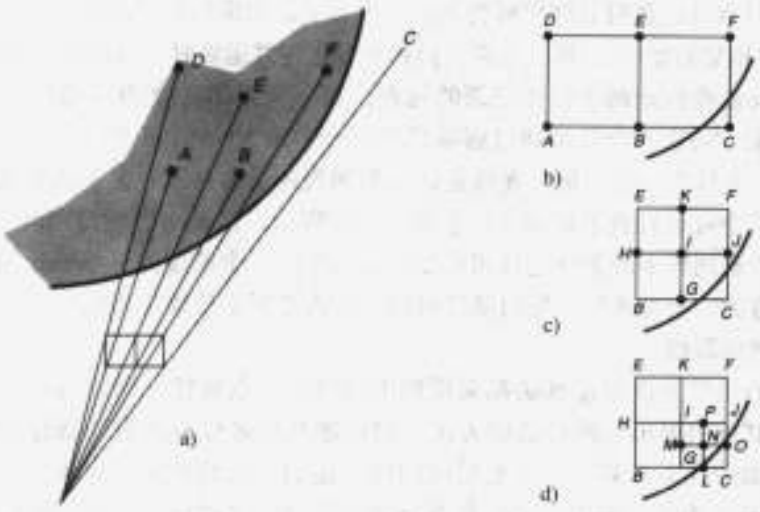


图15-66 自适应超采样。a)两个像素和穿过它们的边角的光线，b)左边的像素不用细分，c)细分右边的像素，d)右下的子像素被细分

15.11 小结

Sutherland、Sproull和Schumacker[SUTH74a]强调可见面判定的核心是排序。确实，我们在算法中看到了许多排序和搜索的实例。有效的排序对有效的可见面判定非常重要。同样重要的是要避免过多的排序，这一点常利用相关性来实现。例如，扫描线算法用扫描线相关性消除每一扫描线上对x的完全排序。Hubschman和Zucker用帧相关性避免动画序列中不必要的比较[HUBS82]。

算法可以根据排序的方法来分类。深度排序算法先按z，然后x，然后y来排序（测试1和2中利用方向性）；因此也称为zxy算法。扫描线算法对y排序（桶排序），然后对x（先为插入顺序，然后在处理每条扫描线时采用冒泡排序），最后在z方向搜索距离视点最近的多边形；因此，这是yxz算法。Warnock的算法对x和y做平行的排序，然后搜索z方向，因此是一种$(xy)z$算法（括号表示维度的组合）。z缓存算法除了z方向外，没有明确的排序和搜索；称为(xyz)算法。

715

Sancha认为排序时依照的顺序是无关紧要的：沿某一特定轴方向排序并不比沿另一方向有实质性的优势，至少在原则上，平均来看，对象在三个方向上都具有相同的复杂度[SUTH74a]。另一方面，类似于Hollywood集合的图形场景可以通过一定的构造使其从某一特定视点看会更好，也会使其在某一方向具有更高的复杂度。尽管我们可以假定对象具有对称的复杂度，但并非所有的算法都具有相同的效率：它们的不同在于是否能有效地利用相关性来避免多余的排序和计算以及是否能在空间-时间上给出最佳的平衡。在[SUTH74a, Table VII]中给出了四种基本算法的性能评估比较，概括起来如表15-3所示。作者提出，既然都只是估计，小的差别可以忽略，但“我们希望对不同的算法有一个数量级的比较，以期能对不同算法的效率有一定的了解”[SUTH74a，p.52]。

深度排序算法对少量的多边形有效，因为对于判断一个多边形是否能扫描转换，简单的重叠测试就足够了。多边形增多时，就需要更复杂的测试，也可能需要对多边形进行分割。z缓存算法的性能是常量，因为，当场景中的多边形数目增加时，单个多边形覆盖的像素点数却减少了。另一方面，它的内存需求较大。Warnock的区域划分算法的每一步测试和计算都较复杂，因此实现起来通常比其他方法要慢。

表15-3 四种可见面判定算法的相对性能评估

	场景中多边形面的个数		
算法	100	2500	60 000
深度排序	1①	10	507
z缓存	54	54	54
扫描线	5	21	100
Warnock区域划分	11	64	307

① 经规格化后将此项作为一个单位。

除了这些非形式化的评估外，还有一部分工作是对可见面问题进行形式化的表述，并分析其运算复杂度[GILO78；FOUR88；FIUM89]。例如，Fiume[FIUM89]证明了对象精度可见面 716
算法有一个下界，该下界比排序算法要差：即使像计算n个凸多边形可见面的简单情形也会生成$\Omega(n^2)$个多边形，输出需要$\Omega(n^2)$时间（图15-67）。

一般而言，对可见面算法进行比较是很困难的，因为算法之间计算的信息和精度都不同。例如，我们讨论过的算法对对象的种类、对象之间的关系，甚至投影的类别都有限制。如我们将在第16章中看到的，可见面算法的选择也会受到明暗处理类型的影响。如果明暗处理的计算过程很复杂，最好选择一种只计算对象的可见部分明暗度的可见面算法，如扫描线算法等。此时，深度排序算法就不是很好的选择，因为它对所有的对象都做整体的绘制。当交互性能很重要时，常常选择硬件的z缓存算法。BSP树算法对静态场景能够很快地生成新的视图，但当场景发生改变时，需要附加的处理。扫描线算法能够达到很高的分辨率，因为在数据结构中需要详细记录所有影响扫描线的图元。总的来看，实现算法所需的时间以及算法可扩充的程度（如适用于新的图元）都是很重要的因素。

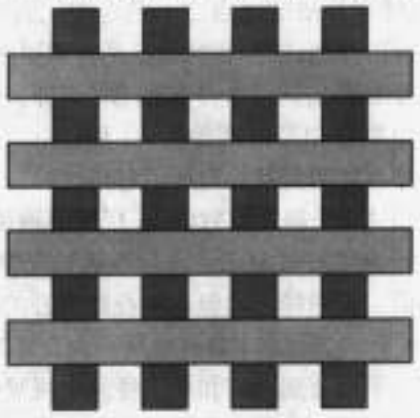


图15-67 $n/2$个矩形位于另$n/2$个较远矩形之上，使得$n/2$个矩形完全可见，另有$(n/2)(n/2+1)$个片段可见。可见多边形个数可记为$\Omega(n^2)$（[FIUM89]）

使用可见面算法的一个重要的考虑是是否有硬件支持。如果采用并行的机器，有一点必须引起注意，那就是如果算法利用相关性，则在每个位置都依赖于前一次计算生成的值。利用并发性可能必须放弃某些形式的相关性。光线跟踪非常适合于采用并发的形式执行，它的每个像素都单独进行计算。在第18章中我们将看到，许多结构专门为执行特定的可见面算法而设计。比如说，内存价格的下降使得硬件z缓存系统变得非常流行。 717

习题

15.1 证明：15.2.2节中的变换M保持(a)直线、(b)平面和(c)深度关系。

15.2 给定平面$Ax+By+Cz+D=0$，用15.2.2节中的矩阵M做变换，求出新的平面方程的系数。

15.3 如何对扫描线算法进行扩充以处理具有共享边的多边形？一条共享边是只保存一次（标记为共享边）还是在每一个归属多边形中各保存一次，不做任何标记？在公共边处计算两个多边形的深度时，深度是相等的。如果扫描正进入两个多边形，哪一个多边形是可

见的呢？

15.4 Warnock的算法生成一棵四叉树。画出对应于图15-44的四叉树。考虑三角形（T）和矩形（R），根据每个节点与它们的关系，对节点做出标记：(a)包围，(b)相交，(c)内含，(d)分离。

15.5 考虑讨论过的每种可见面算法，解释穿透多边形都是如何处理的。是作为特殊情况专门处理，还是由基本算法解决？

15.6 深度排序算法中的测试3和4如何能够高效率地实现？考虑对多边形Q的每个顶点求多边形P所在平面方程的符号，反之也执行一遍。如何确定方程的正值对应于平面的哪一边？

15.7 本章介绍的算法经过怎样的改进才能够适用于有空洞的多边形？

15.8 利用画家算法的思想，试修改15.1节中的双变量函数的可见线算法，使其能作为可见面算法。

15.9 为什么Roberts的可见线算法没有除去所有背面多边形的边？

15.10 z缓存算法的优点之一是对图元的顺序没有要求。但这是否意味着按不同顺序生成的两幅图像的z缓存和帧缓存具有相同的值呢？说明理由。

15.11 考虑两幅同样大小的图像，它们分别用帧缓存和z缓存表示，现要将它们合并在一起。如果已知每幅图像的z_{min}和z_{max}，且原始对应的z值也已知，是否能够将它们正确地合并？还需要其他附加信息吗？

15.12 15.4节提到用整数的z缓存绘制透视投影时产生的z压缩问题。请选择一个透视观察规范和少量的对象点。演示在透视变换过程中，接近投影中心的两个点如何映射到不同的z值，而同样距离但远离投影中心的两点却映射到一个z值。

15.13 a. 假设视见体V的前裁剪平面在距离F处，后裁剪平面在距离B处；视见体V'的前后裁剪平面分别为F'和B'。将两个视见体规格化后，V的后裁剪平面为$z=-1$，前裁剪面为$z=A$。对于V'而言，前裁剪平面变为$z=A'$。试证明：若$B/F=B'/F'$，则$A=A'$。

718

b. 由(a)部分可知，在考虑透视效果时，只需要考虑后平面与前平面距离的比。因此可以简单地研究具有不同前平面距离值的规范视见体。假设有一个规范的透视投影视见体，前裁剪平面为$z=A$，后裁剪平面为$z=-1$，对它做透视变换，得到$z=0$和$z=-1$之间的平行视见体。请给出用原始z坐标表达变换后新z坐标的公式。（当然，你的答案依赖于A。）假设变换到平行视见体中的z值乘以2^n，然后取整（即映射到整型的z缓存）。找出两个尽可能远的z值，使其在该变换下，映射到相同的整数。（你的答案将依赖于n和A。）

c. 假设图像的前裁剪平面到后裁剪平面的比率为R且要求两对象之间在z方向上的距离大于Q时，必须映射到z缓存中的不同值。根据(b)中的工作，试求出z缓存需要的位数。

15.14 试说明遍历BSP树所得从后向前显示顺序同深度排序算法确定的从后向前顺序不一定相同，即使不存在多边形分割也是如此。（提示：仅需要两个多边形。）

15.15 如何修改BSP树算法使其能够处理除多边形以外的对象？

15.16 如何修改BSP树算法以允许有限的移动？

15.17 假设需要设计一个支持CSG的光线跟踪算法。你如何处理非多面体组成部分的多边形？

15.18 某些图形系统在实现硬件变换与齐次坐标裁剪时对X、Y和Z采用相同处理，其裁剪的边

界是

$$-W \leqslant X \leqslant W, \quad -W \leqslant Y \leqslant W \quad -W \leqslant Z \leqslant W$$

而不是

$$-W \leqslant X \leqslant W, \quad -W \leqslant Y \leqslant W \quad -W \leqslant Z \leqslant 0$$

如何改变观察矩阵计算以实现这一点？

15.19 执行光线跟踪时，常常只需要计算光线是否与某一范围相交，而不用求实际的相交点。完成二次方程的光线与球的求交方程（式(15-17)），并说明如何将其简化成仅判断光线与球是否相交。

15.20 通过数值积分，光线跟踪可用于计算对象的质量属性。光线与对象相交的合集给出了光线位于对象内的部分。试通过发射平行光线组来估计对象的体积。

15.21 推导光线与二次曲面的交。修改用于推导光线与球相交的式(15-13)至(15-16)来处理11.4节中给出的二次曲面的定义。

15.22 在式(15-5)中，O是执行对象相交测试的代价。如果包围体相交测试的结果可以用于简化其对象相交测试，则O的代价有可能部分地受B（包围体相交测试的代价）的影响。试举出一个对象及其包围体来描述这一可能性。 719

15.23 实现本章的一个多边形可见面算法，如z缓存算法、扫描线算法或BSP树算法。

15.24 实现一个对球与多边形的简单光线跟踪算法，包含自适应超采样。（选择16.1节中的一个光照模型。）采用空间划分或建立包围体的层次结构来改进你的算法的性能。

15.25 如果已经实现了一个z缓存算法，试通过扩展7.12.2节中描述的拾取窗口方法来考虑可见面判定将命中检测加到该算法中。这里需要用到SetPickMode过程，它传递一个模式标志，标明对象是参与绘制（drawing模式）还是用于命中测试（pick模式）。SetPick Window过程设置一个矩形的拾取窗口。在pick模式起作用之前，必须先填充z缓存（绘制所有的对象）。进入pick模式后，帧缓存与z缓存都不做修改，但落入拾取窗口的图元的每个像素的z值要与相应的z缓存中的值进行比较。若新值使对象进入drawing模式（被绘制），设置模式标志。可以调用InquirePick查询该标志，然后对其重置。如果是在pick模式下对每个图元处理完毕后调用InquirePick，就可基于每一图元进行拾取。试采用InquirePick来判定某一像素处哪个对象是实际可见的。 720

第16章 光照和明暗处理

这一章我们讨论如何根据物体表面的位置、方向和特性以及照射光源的特性为物体表面加上光照效果。我们给出各种不同的光照模型（illumination model），这些光照模型表达确定物体表面给定点处颜色的各种要素。光照模型通常也叫作光照明模型（lighting model）或者明暗处理模型(shading model)。不过这里我们把明暗处理模型这一术语用于适用范围更广的光照模型。明暗处理模型决定何时运用光照模型以及接受何种参数。比如，某些明暗处理模型为图像中的每个像素调用同一个光照模型，而另外一些只为图像中的某些像素调用某种光照模型，而其他像素的颜色值则通过插值计算得到。

与前一章进行可见面计算所用精度比较，我们将严格区分使用真实物体几何的算法和使用多边形近似的算法，严格区分对象精度的和图像精度的算法，区分那些以单点采样图像精度算法和用更好的过滤器的算法。但在任何情况下，决定某物体在某一像素处是否可见的惟一标准是看通过此像素的投影线上该物体和观察者之间是否存在其他物体。与之相反，光和表面的相互作用却更复杂。或许是为了简化计算或许是因为在图形学领域中不知道更精确的模型，图形学的研究者通常对光和热辐射基本规则采用近似方法。所以，计算机图形学领域中用到的许多光照模型和明暗处理模型是通过不断地拼凑、尝试和简化得到的，虽然这些模型在理论上缺乏坚实基础但在实践中却是可行的。这一章的第一部分涵盖了这样一些简单模型。这些简单模型 721
仍被普遍使用，因为它们能以最小的计算得到引人注目的有用结果。

16.1节首先讨论那些仅考虑物体表面单独一点和直接照射它的光源的简单光照模型，首先为单色的表面和光源提出一些光照模型，然后讨论如何将这些计算推广到处理第13章讨论过的颜色系统中。16.2节将介绍一些最常用的使用这些光照模型的明暗处理模型。在16.3节，我们对这些模型进行推广，以模拟表面纹理。

模拟折射、反射和阴影需要一些类似于隐藏面消除并且往往与之结合的额外计算。实际上，这些效果的产生是因为一些被隐藏的表面并不是真的被完全隐藏。它们可以被透视、被反射或者在加上明暗色调的物体表面投下阴影。从16.4节到16.6节讨论如何模拟这些效果。接下来在16.7节中我们介绍那些更能准确刻画单个表面如何与直接照射它的光源相互作用的光照模型。紧接着从16.8节到16.10节涵盖了能产生更真实图像的其他方法。

16.11节到16.13节描述那些试图考虑所有表面之间光线交换的全局光照模型：递归光线跟踪和辐射度方法。递归光线跟踪技术将前一章介绍的可见面光线跟踪算法进一步扩展，以确定各像素的可见性、光照以及明暗处理。辐射度方法模拟一个曲面系统的能量平衡，它们在进行当前视点可见性判定计算前就决定了环境中一系列采样点的光照，并且采样点的光照与视点无关。这里所有的光照模型和明暗处理模型都可以在文献[GLAS89；HALL89]中找到。

最后，在16.14节，结合本章和前几章所讨论的光栅化技术，考察了几种不同图形绘制流水线技术，并探讨实现这些技术的途径以建立高效的可扩展的系统。

16.1 光照模型

16.1.1 环境光

可能最简单的光照模型是那个隐含运用在本书前几章的模型，在其中每个物体用一种内蕴亮

度进行显示。我们可以想像为这样一个模型，它没有任何外部光源，没有任何反射，物体自身发光，这样的世界当然是不真实的。除非是当该物体创建时赋予物体的不同部分（如一个多面体的各个面）以不同的色调，否则每个物体将显示为一个单色的轮廓。彩图II-28展示了这样一种效果。

一个光照模型可以表示为一些变量的光照方程，这些变量与带消隐的物体上的点相关。表示这个简单光照模型的光照方程为

722

$$I = k_i \tag{16-1}$$

其中I是所得亮度，系数k_i是该物体内设光亮度。因为此光照方程中不包含任何依赖于受照射的点的位置的项，所以对整个物体只需要计算一次。对物体上一个或更多点对光照方程求值的过程通常称为照明物体。

现在想像一下，物体不是自发光的，而是存在一个漫射的无方向的光源，环境中存在的光是经过多个表面多次反射后所得到的结果，这种光通常被称为环境光（泛光）。如果我们假设环境光从各个方向均衡地照射到所有物体表面，那么光照方程为

$$I = I_a k_a \tag{16-2}$$

其中I_a是环境光亮度，假设对所有物体都是常数。从物体表面所反射的环境光的量由环境反射系数k_a决定，它的取值范围为0到1。环境光反射系数是一个材质属性。与我们将要讨论的其他各种材质属性一样，它可以看成是刻画构造该物体表面所用材质的特性。像其他的一些性质一样，环境光反射系数是一个适当的经验值，并不直接对应于真实材料的任何物理属性。而且对环境光本身影响不是很大。正如我们将看到的，它通常被用于考虑所有在实际中光能够到达物体但却不能为光照方程所计算的复杂情况。彩图II-28给出了环境光下的光照效果。

16.1.2 漫反射

虽然在环境光照射下物体的亮度差不多与环境光的亮度成正比，但物体表面仍呈现出单一的光照效果。现在考虑以点光源照射的一个物体，该点光源的光线从一点向四周均匀发散。根据到光源的不同方向和距离，物体从一部分到另一部分的辉度将有所不同。

1. 朗伯反射

暗的粗糙表面，如粉笔，展现出漫反射效果，也被称为朗伯反射。因为这些表面从各个方向等强度地反射光，因而从各个视角物体表面呈现出相同的亮度。对于给定的表面，该表面亮度仅依赖于图16-1中该表面到光源的方向$\bar{L}$与物体表面法向$\bar{N}$的夹角。让我们看看为什么会发生这种情况。在这里有两个因素起作用。第一，图16-2显示照射到物体表面的一束光线覆盖了该物体表面一定的面积，该面积大小与该光束与物体表面法向$\bar{N}$所成的夹角θ的余弦成反比。如果光束有一极微小的截面微面元dA，那么该光束在物体表面上的截面积为$dA/\cos\theta$。因此，对一入射光束，落在dA面积上的光的能量与$\cos\theta$成比例。这对所有表面都成立，而与构造该表面的材质无关。

723

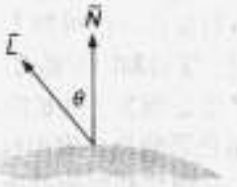


图16-1 漫反射

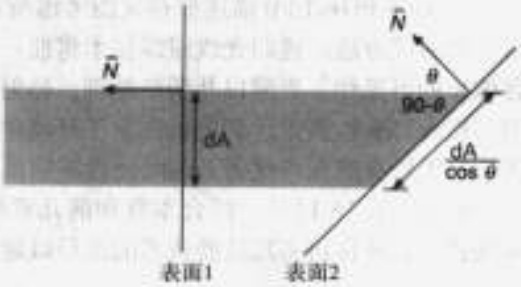


图16-2 具有极微小横截面面积dA的以入射角θ入射的光束（用二维截面图表示）照射到$dA/\cos\theta$的面积

第二，我们必须考虑观察者所看到的光的

量。朗伯表面具有这样一个性质，通常被称为朗伯定律，即从单位微面元dA向观察者反射的光的量直接与物体表面到观察者的方向与物体表面法向 $\overline{N}$ 夹角的余弦成正比。但是所看到的物体表面面积量与这个夹角的余弦成反比，因此这两个因子相抵消。比如，当视角增大，观察者看到更大的物体表面面积，但从此方向上单位面积表面所反射的光量也相应地成比例地少了。因此，对朗伯表面，被观察者所看到的光量是独立于视线方向并且仅与入射光角θ的余弦$\cos\theta$成比例的。

漫反射光照方程是

$$I = I_p k_d \cos\theta \tag{16-3}$$

I_p是点光源的亮度。材质的*漫反射系数*k_d是一个从0到1之间的常数并随材质的不同而变化。如果光源对它所照射的点有任何直接作用，则该光源入射角θ必须在0°到90°之间。这意味着该表面可以进行自遮挡处理，因此从表面一点的后面所投射的光并不照射到它。在这里和下面的等式中，与其不显式地包含一个$\max(\cos\theta,0)$项，还不如假设θ在有效范围内。当我们希望为一个自遮挡的物体表面加上光照时，我们可以用$\text{abs}(\cos\theta)$来倒转它们的表面法向。这使得该表面的两面得以被以同样的方式处理，好像该表面被两个相对的光源照射。

假设向量 $\overline{N}$ 和 $\overline{L}$ 已经被规格化（参考附录）。我们可以用点积重写式(16-3)

$$I = I_p k_d (\overline{N} \cdot \overline{L}) \tag{16-4}$$

表面法向 $\overline{N}$ 可以用第11章所讨论的方法计算。如果多边形的法向预先计算并以对多边形顶点做变换的同一矩阵进行变换，那么重要的是不做诸如错切或不同的缩放那样的非刚体模型变换，因为 724
这些变换不是保角变换，而且可能使得一些法向量不再垂直于它们的多边形。5.6节给出了对物体进行任意变换时，对法向做转换的正确方法。在任何情况下，光照方程必须在世界坐标系下（或者与之尺寸相同的任何坐标系）计算，因为规格化和透视投影变换都会改变θ值。

如果一个点光源与它所照射的物体足够远，这使得它与所有法向相同的表面成同样的入射角。这种情况下，这种光称为*方向光源*，此时 $\overline{L}$ 是常数。

图16-3展示了一个点光源照射下的球体的一系列图片。该明暗处理模型用式(16-4)的光照模型为该球体每一个可见的像素计算亮度。以此方式加上明暗的物体看起来很粗糙，就像一支手电在一个相当暗的房间照射一个物体。所以，通常加入环境光项以产生更真实的光照方程

$$I = I_a k_a + I_p k_d (\overline{N} \cdot \overline{L}) \tag{16-5}$$

图16-4是用公式(16-5)产生的画面。



图16-3 用漫反射模型（式(16-4)）加上明暗的球。对所有球$I_p = 0.1$。从左至右，$k_d = 0.4$，0.55，0.7，0.85，1.0。（由哥伦比亚大学David Kurlander提供。）

2. *光源的衰减*

由视点处照射，如果两个具有同样材质的平行平面的投影在图像中相互重叠，那么无论它们与光源的距离如何不同，由式(16-5)生成的画面将无法区别光线是从哪个表面照射到哪个表面的。为处理这种情况，我们引进了光源衰减因子f_{att}，从而有

$$I = I_a k_a + f_{att} I_p k_d (\overline{N} \cdot \overline{L}) \tag{16-6}$$

f_{att}的选择显然应考虑到这样一个事实，即从点光源到达物体表面某部分的光能量是以d_L的平方的倒数衰减的，d_L是点光源到物体表面的距离。在这种情况下，

$$f_{att} = \frac{1}{d_L^2} \tag{16-7}$$

然而，实际上这种假设并不总是对的。如果光很远，$1/d_L^2$的变化并不大；如果它很近，则变化很大，这样，$\overline{N}$与$\overline{L}$之间相同的夹角却因在不同表面而产生相当不同的明暗效果。虽然这种运行情况对点光源是正确的，但实际中我们所见的物体通常并不为点光源所照射，也不具有计算机图形学中简化了的光照模型计算为物体加上的明暗阴影效果。为使效果更逼真，早期的图形学研究者通常使用一个放在视点处的点光源。他们期望用f_{att}来近似模拟在观察者和物体之间的大气衰减现象（见16.1.3节），同时近似模拟从光源到物体的能量密度的衰减。一种容许比简单的平方规律衰减效果更好的可行折衷是

$$f_{att} = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right) \tag{16-8}$$

这里c_1、c_2和c_3是用户定义的与光源相关的常数。当光源很近时，常数c_1防止分母变得太小，同时该表达式被限定在最大值1以内以确保其总是衰减的。图16-5即是采用这个常数取不同值的光照模型所显示的一系列不同的效果。



图16-4 用环境和漫反射（等式(16-5)）加上明暗的球。对所有球，$I_a = I_p = 1.0$，$k_d = 0.4$，$k_a = 0.0$，0.15，0.30，0.45，0.60。（由哥伦比亚大学David Kurlander提供。）

3. *彩色光和表面*

至今我们描述的都是单色光和表面。彩色光和表面通常通过为颜色模型中每个组成部分写不同的方程的方式进行处理。我们通过O_d的一个值来表示一个物体的漫射颜色的每一部分。比如，在RGB颜色系统中，三元组（O_{dR}，O_{dG}，O_{dB}）定义了一个物体的漫射红、绿、蓝组成部分。在这种情况下，照射光的三个主要组成部分I_{pR}，I_{pG}和I_{pB}分别以$k_d O_{dR}$，$k_d O_{dG}$和$k_d O_{dB}$的比例反射。所以，对红色部分来说，

$$I_R = I_{aR} k_a O_{dR} + f_{att} I_{pR} k_d O_{dR} (\overline{N} \cdot \overline{L}) \tag{16-9}$$

对绿色和蓝色部分I_G和I_B采用相似的等式。使用单一的系数来放大或缩小等式中的表达式使用户不需改变它们的分量的比例就可以控制环境反射或漫反射的量。另一个表达更紧凑但不方便控制的公式则简单地为每个分量采用不同的常数，比如，用k_{aR}代替$k_a O_{dR}$，用k_{dR}代替$k_d O_{dR}$。

这里做了一个简化的假设，即三色模型能完全模拟光与物体的相互作用。正如我们将在16.9节中要讨论的那样，这种假设是错误的，但它更容易实现且常常产生可接受的图片。在理论上，应该在模拟的光谱范围对光照方程进行连续的取值；但在实际中，只在一些间断光谱样本上进行取值。不是将我们自己约束于特定的颜色模型，我们在光照方程中用下标λ显式地标明那些与波长相关的量。这样，等式(16-9)变为

725
|
726

$$I_\lambda = I_{a\lambda}k_a O_{d\lambda} + f_{att}I_{p\lambda}k_d O_{d\lambda}(\bar{N}\cdot\bar{L}) \tag{16-10}$$

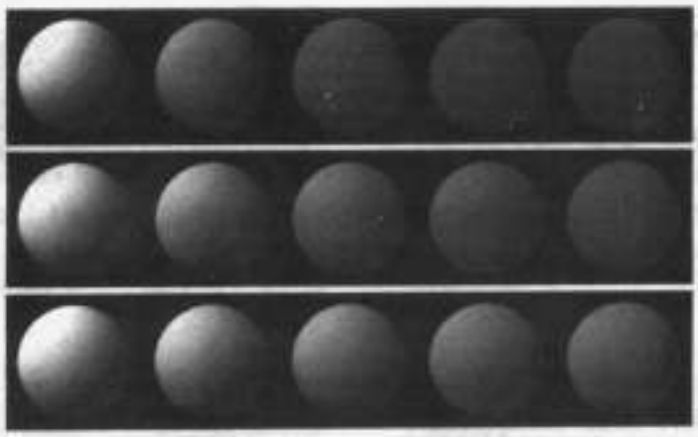


图16-5 用带点光源衰减项（等式(16-6)和(16-8)）做明暗处理后的球的环境和漫反射。对每个球，$I_a = I_p = 1.0$，$k_a = 0.1$，$k_d = 0.9$。从左至右，从光源到球的距离为1.0, 1.375, 1.75, 2.125和2.5。第一行，$c_1 = c_2 = 0.0$，$c_3 = 1.0(1/d_l^2)$；第二行，$c_1 = c_2 = 0.25$，$c_3 = 0.5$；第三行，$c_1 = 0.0$，$c_2 = 1.0$，$c_3 = 0.0(1/d_l)$。（由哥伦比亚大学David Kurlander提供。）

16.1.3 大气衰减

为了模拟从物体到观察者的大气衰减，许多系统提供深度提示（depth cueing）。在这种最初起源于向量图形硬件的技术中，用比近的物体低的亮度绘制更远的物体。PHIGS + 标准推荐了一种深度提示方法，这种方法使得由于居间大气层所引起的颜色变化的近似计算成为可能。在NPC中分别定义前后深度提示参考平面，每个平面分别与一个取值为0到1的缩放因子s_f、s_b相联系。这个缩放因子决定了原始亮度与深度提示颜色的亮度混合。目标是为了修正前面计算的I_λ值以产生最终显示的深度提示值I'_λ。给定物体的z坐标z_o，可以推得一个缩放因子s_o，并将其用于I_λ与$I_{dc\lambda}$之间的插值中以决定

$$I'_\lambda = s_o I_\lambda + (1 - s_o)I_{dc\lambda} \tag{16-11}$$

727

如果z_o在深度提示前参考平面的z坐标值z_f前，那么$s_o = s_f$；如果z_o在深度提示后参考平面的z坐标值z_b后，那么$s_o = s_b$。最后，如果z_o在两参考平面之间，那么

$$s_o = s_b + \frac{(z_o - z_b)(s_f - s_b)}{z_f - z_b} \tag{16-12}$$

s_o与z_o的关系如图16-6所示。图16-7表示加上深度提示的明暗效果的不同球。为避免使等式复杂

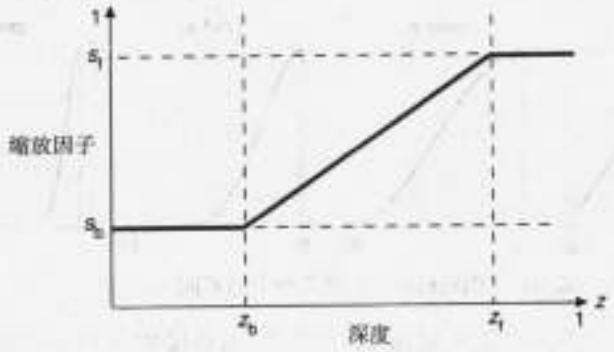


图16-6 计算大气衰减缩放因子

化，在进一步提出光照模型时，我们忽略深度提示效果。在20.8.2节中将讨论模拟大气效果的更真实的方法。



图16-7 用深度提示（等式(16-5)，(16-11)和(16-12)）加上明暗的球。到光源的距离是常值。对每个球，$I_s=I_p=1.0$，$k_s=0.1$，$k_d=0.9$，$z_l=1.0$，$z_h=0.0$，$s_l=1.0$，$s_h=0.1$，半经=0.09。从左至右，球的最前面一点z值为1.0, 0.77, 0.55, 0.32和0.09（由哥伦比亚大学David Kurlander提供。）

16.1.4 镜面反射

在任何发光表面上可以观察到镜面反射现象。用明亮白光照射一个苹果：强光由镜面反射引起，而从其他部分反射的光则是漫反射的结果。同时可注意到，在强光处苹果并非呈现出红色，而是入射光的颜色——白色。像蜡制的苹果或发光的塑料这样的物体，都有一个透光的表面；比如，塑料，是典型的由沉淀于透明材料中的颜色颗粒所构成的。从无色的表面镜面反射的光通常呈现出与光源相同的颜色。

728

现在移动您的头部，可观察到强光也在移动。出现这种现象是因为闪光表面在不同方向上反射光不相同：在一个理想的闪光表面，如一个完全平坦的镜子，光只在反射方向$\bar{R}$上被反射，该反射方向与$\bar{L}$是关于$\bar{N}$对称的。因此，观察者只有在图16-8中的角α为0时，才能观察从镜子反射的镜面光。这里α是$\bar{R}$到视点$\bar{V}$的方向的夹角。

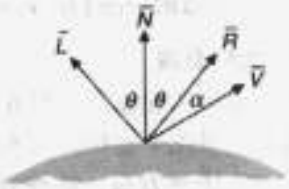


图16-8 镜面反射

1. Phong光照模型

Phong Bui-Tuong[BUIT75]为不完全平坦的反射面（如苹果）提出了一种非常受欢迎的光照模型。它假设当α为0时，发生的镜面反射最大，并且随着α的增大镜面反射大幅度地减弱。这种快速衰减效果由$\cos^n\alpha$来近似，这里n是这种材质的镜面反射指数。依据所模拟的表面材质，n的值从1到几百变化不等。当n为1时，强光区域范围较大、衰减也较缓慢，而n值较大时则模拟强烈的聚焦强光（图16-9）。对一个完美的反射体，n是一个无穷大的值。与以前一样，我们把$\cos\alpha$的负值当作0处理。Phong光照模型基于Warnock等研究者早期的工作，他们曾用项$\cos^n\alpha$来模拟放置在视点处的光源的镜面反射效果。然而Phong是第一个考虑观察者与光源可以放置在任意位置的人。

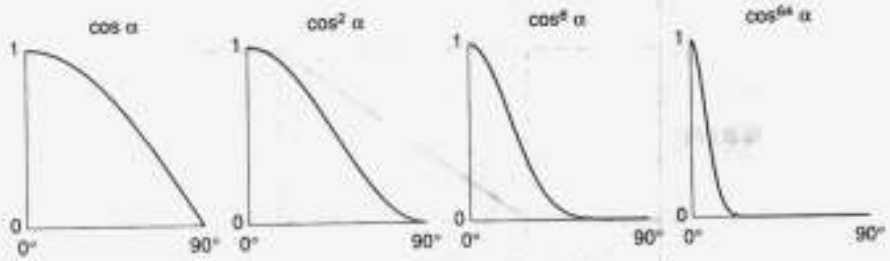


图16-9 用在Phong光照模型中的不同$\cos^n\alpha$值

镜面反射的入射光的量依赖于入射角θ。如果$W(\theta)$是镜面反射光的比例，那么Phong模型是：

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}I_{p\lambda}\,[k_dO_{d\lambda}\cos\theta + W(\theta)\cos^n\alpha] \tag{16-13}$$

如果反射方向$\overline{R}$和视点方向$\overline{V}$已规格化，那么$\cos\alpha = \overline{R}\cdot\overline{V}$。此外，$W(\theta)$通常设为常数$k_s$，$k_s$称为该材质的镜面反射系数，一般在0至1之间变化。$k_s$的值依据经验选取，以产生美观的令人 729
愉悦的效果。那么，等式(16-13)可以重写为

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}I_{p\lambda}[k_dO_{d\lambda}(\overline{N}\cdot\overline{L}) + k_s(\overline{R}\cdot\overline{V})^n] \tag{16-14}$$

注意，Phong光照模型中的镜面反射分量的颜色不依赖于任何材质属性。因此，此模型能很好地模拟塑料表面的镜面反射。如我们在16.7节中讨论的，镜面反射受材质表面本身属性的影响，并且，当该材质表面是不同材料组合而成时，镜面反射通常具有不同于漫反射的颜色。我们可以通过修改方程(16-14)中的一个初步近似值来提供这种效果。

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + f_{att}I_{p\lambda}[k_dO_{d\lambda}(\overline{N}\cdot\overline{L}) + k_sO_{s\lambda}(\overline{R}\cdot\overline{V})^n] \tag{16-15}$$

这里$O_{s\lambda}$是物体的镜面反射颜色。图16-10给出用等式(16-14)中不同k_s和n值所计算的球面的光照。

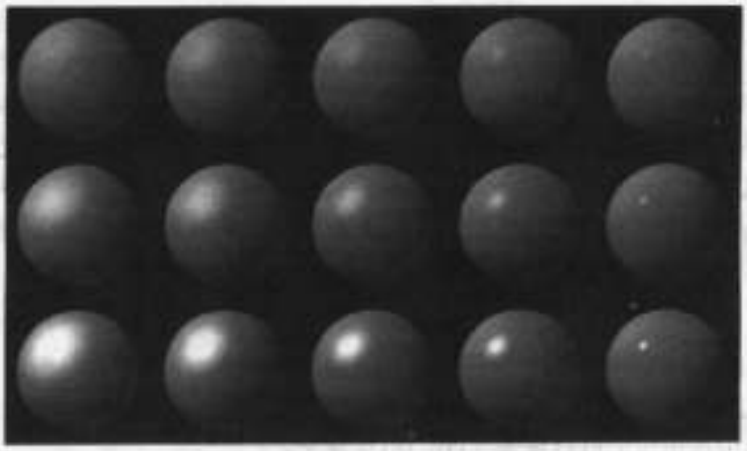


图16-10 使用Phong光照模型和不同的k_s和n值加上明暗的球面。对每个球，$I_a = I_p = 1.0$，$k_a = 0.1$，$k_d = 0.45$。从左至右，$n = 3.0$，5.0，10.0，27.0，200.0。从上到下，$k_s = 0.1$，0.25，0.5。（由哥伦比亚大学David Kurlander提供。）

2. 计算反射向量

计算$\overline{R}$要求$\overline{L}$关于$\overline{N}$的对称向量。如图16-11所示，这可以通过简单的几何计算完成。因为$\overline{N}$和$\overline{L}$已规格化，$\overline{L}$到$\overline{N}$上的投影是$\overline{N}\cos\theta$。注意，这里$\overline{R} = \overline{N}\cos\theta + \overline{S}$，$|\overline{S}|$为$\sin\theta$。但通过向量减法和全等三角形，$\overline{S} = \overline{N}\cos\theta - \overline{L}$。所以$\overline{R} = 2\overline{N}\cos\theta - \overline{V}$。用$\overline{N}\cdot\overline{L}$代替$\cos\theta$，$\overline{R}\cdot\overline{V}$代替$\cos\alpha$，得到

$$\overline{R} = 2\overline{N}(\overline{N}\cdot\overline{L}) - \overline{L} \tag{16-16}$$

$$\overline{R}\cdot\overline{V} = (2\overline{N}(\overline{N}\cdot\overline{L}) - \overline{L})\cdot\overline{V} \tag{16-17}$$ 730

如果光源在无穷远处，对所有给定的多边形，$\overline{N}\cdot\overline{L}$都是常数，而$\overline{R}\cdot\overline{V}$则在给定的多边形上要变化。对曲面表面或者光源不在无限远处的，$\overline{N}\cdot\overline{L}$和$\overline{R}\cdot\overline{V}$在表面上都要变化。

3. 中间向量

Phong光照模型的另一个公式使用中间向量$\overline{H}$，之所以这样叫是因为它的方向正如图16-12中所示处在光源方向和观察者方向中间。$\overline{H}$也是人们所知的为最强强光方向。如果调整表面位置使

得它的法向与$\overline{H}$相同，则观察者将会看到最明亮的镜面强光，因为$\overline{R}$和$\overline{V}$会指向同一个方向。新的镜面反射项可以表示为$(\overline{N}\cdot\overline{H})^n$，其中$\overline{H}=(\overline{L}+\overline{V})/|\overline{L}+\overline{V}|$。当光源和观察者都在无穷远处时，$\overline{N}\cdot\overline{H}$的使用提供了计算优势，因为$\overline{H}$是常数。注意，$\overline{N}$与$\overline{H}$的夹角$\beta$并不等于$\overline{R}$与$\overline{V}$的夹角$\alpha$，因此，同样的强光指数$n$在这两个公式中将产生不同的结果（见习题16.1）。虽然用项$\cos^n$允许生成明显的光滑表面，但应该记住该项基于的是经验观察结果，而非镜面反射过程的理论模型。

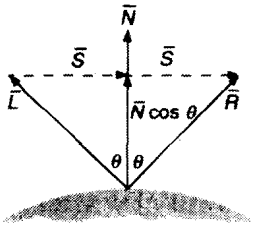


图16-11 计算反射向量

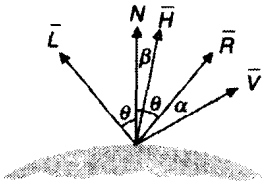


图16-12 中间向量$\overline{H}$在光源方向和观察者方向的中间

16.1.5 点光源模型的改进

真实的光源并不在各个方向等同地发散光能。Warn[WARN83]提出一些易于实现的光照控制方法，也就是说，可以将这些控制加入到任何光照方程中，用来模拟摄影者使用的光的定向方法。在Phong模型中，点光源只有一个亮度值和一个位置，而在Warn的模型中，通过一个假定的镜面反射表面上的点来模拟一个光源L（如图16-13所示）。该表面为点光源L'从方向$\overline{L}'$照
731 射。假设$\overline{L}'$垂直于这个假设的反射平面，那么，我们可以用Phong光照模型以$\overline{L}$与$\overline{L}'$之间的夹角决定该表面任一点所模拟的光的亮度。如果进一步假设反射面仅反射镜面光并且镜面反射系数为1，那么在表面上一点处光的亮度为

$$I_{L'\lambda}\cos^p\gamma \tag{16-18}$$

其中，$I_{L'\lambda}$是假设的点光源的亮度；p是反射物的镜面反射指数；γ是$\overline{L}$与假设表面法向$\overline{L}'$的夹角，它是到L'的方向。式(16-18)模拟了一个对称的有向光源，其对称轴为$\overline{L}'$，是所模拟的光的假设照射方向。用点积我们可以将等式(16-18)写为

$$I_{L'\lambda}(-\overline{L}\cdot\overline{L}')^p \tag{16-19}$$

另外，我们将负的点积视为0。因此，可用式(16-19)代替式(16-15)或其他任何光照方程中的光源亮度项$I_{p\lambda}$。比一比均匀发光的点光源的亮度分布与图16-14中Warn的点光源的$\cos^p$分布。以截面图描述每一种分布，并将亮度表示为极坐标系统中绕光轴夹角方向的函数。$\overline{L}'$表示为箭头。这些图叫作测角图（goniometric diagram）。p值越大，则光越集中在$\overline{L}'$方向上。因此，大的p值可以模拟高度方向性的点光源，而小的值则模拟一个均匀发散的柱光源。如果p为0，那么光就像一个均匀发散的点光源。图16-15a至图16-15c展示了不同p值的效果。Verbeck[VERB84]和Nishita等人[NISH85b]分别模拟了具有更复杂的非均匀亮度和光谱分布的点光源。然而，一般地，一旦决定了从一个特定方向看过去的点光源的亮度，就可以将这个值用在任何光照方程中。

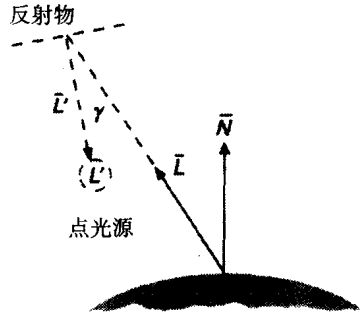


图16-13 Warn的光照模型。通过由点光源照射下的一点的镜面反射来模拟一个光源

为了将光的效应局限于场景中的有限区域内，Warn实现了挡板（flap）和锥体（cone）的模拟。挡板仿照专业摄影中的挡光板制作的，将光的效果限制于x、y、z坐标中的预定范围。

每个光有六个挡板，分别对应于用户定义的各坐标轴的最小值和最大值。每个挡板还有一个标记位表明它是开的还是关的。在决定一个点的色深时，只有当该点的坐标值在那些开着的挡板所定义的最小和最大坐标值的范围内时，才对一个光源进行光照模型的计算。比如，如果$\bar{L}'$平行于y轴，那么，就像摄影灯中的挡光板一样，x轴、z轴上的挡板可以严格限制此光的效果。图16-16a描述在这种情形下x轴上的挡板的运用。y轴上的挡板也可以在这里用于限制光线，但这种方式在实际中并不存在，它只容许在一定光源范围内的物体受到该光源的照射。在图16-15d中，立方体与坐标系统对齐，因此两对挡板就可以产生如图16-15所示的效果。

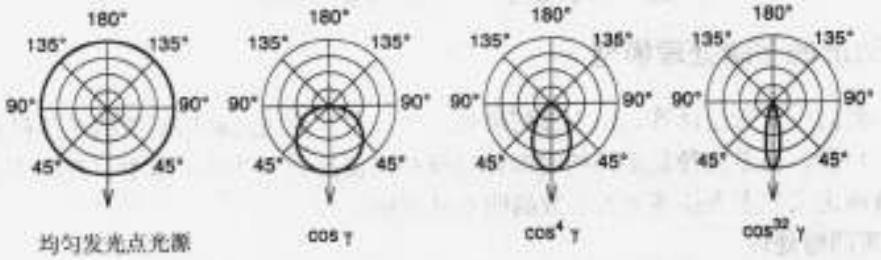


图16-14 均匀发光的点光源和具有不同p值的Warn光源模型的亮度分布



图16-15 用Warn的光源控制照射的立方体和平面。a)均匀发光的点光源（或$p=0$），b)$p=4$，c)$p=32$，d)挡板的效果，e)$\delta=18°$的锥体。（由哥伦比亚大学David Kurlander提供。）

通过一个锥顶放在光源处而轴落在$\bar{L}'$上的锥体，Warn产生了轮廓分明的聚光效果。如图16-16b所示，通过计算仅当$\gamma<\delta$时（或者当$\cos\gamma>\cos\delta$时，因为已经计算了$\cos\gamma$）的光照模型，一个具有δ大小生成角的锥体可以用来限制光源的影响范围。PHIGS + 中的光照模型包含了Warn的$\cos^n\gamma$项和锥角δ。图16-15e显示了如何用锥体来限制光源。彩图II-17是一幅用Warn的光照控制绘制的汽车。

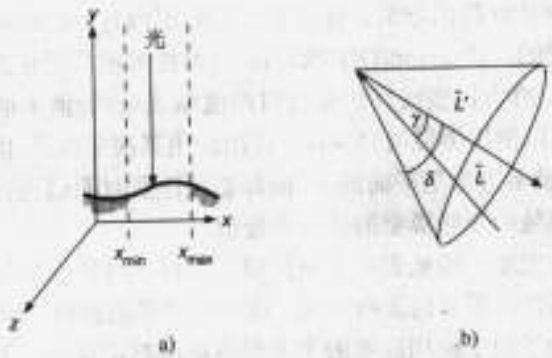


图16-16 Warn光源模型的亮度分布可以用a)挡板和b)锥体来控制

16.1.6 多光源

如果有m个光源，那么每个光源项的和是

$$I_\lambda = I_{a\lambda}k_a O_{d\lambda} + \sum_{1 \le i \le m} f_{att_i} I_{p\lambda i} [k_d O_{d\lambda}(\bar{N} \cdot \bar{L}_i) + k_s O_{s\lambda} (\bar{R}_i \cdot \bar{V})^n] \quad (16\text{-}20)$$

这个和式可能隐含了一个新的错误，即I_λ可能超出了最大的可显示像素值。（虽然对一个光源这个错误也可能发生，但我们可以通过选择适当的f_{att}和材料参数很容易地避免它。）有一些方法可以用来避免溢出。最简单的方法是将每个值取上限。另一个方法是一起考虑所有像素的I_λ值。如果至少有一个太大，那么将每个值除以这个最大值以保证色彩和饱和度。如果在显示前可以计算出所有像素的值，那么可以将图像处理中的变换作用在整幅图上以使所有值在所要求的范围内。Hall[HALL89]对这些技术和另外一些技术进行了比较。

16.2 多边形的明暗处理模型

显然，我们可以通过计算表面上任何可见点的法向并为该点调用所需的光照模型来给任何面加上明暗色调。但是这种最原始的明暗处理模型代价昂贵。在这一节中，我们为由多边形和多边形网格所定义的表面描述更为有效的明暗处理模型。

16.2.1 恒定明暗处理

最简单多边形光滑明暗处理模型是*恒定明暗处理*，通常也称为*面片明暗处理*或*挡板明暗处理*。该方法用光照模型对整个多边形只计算一次亮度值，用于处理整个多边形的明暗效果。实质上，只是为每个多边形进行一次光照模型的值采样，并在多边形内保持这个值来重建此多边形的明暗色调。在以下假设成立的情况下这种方法是正确的：

1) 光源在无穷远处，因此在多边形范围内$\bar{N} \cdot \bar{L}$是常量。

2) 观察者位于无穷远处，因此在多边形范围内$\bar{N} \cdot \bar{L}$是常量。

732
ι
734

3) 多边形代表了所模拟的真实表面，而不是对物体曲面表面的一个近似。

如果使用的是一个输出一系列多边形的可见面判定算法，如列表优先级算法，恒定明暗处理可以利用这个处处为单色的二维多边形图元。

如果前两个假设中的任何一个是错误的，那么，如果我们仍用恒定明暗处理，则需要一些方法为$\bar{N}$和$\bar{V}$分别决定一个值。比如，该值可以是为多边形中心计算的值或者为多边形第一个顶点计算的值。当然，恒定明暗处理不会在多边形上产生本应在此情况下发生的明暗的变化。

16.2.2 插值明暗处理

为减少多边形每点光照方程的计算，Wylie、Romney、Evans和Erdahl[WYLI67]首先提出了*插值明暗处理*多边形光照计算替代方法。在这里，三角形内各处的明暗度信息是通过对其顶点的明暗值的线性插值得到的。Gouraud[GOUR71]将这种技术推广至任意多边形。该技术非常容易用扫描线算法来实现，因为扫描线算法即通过跨度端点处的z值来插值计算得到整个跨度上的z值。为提高效率，可以使用类似与15.4节中提出的用其决定像素z值的差分方程。虽然对z值进行插值是正确的（假设多边形是平面的），但却必须注意对明暗进行插值处理却不是，因为它仅仅是对多边形上各点处的光照模型的计算的近似。

最后的这个假设（即多边形正确地表示了所模拟的表面）却往往是错误的。它对最终图像的影响比其他两个假设的错误所带来的影响更大。许多物体是曲面的，而不是多边形的。然而，仍用多边形网格表示只是为了便于使用高效的多边形可见面判定算法。下一步我们讨论如何绘制多边形网格以使它尽可能看起来更像曲面。

16.2.3 多边形网格的明暗处理

假设用多边形网格来近似曲面表面。如果网格中的每个多边形面单独进行明暗处理，那么

它就很容易与周围不同方向的面区分开来，从而产生如彩图II-29中所示的面效果。如果多边形以恒定明暗处理、插值明暗处理甚至单像素光照计算进行绘制，都将产生这个效果，因为相邻的两个具有不同方向的多边形在它们的边界都具有不同的亮度。结果表明，使用更精细的多边形网格这种简单解决方法很低效，因为在两个相邻面上所觉察到的色调的不同被马赫带效应加剧（由马赫于1865年发现并在[RATL72]中详细描述），在具有不连续的亮度大小或变化的边界处，马赫带效应将加剧其亮度变化。在两个面的边界，暗的面看起来更暗，而亮的面看起来更亮。图16-17显示，在两种不同情况下真正的和可察觉到的沿表面的亮度变化。

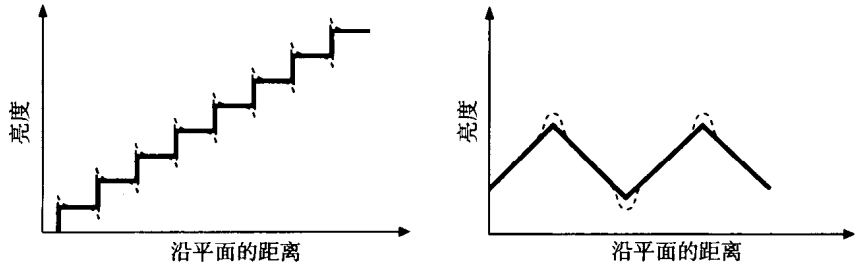


图16-17 真正的和在马赫带效应中察觉到的亮度。虚线是察觉到的亮度，实线是真实亮度

马赫带效应是由眼中感光细胞的*侧向抑制*所引起的。感光细胞接受的光越多，它对邻接感光细胞的反应则抑制得越大。感光细胞对光反应的抑制与邻接感光细胞的距离成反比。在亮度不连续处较亮一侧的感光细胞中，紧邻该不连续处的将比远离该不连续处的感光细胞具有更强的反应，因为它们受到较暗一侧的邻接感光细胞较少的抑制。同样，在较暗一侧的感光细胞中，处在暗区的将比那些离该暗区远的感光细胞有较弱的反应，因为它们受处在较亮区域相邻感光细胞较多的抑制。彩图II-29中的马赫带效应非常明显，特别是在那些颜色相近的相邻多边形之间。

我们已描述的多边形明暗处理算法都是单独决定各多边形的明暗色调的。另有两种基本的多边形网格明暗处理模型，这两种明暗处理模型利用多边形的相邻信息来模拟平滑表面。以复杂度（或真实感效果）增加的顺序，通称为Gouraud明暗处理和Phong明暗处理，分别以提出它们的研究者的名字命名。当前三维图形工作站一般通过硬件和固件的结合来支持这两个方法中的一个或两个。

16.2.4 Gouraud 明暗处理技术

Gouraud明暗处理[GOUR71]又被称为*亮度插值明暗处理*或*颜色插值明暗处理*技术，它消除了亮度的不连续性。彩图II-30采用了Gouraud明暗处理技术。虽然大部分在彩图II-29中的马赫带现象在彩图II-30中不再可见，如圆环面和圆锥体上的亮脊即是马赫带效应，它是由亮度曲线斜率的陡然变化而不是非连续所引起的马赫带现象。Gouraud明暗处理不能完全消除这种亮度变化。

通过插值多边形顶点的光照值（这些光照值充分考虑到多边形网格所近似的表面），Gouraud 明暗处理技术扩充了施加于单个多边形上的明暗插值概念。Gouraud明暗处理技术过程要求多边形网格的每个顶点的法线是已知的。这可以通过两种方法获取。Gouraud 明暗处理技术可以直接从物体表面的解析描述中计算这些*顶点法线*。另一个方法是，如果网格中没有存储顶点的法向并且又不能直接从真实表面决定，Gouraud建议，可以通过共享该顶点的所有多边形面的法向的平均来近似顶点法线（图16-18）。如果希望一条边可见（如在飞机的机翼与机身的连接处），那么我们可以通过平均该边一侧的多边形的法向找到两个顶点法线，边的两边一边一个。彩图II-30中茶壶上的折痕处的向量没有平均（见彩图II-30的说明）。

735 ~ 736

下一步，Gouraud明暗处理技术通过将顶点法线运用到所要求的光照模型中找到顶点的亮度。最后以类似于15.4节中描述的z值插值方法为整个多边形加上明暗，即先用多边形每条边顶点的亮度值插值出当前扫描线与多边形边交点处的亮度，然后再用交点的光亮值插值出扫描线位于多边形内区段上每个像素处的亮度值（图16-19）。术语Gouraud 明暗处理通常被推广到甚至是独立的一个多边形的亮度插值的明暗处理，或者是对于与多边形顶点相关的任意颜色值的插值。

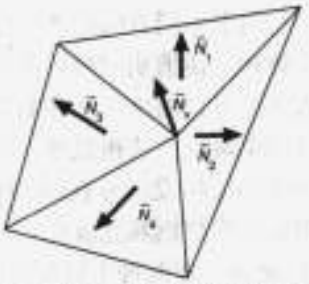


图16-18 可以通过平均规范化的多边形向量来获取顶点法线。平均法向量$\overline{N}_v$为$\sum_{1\le i\le n}\overline{N}_i/\left|\sum_{1\le i\le n}\overline{N}_i\right|$

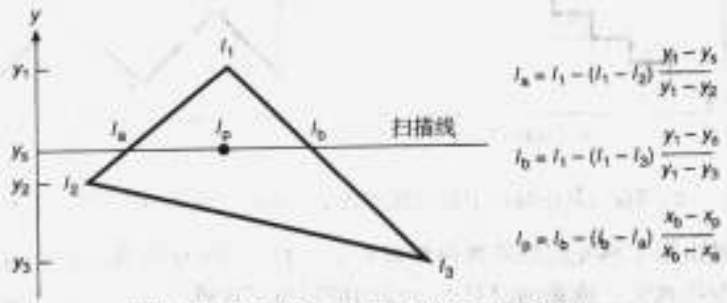


图16-19 沿多边形边和扫描线的亮度插值

Gouraud明暗处理沿边的插值可以非常容易地与15.6节中的扫描线可见面算法结合起来。对每条边，我们对每个颜色分量记下起始亮度和亮度在y方向上的单位变化量。可以通过扫描线可见跨度的多边形边的亮度插值对可见跨度进行填充扫描。与在所有的线性插值算法中一样，
737 可以用差分方程来增加效率。

16.2.5 Phong明暗处理技术

Phong明暗处理技术[BUIT75]，又称为法向量插值明暗处理技术，是对表面的法向量N而非亮度进行插值。插值发生在扫描线位于多边形内的跨度上，并且在跨度起始和终点向量之间插值。如果需要，就像在Gouraud明暗处理技术中一样，这些跨度端点处的向量自身又是沿着多边形边通过顶点向量的插值得到的。线性插值可以使用增量法进行计算。向量的三个分量在扫描线之间进行递增。沿扫描线的每个像素，插值所得到的向量首先规格化，并逆映射到世界坐标系或者与之尺度相同的坐标系中，然后用光照模型进行新的亮度的计算。图16-20显示规格化前后的两个边法向量和它们插值得到的法向量。

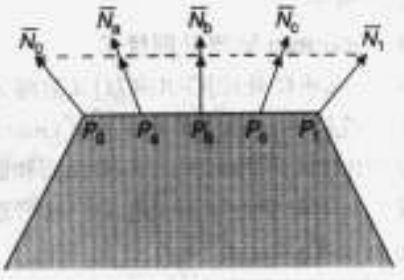


图16-20 法向量插值（来源于[BUIT75]）

彩图II-31和III-32分别是用Gouraud明暗处理、Phong明暗处理和一个带有镜面反射项的光照方程生成的。当使用这样的光照模型时，Phong明暗处理大大地改善了Gouraud明暗处理。因为正如图16-21所展示的，Phong模型再现了更真实的高光。想一想，如果在Phong光照模型中的$\cos^n\alpha$光照项的n很大，且其中一个顶点有很小的α值而其周围的顶点却有很大的α时，会发生什么？具有小α值的顶点的亮度相当于高光，但其他顶点却没有高光的亮度。如果使用Gouraud

明暗处理，那么，多边形内部的亮度通过该高光亮度和相邻顶点的非高光亮度的线性插值得到，将高光扩展到整个多边形（图16-21a）。将此与图16-21中所示的将线性插值得到的法向量用于计算每个像素的$\cos^n\alpha$项所得到从高光亮度处陡然下落相比较（图16-21b）。而且，如果高光不能落在一个顶点上，那么Gouraud明暗处理可能完全遗漏它（图16-21c），因为没有一个内部顶点可能会比它所插值的顶点更亮。相反，Phong明暗处理允许高光落在多边形内部（图16-21d）。比较彩图II-31和II-32中球上的高光。



a) b) c) d)

图16-21 使用Gouraud明暗处理和Phong明暗处理的镜面反射光照模型。高光落在左顶点：a)Gouraud明暗处理，b)Phong明暗处理。高光落在多边形内部：c) Gouraud明暗处理，d)Phong明暗处理。（由哥伦比亚大学David Kurlander提供。）

即使采用不考虑镜面反射的光照模型，因为在每点处都使用法向的一个近似值，所以一般法向插值所得到结果优于亮度插值所得的结果。这减轻了大多数情况下的马赫带效应，但却大大地增加了明暗处理的耗费，因为在光照模型计算时必须对每次插值得到的法向量进行规格化。Duff[DUFF79]提出结合差分方程和表查找来加速这种计算。Bishop和Weimer[BISH86]提出利用泰勒级数展开来对Phong明暗处理进行近似，该方法大大增加了明暗的计算速度。

另一种明暗处理技术，其复杂度介于Gouraud明暗处理和Phong明暗处理之间，涉及光照模型中使用的点积运算的线性插值。就如在 Phong明暗处理中一样，在每个像素点处计算光照模型，插值点积可以用来避免方向向量计算和规格化的耗费。当和镜面反射光照模型一起使用时，因为单独计算镜面项，并具有按幂律而非线性的衰减，这种技术可以产生比Gouraud明暗处理技术更满意的效果。但就像Gouraud明暗处理一样，如果高光不是落在某个顶点上高光可能被遗漏，因为任何用插值点积计算出来的亮度值都不可能超出跨度两端处点积计算出来的亮度值。

16.2.6 插值明暗处理中的问题

所有的这些插值明暗处理模型都存在许多共同的问题。这里我们罗列了其中的几种。

1. 多边形轮廓

不论一个插值明暗处理模型可以为曲面表面的真实明暗提供多么好的近似，网格的轮廓边表明其仍是多边形的。我们可以通过将表面分成更多的更小多边形来改善这种情况，但相应地也增加了耗费。

2. 透视变形

由于插值计算是透视变化后在三维屏幕坐标系统中进行的，而不是在世界坐标系中完成的，因此产生了变形现象。比如，线性插值使得图16-19中的明暗信息沿着每条边以常值从一条扫描线到另一条扫描线递增。考虑当顶点1比顶点2更远时会发生什么情况。透视缩小意味着扫描线间没有转换的z值的差别将沿着多边形边更远的方向增加。因此，如果$y_s = (y_1 + y_2)/2$，那么$I_s = (I_1 + I_2)/2$，但z_s将不会等于$(z_1 + z_2)/2$。这个问题可以通过采用大量更小的多边形来减轻。减小多边形的大小相应地增加了原来应插值而现在进行采样的点的数量，因此增加了明暗处理的准确性。

738 ↓ 739

3. 方向的依赖性

插值明暗处理模型的结果并不独立于多边形的投影方向。因为在顶点之间和水平扫描线上进行插值，所以，当多边形被旋转时，结果可能不同（见图16-22）。当动画相邻帧之间方向缓慢改变时，这个现象相当明显。同样的问题也可能发生在可见面判定中，因为各点的z值是通过各顶点所赋予的z值插值得到的。两个问题都可以通过将多边形分解为三角形来解决（见习题16.2）。此外，Duff[DUFF79]提出了一种插值方法解决了此问题，该方法不需分解且独立于旋转方向，但耗费高昂。

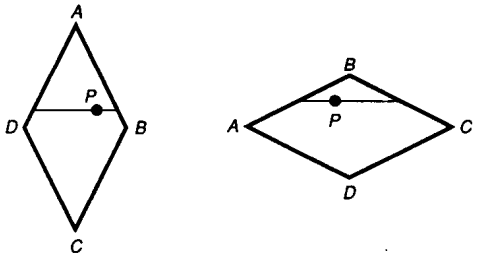


图16-22 在同一多边形不同方向插值推导的点P的不同值。a)中P由点A、B和D插值得到；b)中P由点A、B和C插值得到

4. 在共用顶点处的问题

当两个相邻的多边形并不共用一个落在公共边上的顶点时，明暗处理不连续问题就会发生。考虑图16-23中的三个多边形，其中，顶点C被右边的两个多边形共享，但却不被左边的大多边形所共用。由右边的大多边形直接决定的C点处的明暗信息通常不同于由左边多边形的顶点A、B插值所得C点处的明暗信息。结果，明暗处理就出现了不连续。这个不连续可以通过在左边的多边形插入一个额外的顶点来消除，该顶点共享C点的明暗信息。为消除这个问题，可以预先处理静态多边形库，另外一种方法是，如果在事先计算中对多边形（如用BSP树可见面计算）进行分割，那么可以做特别的记号在共用一条被分割的边的边上增加新的顶点。

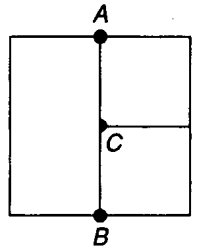


图16-23 顶点C被右边的两个多边形共用，但却不被左边的大多边形所共用

740

5. 非直接表示的顶点法线

所计算的顶点法线并不能充分代表了表面的几何性质。比如，如果我们通过共用一个顶点的所有面的法线的平均求取该顶点的法线，那么在图16-24中的所有顶点的法线将相互平行，如果光源很远，就会导致画面的明暗变化很小甚至没有变化。在顶点法线计算前进行此多边形进一步的细分可以解决这个问题。

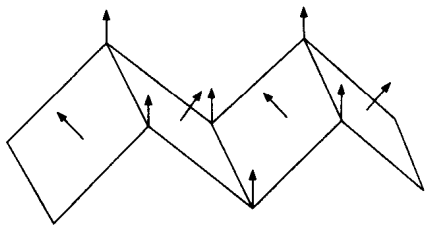


图16-24 计算顶点法线时的问题。所有顶点法线都是平行的

虽然这些问题激发了许多直接处理曲面表面的绘制算法的工作，但多边形却仍是最快且最易进行处理的，这些处理算法构成了大多数绘制系统的核心。

16.3 曲面细节

在平面或双三次曲面上加上我们至今所描述的任何明暗处理模型将产生光滑的均匀曲面——这与我们在现实生活中看到和感觉到的成鲜明对比。下面讨论已有的模拟这些遗漏曲面细节的各种不同算法。

16.3.1 曲面细节多边形

最简单的增加粗糙细节的方法是通过在基多边形上（比如建筑物的一边）覆加曲面细节多边形来展示一些特征（如门、窗、平面文字等）。每个曲面细节多边形与它的基多边形共面，并被标上记号以使之在可见面判定时不需与其他多边形进行比较。当为基多边形加上明暗色调时，在它覆盖的区域，曲面细节多边形和它们的材质优先级较高。

16.3.2 纹理映射

当细节越来越精细并且更复杂时，用多边形或其他几何图元进行直接造型不太实际。另外 741
一种方法是将数字化或合成的图像映射至物体表面。这项技术首先由Catmull[CATM74b]提出并由Blinn和Newell[BLIN76]进一步改进。这种方法称为纹理映射（texture mapping）或模式映射（pattern mapping）。图像称为纹理图（texture map），它的每个元素通常称为纹元（texel）。这个长方形的纹理图落在它自己的纹理坐标空间。此外，纹理可以由一个过程定义。彩图II-35给出了一些用图16-25中的纹理进行纹理映射的例子。在绘制的每个像素中，所选的纹元要么替代该表面的某个材质属性，比如漫射颜色分量，要么对该属性进行去除。一个像素经常被多个纹元所覆盖。为避免走样，必须考虑所有相关的纹元。

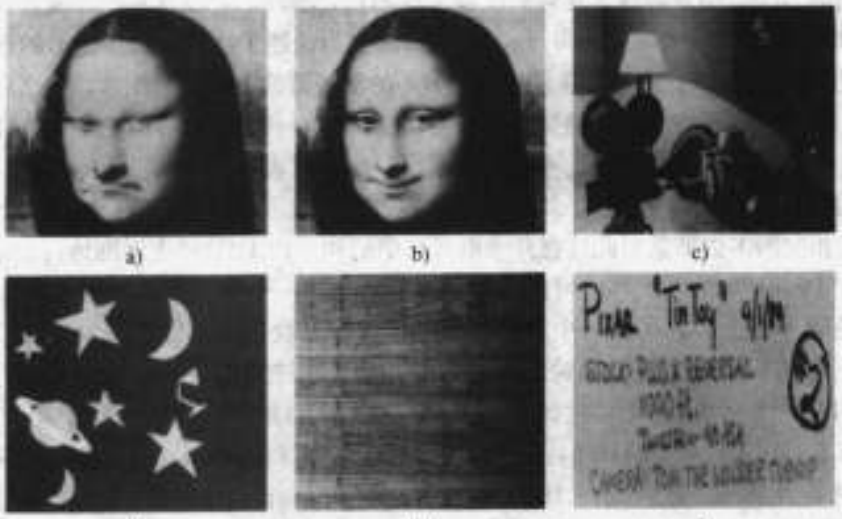


图16-25 用于生成彩图II-35的纹理。a)皱眉的蒙娜丽莎，b)微笑的蒙娜丽莎，c)油画，d)魔帽，e)地板，f)电影邮票。（Pixar公司1990版权，由Thomas Williams和H.B.Siegel用Pixar公司的PhotoRealistic RenderMan软件绘制的图像。）

如图16-26所示，纹理映射可以分为两步完成。简单的方法开始首先将像素的四角点映射到表面上。对一个双三次曲面片，这个映射自然在该表面的（s, t）参数坐标空间中定义了一组点。下一步，在表面参数坐标系下的四个角点被映射到纹理的（u, v）坐标空间。在此纹理图像的四个（u, v）点定义了一个四边形，该四边形在此像素映射的区域近似模拟了原本由表面斜率所描述的更复杂的形状。我们通过落在此四边形内的所有纹元的加权和来计算该像素的值，并以落在此四边形内的纹元所占该像素面积比例加权。如果被转换到（u, v）空间中的点落在纹理图外部，那么就像2.1节的图案一样，可以认为纹理图是可以复制的。在（s, t）与 742
（u, v）之间并非总是使用单一的映射，我们可以定义从0到1的（s, t）空间四边形的四个角与（u, v）空间四边形的对应关系。当一个表面为多边形时，通常将其纹理坐标直接赋予它的顶点。

既然在一个任意多边形内部的线性插值所得到的值是依赖于方向的，那么多边形可以首先划分为三角形。然而，即使在三角剖分后，线性插值在透视投影情形下仍然引起变形。这种变形比其他明暗的线性插值所引起的变形更引人注目，因为纹理特征将不能正确地透视缩小。我们可以通过将多边形划分为更小的多边形来获得这个问题的一个近似解，或者以更大的耗费在插值时进行透视分割来获得此问题的精确解。



图16-26 从像素到物体表面、纹理图的纹理映射过程

刚才介绍的方法采用方形像素几何形状和简单的盒式滤波。它没有考虑到像素仅仅映射到部分物体表面这种情况。Feibush、Levoy和Cook[FEIB80]为论述了纹理映射多边形的这些问题。把图16-26中的方形像素区域看成是以像素为作用中心的滤波器的支集的包围盒。那么可以将像素或纹元看成点。实际上，选中的纹元是那些落在要变换的包围盒和多边形的交集中，并将其转换到包围盒的坐标系统中。所转换得到的坐标用于索引一个滤波表以决定该纹元的权值，然后计算所有纹元亮度的加权平均。这个加权平均值又以该多边形对该像素亮度值的贡献进行加权。对每个投影与该像素相交的多边形重复此过程并求取所有值的和。17.4.2节将更详细地讨论这个算法。

Feibush、Levoy和Cook的算法可能非常低效。考虑将一个黑白相间的方格图案模板映射到一个无穷的地表平面上。为了给一个很远的地表像素贴上纹理，不得不对大量的纹元进行加权求和。这个问题的一个解决方法是对纹理图像进行前置滤波并以一种空间上高效的方式将结果存储起来，以便快速决定映射至该像素的所有纹元的加权平均。在17.4.3节中将讨论
743 Williams[WILL83]、Crow[CROW84]、Glassner[GLAS86]和Heckbert[HECK86a]设计的采用这种解决方法的各种算法。在习题17.10中讨论Catmull和Smith[CATM80]的直接映射整个纹理图至一物体表面的高效方法。Heckbert[HECK86]提供了一个所有纹理映射方法的综述。

16.3.3 凹凸映射

纹理映射虽然能影响表面的明暗色调，但物体表面仍表现为几何平滑的。如果纹理图是一个粗糙表面的照片，那么加上明暗色调的表面看起来不是很对，因为在生成此纹理图时的光源方向一般不同于照射该表面的光源方向。Blinn [BLIN78b]提出了一种无须显式几何造型就能显示出凹凸不平的表面几何的方法。就像物体表面的细微变化引起物体表面法向扰动一样，该方法在物体表面法向用在光照模型计算前对其进行扰动。这种方法称为凹凸映射，是基于纹理映射的一技种术。

一个凹凸映射是一组位移量，每个值被用来表示一点比实际表面位置或高或低的位移量。用一向量$\overline{P}$代表表面上一点，这里$\overline{P}=[x(s,t),y(s,t),z(s,t)]$，我们把$\overline{P}$点处关于物体表面参数化坐标轴$s$、$t$的偏导数，分别称为$\overline{P}_s$和$\overline{P}_t$。因为每个偏导数都与该表面相切，它们的叉积形成了该表面在$\overline{P}$点的表面法向（没有规格化）。因此，

$$\overline{N}=\overline{P}_s\times\overline{P}_t \tag{16-21}$$

可以通过在它的规格化法向上增加一个凹凸映射值B来偏移点P，这个新的点是

$$\overline{P}' = \overline{P} + \frac{B\overline{N}}{|\overline{N}|} \tag{16-22}$$

Blinn给出了对此新法向量（没有规格化）的一个较好的近似公式：

$$\overline{N}' = \overline{N} + \frac{B_u(\overline{N} \times \overline{P}_t) - B_v(\overline{N} \times \overline{P}_s)}{|\overline{N}|} \tag{16-23}$$

其中B_u和B_v是所选的凹凸映射值相对于凹凸图参数坐标u和v的偏导数。然后规格化$\overline{N}'$并在光照模型中代替表面法向值进行光照计算。注意，式(16-23)用的仅仅只是凹凸映射值的偏导数值，而不是其值。可以用双线性插值来推导指定（u, v）位置的凹凸映射值，而用有限差分计算B_u和B_v。

凹凸映射的效果非常逼真，观察者往往注意不到物体的纹理并没有影响它的轮廓边。彩图III-3和III-4给出了两个凹凸映射的例子。不像纹理映射，凹凸映射所引起的走样问题不能用滤波技术加以处理，因为这些值并不线性地对应于亮度值。对凹凸映射进行滤波只会平滑凹凸起伏。相反，可以计算子像素的亮度值并对每个像素进行滤波，或者在凹凸映射上进行一些前置滤波来改善粗糙走样问题。 744

16.3.4 其他方法

虽然二维映射在大多数情况下是有效的，但它往往产生不了令人信服的结果。当映射至曲面时，纹理经常暴露出其二维本质，并且在纹理拼接处出现许多问题。比如，当一个木纹纹理被映射至曲面物体的表面时，物体看起来似乎是被画上该纹理。为了正确绘制如彩图IV-21所示的用木头或大理石雕刻出来的物体，Peachey [PEAC85]和Perlin[PERL85]对实体纹理进行了研究。在20.8.3节将描述的这种方法中，纹理是物体上一点位置的一个三维函数。

其他表面性质也可以被映射。比如，在20.8.2节描述的Gardner[GARD84]利用透明映射从相当简单的形状产生了令人印象深刻的树和云。彩图IV-24展示了将复杂的函数式透明纹理运用到一组由二次曲面组成的物体上的效果。Cook已经实现了位移映射[COOK84a]，在这种方法中，真正的物体表面被偏移，而不仅仅只是物体的表面法线被偏移。该过程必须在可见面判定之前完成。彩图II-36即用该方法修饰的锥体和圆环的表面。在20.3节中我们将讨论用分形从简单的几何模型中产生丰富的几何细节。

至此，我们做了一个默认的假设，即对物体上一点的明暗处理过程不受该物体其他部分或其他任何物体的影响。但是实际中一个物体可能被落在它与光源之间的物体所遮蔽；可以透射光，以允许透过它看到另一个物体；或者可以反射其他物体，允许其他物体因为它而被看见。在下面的几节中，我们讨论如何模拟这些效果。

16.4 阴影

可见面算法判定从视点处可见哪些面，而阴影算法判定从光源处可见哪些面。因此，可见面判定算法和阴影算法本质上是一致的。从光源处可见的面不处在该光源的阴影区域中，而从此光源不可见的景物表面则处在该光源阴影区域中。当有多个光源时，必须相对于每个光源对表面进行分类。

这里我们考虑点光源的阴影算法。扩充光源将分别在16.8节、16.12节、16.13节中加以讨论。从一点光源处的可见性就像从视点处的可见性一样，要么完全可见要么完全不可见。当从光源处看不见面上一点时，该点的光照计算必须加以修正以考虑这种情形。在光照方程中加入阴影效果，则方程变为

$$I_s = I_{as}k_aO_{d\lambda} + \sum_{1\le i\le m} S_i f_{atti} I_{pi\lambda}\,[k_dO_{d\lambda}(\bar{N}\cdot\bar{L}_i) + k_sO_{s\lambda}\,(\bar{R}_i\cdot\bar{V})^n] \tag{16-24}$$

其中：

$$S_i = \begin{cases} 0, \text{如果在此点光被遮挡} \\ 1, \text{如果在此点光没有被遮挡} \end{cases}$$

745

注意，所有处在点光源阴影处的区域仍为环境光所照射。

虽然计算阴影要求从点光源处计算可见性，但正如我们所指出的，如果不进行任何可见性测试，有可能产生“虚假”的阴影。这可以通过从点光源将所有物体变换为到一预定平面上的多边形投影来高效生成，不必对被变换的多边形所遮挡的表面进行裁剪，也不需检测该变换多边形是否被中间的多边形所遮挡[BLIN88]。这些阴影被用作为曲面细节多边形。但对一般情形，这些“虚假”阴影是不够的，其他许多阴影生成方法也是可能的。我们可以首先进行所有阴影处理，将其与可见性处理进行多种方式的交错处理，或者甚至在可见面处理完成后就进行阴影处理。这里我们看看按照这些方法的一些不同算法，这些算法建立在[CROW 77a]中对阴影算法的分类上。为简化解释，如果不做另外的定义，我们认为所有的物体都是多边形的。

16.4.1 扫描线生成阴影算法

最早的阴影生成方法是将扫描线算法加以改进，交错进行阴影和可见面的处理[APPEL68；BOUK70b]。以光源为投影中心，将可能投下阴影的多边形的边投影到与当前扫描线相交的多边形上。当扫描扫过这些阴影边中的一条时，相应地修改该像素点的颜色。

这种算法最简单的实施方法必须为多边形进行$n(n-1)$次到其他多边形的投影计算。Bouknight和Kelley [BOUK70b]代之以一个灵活的预处理步骤。在该步骤中，所有多边形被投影到一个以光源为中心的球面上。两个范围不相重叠的投影可以删除，可以鉴别其他一些特殊情况来限制算法余下部分必须考虑的多边形对的数目。然后，算法将所有这些投影以光源为投影中心向它可能会投下阴影的多边形平面进行投影，如图16-27所示。每个这样的阴影多边形投影都有关于投下阴影或有可能接受这些阴影的多边形的信息。当一个扫描线算法的规则扫描跟踪记录下正扫过的那个规则多边形边时，另外一个单独并行的阴影扫描跟踪记录下当前扫描线正扫过的那个阴影多边形的投影边，因此当前阴影扫描将跟踪记录下所处的那个阴影多边形投影。当计算一个跨度的明暗色调时，如果阴影扫描正处在任何一个投影到该多边形平面的阴影投影中，那么它处在阴影中。因此，在图16-27的a中的bc段在阴影中，而ab、cd段不在。注意，算法不需要以计算阴影的多边形为窗口对阴影多边形的投影进行裁剪。

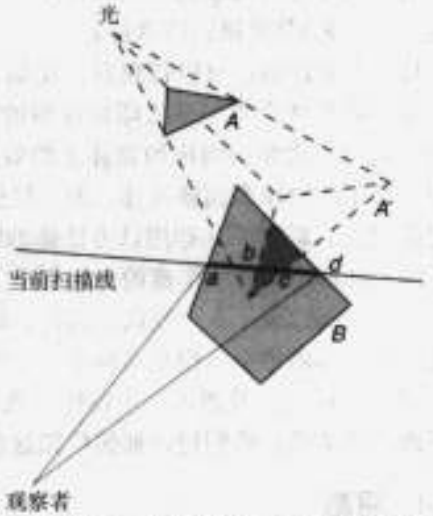


图16-27 使用Bouknight和Kelley方法的扫描线阴影算法。多边形A在平面B上投下阴影A'

16.4.2 对象精确的两步法阴影算法

Atherton、Weiler和Greerberg提出了一种在可见面判定前进行阴影判定的算法[ATHE78]。他们用同样的算法对物体描述进行两次处理，一次从视点，一次从光源。两次处理的结果结合起来判定为该光源所照射的多边形可见部分并扫描转换整个场景。既然阴影是不依赖于视点的，

因此只要光源和物体固定不动，那么对不同视点下观察的同一物体的不同图像，所有的阴影计算只需要进行一次。

如在16.28节中所示，该算法首先用15.7.2节中所讨论的Weiler-Atherton可见面算法决定从光源处可见的面。这一遍的输出结果是一列被照亮的多边形。每个多边形被标上它的父多边形的标识。所有物体必须全部装入光源视见体中，因为没有装入的物体部分被视为不被光所照射。如果一个光源视见体内不能包括所有物体，可以建立多个不相重叠的从光源发出的视见体。这种技术称为分区（sectoring）。

接着，这些被照亮的多边形被转换回模型坐标并以曲面细节多边形（16.3节）的形式与原数据库的一个副本合并，产生一个如图16-29所示的不依赖于视点的合并数据库。注意，图16-28所示的实现在合并两个数据库前对它们做了同样的转换。然后就可以使用Weiler-Atherton算法从任意观察者的视点在此合并的数据库中进行不可见面消除。到目前为止所做的所有处理都是以对象精度进行的，并且产生一个多边形链表。然后用多边形扫描转换算法来绘制图像。被曲面细节多边形所覆盖的可见多边形绘制为被该光源所照亮的，而没有被覆盖的可见多边形绘制为在阴影中。彩图III-5即是用这种方法产生的。多光源可以通过为每个新的光源以该新光源为视点处理得到的合并数据库并合并所有结果来解决。

746
747

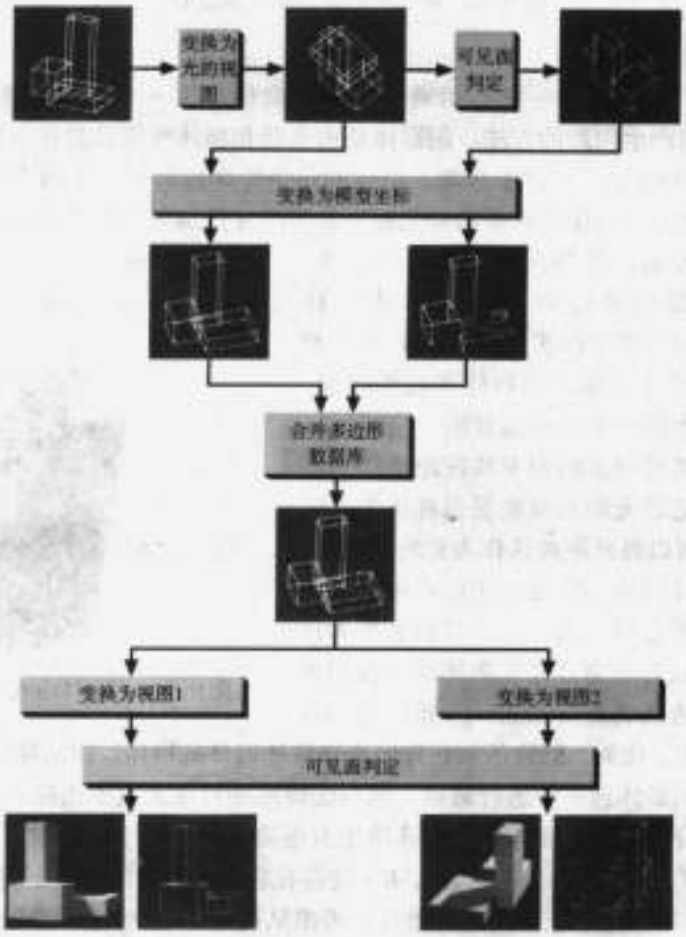


图16-28 Atherton, Weiler和Greenberg算法中，阴影的生成和显示。（图像来源于Peter Atherton, Kevin Weiler, Donald P. Greenberg，康奈尔大学计算机图形学项目，1978。）

748

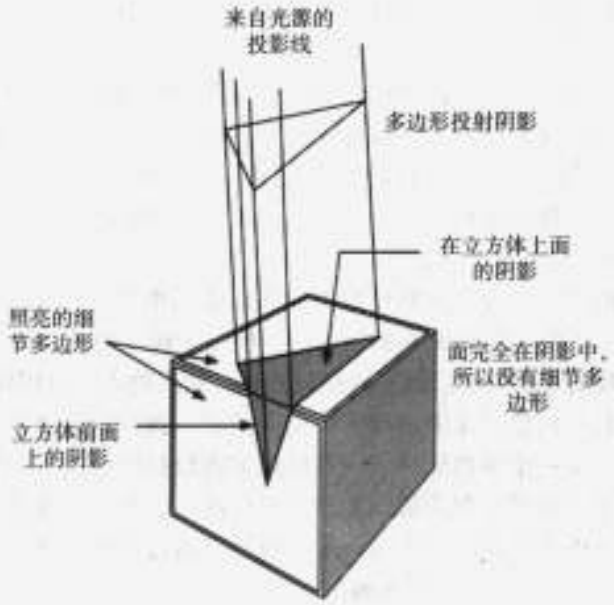


图16-29 被照亮的曲面细节多边形

16.4.3 阴影体

Crow[CROW77a]描述了一种如何通过为每个物体产生一个该物体遮挡光源的阴影体（shadow volume）来产生阴影的方法。阴影体是由光源和物体所定义的并由看不见的阴影多边形所围成。如图16-30所示，相对于光源，物体的每条轮廓边都有一个四边形阴影多边形。该阴影多边形的三个侧面由物体的轮廓边和光源所发出的穿过那条轮廓边的两端点的两条线所决定。每个阴影多边形有一个指向阴影体外的法线。只有那些面对光源的多边形产生阴影体。在Bergeron[BERG86a]所描述的实现中，阴影体（和它的每个阴影多边形）一端由原物体多边形封顶，而另一端由该多边形的一个通过缩放的、法线反向的副本封底。这个通过缩放的副本放在距光的一定距离处，在该距离之后光的衰减能量强度认为是可以忽略不计。我们可以将此距离认作为光的影响球。在此影响球之外的任何点实际处于阴影点中而且不需要任何额外的阴影处理。实质上，没有必要为任何完全处在影响球以外的物体产生阴影体。我们通过考虑影响区域的方法将此方法进一步推广运用在非均匀发射的光源上，比如，删除落在光源挡板和锥体以外的物体。如果视见体预先已知的话，还可以用视见体对阴影体进一步进行裁剪。该算法将那些封顶多边形也视为阴影多边形。

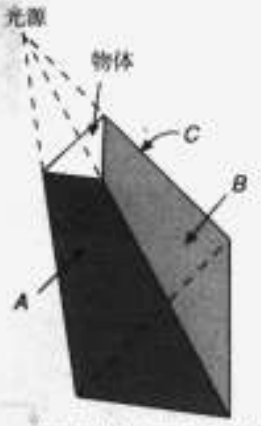


图16-30 阴影体由光源和物体所定义

阴影多边形本身并不被绘制，但却用来决定其他物体是否处于阴影中。相对于观察者，一个正面阴影多边形（图16-30中的多边形*A*、*B*）使得在它之后的物体被投上阴影；一个背面阴影多边形（多边形*C*）则忽略了正面的这种效应。考虑从视点*V*到物体上一点的向量。如果与该向量与阴影体相交的正面多于背面，该点处于阴影中。因此，在图16-31a中的点*A*和*C*处于阴影中。这是惟一的一种当*V*不在阴影中时点在阴影中的情形。因此点*B*被照亮。如果*V*处在阴影中，则存

在另外一种点在阴影中的情形，即从视点到物体上一点的向量没有遇到遮挡该视点的物体多边形的相应背面阴影多边形。因此，图16-31b中的点A、B、C皆处在阴影中，即使从V到B的向量如它在图16-31a中所经历的一样与相同数量的正面阴影边界多边形与背面阴影边界多边形相交。

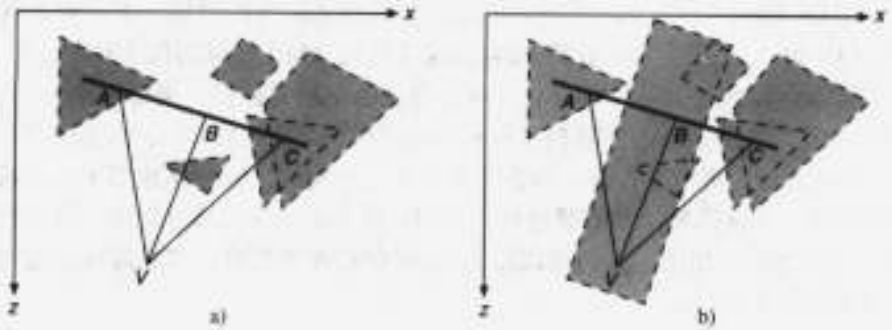


图16-31 为位于V的观察者决定一点是否在阴影中。虚线表示阴影体（加上阴影）。a)V不在阴影中，点A和C在阴影中，但点B被照亮；b)V在阴影中，点A、B和C都在阴影中

749
↓
750

我们可以通过为每个正面（相对于观察者）阴影多边形赋值为+1，而背面阴影多边形为-1来计算一个点是否处于阴影。设置一个计数器，该计数器的初始值设置为包含当前视点的阴影体的数目，并且按视点与物体上的点之间的所有阴影多边形的相关值递增。如果该点的计数器的值为正，则该点处于阴影中。对每个视点，只需要计算一次包含该视点的阴影体的数目，通常取由视点到无穷远处的一个任意投射线所交的所有阴影多边形的值的和的负数。

虽然可以为每个多边形建立一个阴影体，我们可以利用对象相关性为相互连接的多面体计算一个阴影体。这可以通过只从那些相对于光源为轮廓边的边来产生阴影多边形来实现。它们是相对于光源的轮廓边（如15.3.2节中所定义）。

多个光源可以通过为每个光源单独建立阴影体来处理，为阴影体的每个阴影边界多边形标上光源标识并为每个光源保持一个不同的计数器。Brotman和Badler[BROT84]实现了z缓存的阴影体算法，Bergeron[BERG86a]讨论了有效地处理了包括非平面多边形的任意多面体物体的扫描线阴影体算法的实现。

Chin和Feiner[CHIN89]描述了一种对象精确的算法。该算法用12.6.4节中讨论的BSP树实体造型表示为多边形环境建立单一的阴影体。以相对于光源以从前向后的顺序处理各多边形，将每个面向光源的多边形沿树作过滤，将多边形划分为由光源直接照射的片段和阴影区片段。只有那些由光源直接照射的部分可以投下阴影，因此，由光源和每个由光源直接照射的片段所形成的半无穷锥体并入到阴影体中。因为以从前向后的次序进行处理，从而保证了当前所处理的多边形不会落在光源和已处理过的多边形之间。所以，既然任何多边形都不需要与已处理的多边形的平面进行比较，那么多边形本身就不必加入到该阴影体了。当与Atherton-Weiler -Greenberg算法一起使用，由光源直接照射的片段可以以曲面细节多边形的方式加入环境中，或者与处于阴影中的片段一起显示。可以通过将一个阴影体的BSP树的多边形片段沿下一个光源的阴影体的BSP树过滤来处理多个光源。每个片段标上记号以表明哪个光源照射它，从而允许使用任何多边形可见面算法来显示所得到的多边形片段场景。因为这种阴影体表示方式，光源可以放在相对于物体的任何位置上。因此没有必要分区。在[CHIN90]中讨论了该方法的几种优化算法和并行算法。彩图III-6a即是用该算法绘制的，彩图III-6b显示了将多边形沿阴影体BSP树过滤后所产生的多边形片段。

16.4.4 两遍z缓存阴影算法

Williams[WILL78] 提出了一种基于两遍实施z缓存算法的阴影生成方法，一遍从观察者方

751 向，一遍从光源方向。不同与16.4.2节中的两遍算法，Williams通过使用一个图像精确的计算决定一个表面是否在阴影中。图16-32a给出了由L处光源照射下的环境的总体视图。图16-32d给出的是从视点V看到的没有任何阴影效果的图像。该算法首先计算并存储以光源作为视点所看到的图像（图16-32b）。在图16-32b中，亮度的增加代表距离的增加。下一步，从观察者视点处使用做了以下修改的z缓存算法计算z缓存图像（图16-32c）和该图像（16-32e）。无论什么时候，当一个像素点被判定为可见时，它在观察者视图中的对象精度坐标（x_0, y_0, z_0）转换为光源视图中的坐标（x_0', y_0', z_0'）。该转换坐标x_0'和y_0'被用来选择光源z缓存中的z值并与该转换z_0'值进行比较。如果z_L比z_0'更接近光源，那么其间存在一个物体，该物体遮挡了该点从光源处来的光。该像素被绘上了在阴影处的明暗色调；否则，从光源处可以看见该点，该点被绘上为照亮的明暗值。与纹理映射相仿，我们可以将光源z缓存看成别彩图。多光源可以通过为每个光源使用不同的阴影图来处理。

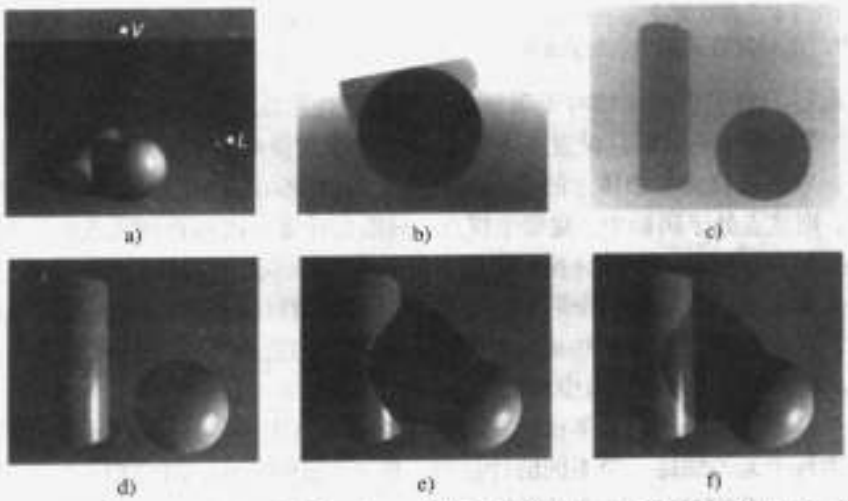


图16-32 z缓存阴影生成方法。a)场景概况，b)光源z缓存图像，c) 观察者z缓存图像，d)观察者视图图像，e)带阴影的观察者视图图像，f)带有后处理阴影的观察者视图图像。（由哥伦比亚大学David Kurlander提供。）

像一般的z缓存可见面算法一样，这个算法要求每个被绘制的像素都应进行明暗处理。这意味着必须为每个这样的像素进行阴影计算，即使它最终被更近的物体所覆盖。Williams建议对他的算法进行修改，以充分利用z缓存算法与可见面判定算法相结合的简易性，消除那些被遮挡物体的阴影计算。该修改算法首先不仅仅计算了阴影图，而是同时计算了从观察者的角度所看到的一般明暗图像（图16-32d）和观察者的z缓存（所有这些计算使用传统的基于z缓存的
752 硬件）。然后通过一个后处理过程来增加阴影，产生图16-32f，该后处理过程的复杂度与图像的像素数目成线性关系。对观察者视图图像中的每一个像素进行与前面同样的变换和比较。如果z_L比z_0'更接近光源，那么早先计算的观察者视图图像中该像素的明暗色调被减暗。虽然该方法比原方法更有效，但它产生了一些不自然效果。最明显的是，处在阴影处的物体将有（暗的）镜面高光，即使光源处来的光被遮挡使得该物体不应有任何镜面高光。另外，在原来的算法中被转换的z_0值是对象精确的，但这里却是较低的z缓存精度。（见习题16.15。）

不同于到目前为止所讨论的其他的阴影算法，Williams的算法对于那些可以扫描转换的物体（包括曲面）的阴影生成变得非常简单。但是因为所有的操作都以图像精度产生，必须对有限的数值精确度做一定的估量。比如，从z_0到z_0'的转换应将z_0'移到光源更近处，以避免可见点自己给

自己投上阴影。像z缓存可见面算法一样，该阴影算法容易出现走样问题。Reeves、Salesin和Cook[REEV87]给出了用百分比比率更接近的滤波做改进。每个z_0'和在阴影图中的一定区域的值进行比较，更近z_0'值所占百分值决定阴影的多少。这个改进的算法被用来绘制彩图D、F和II-36。图16-33是用来产生彩图II-36的阴影图。



图16-33 用于创建彩图II-36的阴影图。（Pixar公司1990版权，阴影图由Thomas Williams和H. B. Siegel使用Pixar公司的PhotoRealistic RenderMan软件绘制。）

16.4.5 全局光照阴影算法

在复杂环境下光线跟踪和辐射度算法给出了一些非常好的具有阴影效果的图片。简单的光线跟踪用于模拟点光源产生的阴影效果，而更先进的算法则允许扩展的光源类型。以上内容将在16.12节讨论，而在16.13节讨论的辐射度算法把光源看成和其他表面具有相同几何性质的发光面。因此，这些方法隐式地支持扩展光源。 753

16.5 透明性

正如物体表面会产生镜面反射或漫反射那样，透明或者半透明的物体则可以透光，我们可以透过诸如玻璃这类透明材质观看，尽管大多数光线已被折射（弯曲），而对于像磨砂玻璃的半透明材质则产生了漫射透射。由于半透明物体表面以及内部的不规则性，穿过该半透明材质的光线将会变得杂乱无章，这样透过半透明材质所看到的物体必然显得模模糊糊。

16.5.1 无折射的透明性

模拟透明性的最简单方法是忽略折射，这样一来，光线在穿越物体表面时不再被弯曲。因此，穿越透明物体视线上所见的物体也必然在几何意义上落在该视线上。虽然无折射的透明性并不存在，但它通常产生比折射更为有用的效果。例如，如彩图III-7所示，它可以体现一种透过物体表面后无扭曲的视图。正如我们以前曾经提到的那样，完全的照片真实感并不总是我们生成画面的目标。

通常有两种方法来近似模拟透过一个物体看见另一个物体时发生的物体颜色的混合的方法。我们分别称之为插值透明法和滤光透明法。

1. 插值透明法

如图16-34所示，考虑在不透明的多边形2以及观察者中间存在一个透明的多边形1时会发生什么。插值透明法通过对两个多边形各自的计算所得的明暗值进行线性插值来决定位于两个多边形重叠部分的像素的明暗值：

$$I_\lambda = (1 - k_{t1})I_{\lambda 1} + k_{t1}I_{\lambda 2} \tag{16-25}$$

透射系数k_{t1}用来度量多边形1的透明度，取值为0到1。当k_{t1}为0时，多边形1变成了不透明的，不再透过光线；而当k_{t1}取值为1时，多边形1便成了完全透明的，对I_λ值不再有任何影响：（$1 - k_{t1}$）值称为多边形的不透明度。插值法可以理解为将一个多边形模拟为一个非常精细的不透明材料构成的网格。透过这个网格可以看到其他物体：k_{t1}值是透过该网格能看到其他物体的比例。这样处理的一个完全透明的多边形将不会有任何镜面反射。为了达到更真实的效果，我们可以首先只对多边形1的环境光和漫射光部分以及多边形2的全部的明暗色调进行插值，然后再加入多边形1的镜面反射光部分[KAY79b]。

另一种方法通常称为筛子透明法。它通过只绘制那些与透明物体的投影相联系的像素来生

成网格。像素的（x, y）地址的低位被用于索引，指向一个透明位掩码。如果这个索引位值为1，则写出该像素值。否则，该像素被挂起，下一个最接近的物体成为可见。掩码中1的位数越少，则该物体越透明。这种方法依靠我们的眼睛进行空间合成来产生插值透明度值。但是请注意，一个物体将屏蔽掉其他任何一个使用相同透明度掩码的物体，而且很难避免不同掩码之间的不恰当的相互作用。

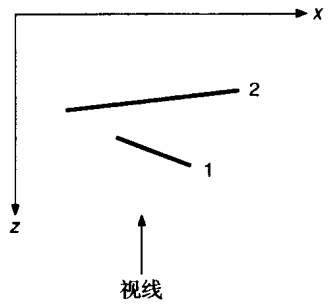


图16-34 两个多边形的截面图

2. 滤光透明法

滤光透明法将多边形看成是一个有选择地允许一定波长光线通过的透明滤镜，模型可表示为

$$I_\lambda = I_{\lambda 1} + k_{t1} O_{t\lambda} I_{\lambda 2} \tag{16-26}$$

这里$O_{t\lambda}$是多边形1的透明度颜色。可以通过给不同的波长λ选择不同的$O_{t\lambda}$值模拟一个颜色滤镜。无论在插值透明法还是在滤光透明法，若在这些多边形前还有别的透明多边形，那么运算将从后向前递归地执行，并把上一次计算所得的I_λ作为当前的$I_{\lambda 2}$。

3. 实现透明效果

可以非常容易地将一些可见面算法加以修改加入透明效果。如扫描线算法和列表优先级算法。在列表优先级算法中，读入一个将被透明多边形覆盖的像素的颜色值，并且在扫描转换该多边形时，该值将用于光照明模型的计算中。

大部分基于缓存的系统支持筛子透明法，因为该算法允许透明物体和不透明物体混合在一起并以各种顺序进行绘制。但是要想把式(16-25)和式(16-26)所示的透明度引入到z缓存算法却比较困难，因为多边形是以随意的顺序绘制的。想像一下在先处理了几个叠加的透明多边形后，接着是一个不透明的的多边形。我们当然希望将此多边形插入相应的透明多边形后。但是，在z缓存中并没有存储所需的信息来判定这些多边形的颜色值。一个简单的但却不正确的方法是：最后处理透明多边形，并将它们的颜色与帧缓存中已有的颜色值结合，但它们的相对深度并没有得到考虑。

Mammen[MAMM89]描述了一种通过多步绘制和辅助内存在缓存系统中以由远到近的顺序正确地绘制透明物体的方法。首先，对所有的非透明物体用传统的z缓存方法进行绘制。然后分别处理透明物体，将其写入一组不同的缓存中。该缓存不但存有该像素的颜色和z值，还存有该像素的透明度值和一个标志位。每个标志位初始化值为0，且z值初始化为最近值。如果像素位上透明物体的z值小于相应不透明物体的z缓存的值，但却大于透明物体的z缓存中的z值，那么该物体此像素点上的颜色、z值存入z缓存中且标志位设为1。在处理完所有的物体之后，透明物体的缓存中那些标志位为1的像素位保存了该像素上最远的透明物体的有关信息。这些信息便和原来在帧缓存以及z缓存中的信息相结合。已标志的像素位的透明物体z缓存的值取代非透明物体的z缓存中的值，标志位被复位（置“0”）。不断地重复上述过程，从后向前绘制不断靠近的物体。彩图III-7便是采用这种算法实现的。

754
~
755

Kay 和Greenberg[KAY79b] 实现了一种模拟发生在薄曲面轮廓边界上的光的不断衰减现象的方法。在这些地方，光线通过更多的材质。它们将k_t定义为一个在透视投影变换后曲面法向的z分量值的非线性函数。

$$k_t = k_{t\min} + (k_{t\max} - k_{t\min})(1 - (1 - z_N)^m) \tag{16-27}$$

其中$k_{t\min}$和$k_{t\max}$是物体透明度的最小值和最大值，z_N是计算k_t的那个点的规格化的曲面法向量的z分量值，m是一个幂因子（通常为2或3）。m值越高，模拟的曲面越薄。这里算出的k_t可以作为

式(16-25)或式(16-26)中的k_{t1}。

16.5.2 折射透明性

折射透明性的模拟要比无折射的透明性的模拟困难得多，因为几何视线与光学视线不同。在图16-35所示的情况下，如果考虑折射，沿所示视线透过透明物体后可看见A；若忽略折射，则应该看见B。Snell法则给出了入射角θ_i和折射角θ_t的关系：

$$\frac{\sin\theta_i}{\sin\theta_t}=\frac{\eta_{t\lambda}}{\eta_{i\lambda}} \tag{16-28}$$

其中，$n_{i\lambda}$和$n_{t\lambda}$分别为光透过材料的折射率。一种材料的折射率是光线在真空中传播的速度与光在该材料中传播速度的比值，随光的波长甚至温度而变化。真空的折射率为1，空气的折射率则接近于1；所有其他材料有着更高的值。折射率依赖于波长。在许多折射实验中我们都可以明显地观察到这个事实，如我们所熟悉的但却很难模拟的色散（dispersion）现象，一种光被折射为光谱的现象[THOM86；MUSG89]。

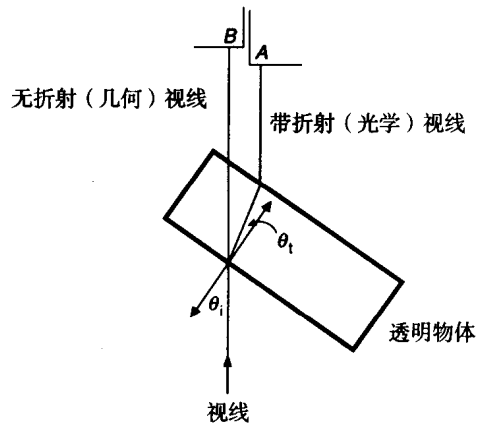


图16-35 折射

1. 折射向量的计算

可以由下式求得折射方向的单位向量$\bar{T}$：

$$\bar{T}=\sin\theta_t\,\bar{M}-\cos\theta_t\,\bar{N} \tag{16-29}$$

其中$\bar{M}$是在向量$\bar{I}$和$\bar{N}$所在平面上的垂直于$\bar{N}$的单位向量[HECK84]（图16-36）。回忆一下在16.1.4节中我们利用$\bar{S}$来计算折射向量$\bar{R}$，我们有$\bar{M}=(\bar{N}\cos\theta_i-\bar{I})/\sin\theta_i$。通过替换，

$$\bar{T}=\frac{\sin\theta_t}{\sin\theta_i}(\bar{N}\cos\theta_i-\bar{I})-\cos\theta_t\,\bar{N} \tag{16-30}$$

若令$\eta_{r\lambda}=\eta_{i\lambda}/\eta_{t\lambda}=\sin\theta_t/\sin\theta_i$。重排各项后有

$$\bar{T}=(\eta_{r\lambda}\cos\theta_i-\cos\theta_t)\,\bar{N}-\eta_{r\lambda}\,\bar{I} \tag{16-31}$$

注意$\cos\theta_i$是$\bar{N}\cdot\bar{I}$，$\cos\theta_t$可以由下式求得：

$$\cos\theta_t=\sqrt{1-\sin^2\theta_t}=\sqrt{1-\eta_{r\lambda}^2\sin^2\theta_i}=\sqrt{1-\eta_{r\lambda}^2(1-(\bar{N}\cdot\bar{I})^2)} \tag{16-32}$$

因此有

$$\bar{T}=\left(\eta_{r\lambda}(\bar{N}\cdot\bar{I})-\sqrt{1-\eta_{r\lambda}^2(1-(\bar{N}\cdot\bar{I})^2)}\right)\bar{N}-\eta_{r\lambda}\bar{I} \tag{16-33}$$

2. 全反射

当光线从一个高折射率介质进入一个低折射率介质时，光线透射角θ_t将大于入射角θ_i。如果θ_i足够大，以至于θ_t超过了90°，则光线则会在介质之间的交界面上反射回来，而不是进入另一个介质。这种现象称为全反射。发生全反射的θ_i的最小值称为临界角。可以很容易观察这种现象，如透过鱼缸观察您的手。当您的观察角大于临界角时，您所能看到的就仅仅只是那些紧贴在玻璃上的部分手掌，因为在这部分手掌和玻璃之间没有空气层（空气的折射率要小于玻璃和水）。当$\sin\theta_i$为1时，θ_i的值便是临界角度值。若在式(16-28)中将$\sin\theta_t$置为1，临界角即为$\sin^{-1}(\eta_{t\lambda}/\eta_{i\lambda})$。当式(16-33)中的那个平方根得到一个虚数时，全反射现象便发生了。

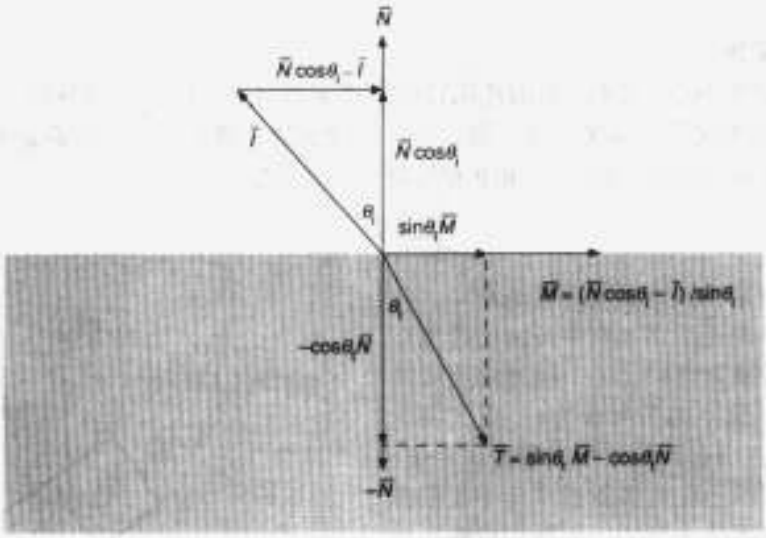


图16-36 计算折射向量

16.12节讨论了运用Snell法则和光线跟踪来模拟折射透明效果，半透明效果则在16.12.4节和16.13节中讨论。可以将一种对折射现象的近似方法结合到使用由远及近进行绘制处理的方法中[KAY79b]。

16.6 物体间的反射

当一个物体表面反射场景中任一表面所传来的光，就发生了物体间反射。从随观察者位置变化而变化的或强或弱的镜面反射（类似于镜面高光）到与观察者位置无关的漫反射，都有这种效果。无论是光线跟踪方法（16.12节）还是辐射度方法（16.13节）都产生了一些非常令人满意的效果，充分展示了物体间的镜面反射和漫反射效果；而且早期的一些技术也能产生这种吸引人的结果。

Blinn 和Newell[BLIN76]提出了反射映射（通常也称为环境映射）技术来模拟物体间的镜面反射。先选择一个投影中心，从该中心把被反射的周围环境投影到将反射到的包围该绘制物体的包围球的球面上。可以将被投影的环境看成一张二维纹理图。在物体上每一个要显示的点，反射图将通过以 $\bar{V}$ 关于 $\bar{N}$ 的映像得到的向量的极坐标来索引。反射图的x轴和y轴分别代表经度（从0°到360°）和纬度（从 – 90°到90°），如图16-37a所示。Hall[[HALL86] 做了修正，提出一种用y轴表示纬度的正弦值，这样球面上相同面积区域映射至反射图同样的面积区域。另外还有一种投影方法是，将周围环境投影到包围立方体的6个面上。这个立方体各边与世界坐标系坐标轴方向一致，这样规范化反射向量中的最大分量表明了所索引的相应面。图16-38a显示了立方体和立方体展开图之间的对应关系。图16-38b则是用于生成彩图II-37中的茶壶的反射图。和纹理映射一样，通过索引值周围一定数目反射图的值的过滤来实现反走样，以决定物体上给定一点的反射光。在图16-38b中使用了一个稍微大于90°的角为立方体的每个面的边界提供一定的余地，以避免在过滤时需要同时考虑两个或两个以上的面。

756 - 758

虽然反射映射可以用来产生一些有用的效果[GREE86]，但它仅仅提供了一个对正确的反射信息的近似值。如果仅仅考虑表面的反射方向，而没有考虑其在球体中的位置，则建立了一个无穷大的环境球。这个问题可以通过以下方法进行一定的改进，即通过表面所在位置来帮助

确定反射图的索引部分来建立一个有限大小的环境球。但在上述两种方法中，因为反射图仅仅考虑了在一个单独的点上的可见性关系，因此一个面若离创建图时所使用的投影中心越远，它就扭曲得越厉害。一种改进方法是建立多个反射图，每个都以一个主要物体为中心，并为所要映射的物体索引到最近的反射图。甚至可以通过一维反射图来获取一个简单而有效的反射效果。例如，$\bar{V}$的反射向量的y分量可以用来索引一个一维亮度数组，该亮度数组代表了从地面经地平线到天空的颜色的范围。

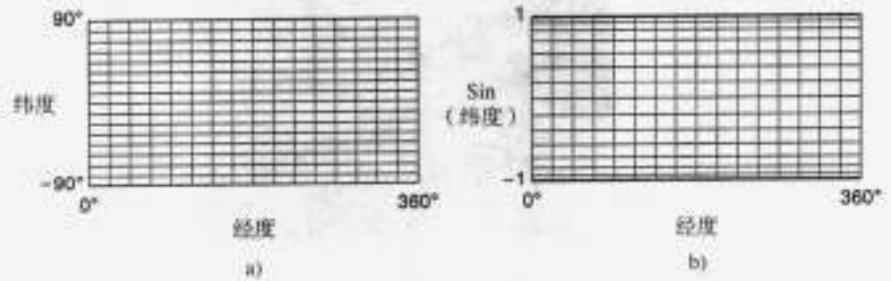


图16-37 反射图坐标系统。a)纬度-经度，b)正弦（纬度）-经度

对于反射映射而言，平面是困难所在，因为反射角变化非常缓慢。但是如果仅仅从一个视点来观察平坦表面的反射，那么可以使用另一种技术。这时，将视点对称反射到平面的另一边，然后从这个对称视点绘制场景的倒置图像，如图16-38c中的截面图所示。在该物体可见的任何地方，该图像都可和原来的图像合并。图16-38d便是采用了这种技术产生的，并用来绘制彩图II-37地板上的反射图案的。

759

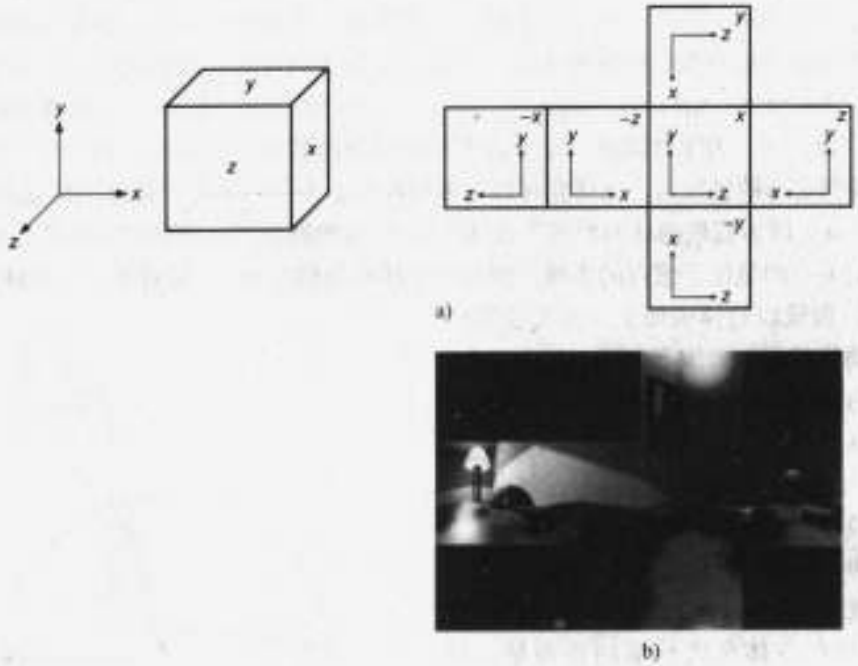


图16-38 反射图。a)立方体反射图轮廓图，b)彩图II-37中的茶壶的反射图，c)平面反射的几何特性，d)与彩图II-37中的地板合成的反射图像。（Pixar公司1990版权，b)和d)的图像由Thomas Williams和H. B. Siegel使用Pixar公司的PhotoRealistic RenderMan软件绘制。）



c)



d)

图16-38（续）

16.7 基于物理的光照模型

在此之前所讨论的光照模型大部分基于我们的常识，是图形学中的一些实际的方法。虽然，所用等式近似地描述了光线如何与物体相互作用，但是它们并没有一个物理基础。在本节里，我们将讨论一些基于物理的光照模型，其中一部分内容将基于Cook 和Torrance的工作[COOK82]。

到目前为止我们一直都在使用亮度（intensity）这个词，但却没有严格定义，并经常用它来非正式地描述一个光源的亮度、一个面上的一点甚至是一个像素的亮度。现在用热辐射学中的辐射计量术语来正式解释我们使用的术语。热辐射学是我们理解光线如何与物体相互作用的基础[NICO77；SPARR78；SIEG81；IES87]。首先，我们介绍光通量（flux），即单位时间内传播的光能，单位为瓦特。为了计算在一个方向上接收或发送的光通量的值，我们还需要引入立体角的概念，即圆锥体的锥顶角。立体角可以以锥顶在球心上的圆锥所截取球的球面面积来计量。单位立体角（sr）即为截取的球面积等于该球半径r平方的圆锥的立方体的立体角。如果球面上有一点，我们关心的是位于该点的半球。因为一个球面面积为$4\pi r^2$，则对于一个半球，有$4\pi r^2 / 2r^2 = 2\pi$个sr。假设以物体表面上一点为投影中心，将一个物体投影到以该点为球心的球面上，该物体所张的立体角ω便是用球面上该物体投影面积除以半球半径的平方，（这个除法消除了立体角对球面大小的依赖。）因此，为方便起见，我们经常把立体角以物体在单位球体或半球上的投影面积为计量单位，如图16-39所示。

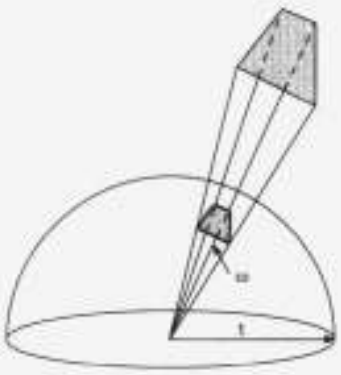


图16-39 一个物体对一表面上一点所张的立体角是该物体在位于此点的单位半球面上的投影所占的面积

辐射强度（radiant intensity）定义为一特定方向一个单位立体角内传播的光通量，以W / sr计量，当我们用亮度表示一个点光源时，通常是指它的辐射强度。

面辐射强度（radiance）是每一个单位透视

缩小曲面面积上的辐射强度，用W / (sr · m²)来度量。透视缩小（foreshorten）曲面面积也叫投影面面积，指的是该表面到垂直于辐射方向的平面上的投影。透视缩小曲面面积可以通过表面面积乘以$\cos\theta_r$求得，其中θ_r为辐射光线相对于表面法线的角。一个小的立体角$d\omega$可以通过物体的透视缩小曲面面积除以立体角所在顶点到物体的距离的平方来近似。以前当我们说到一个表面的亮度时，通常就是指它的面辐射强度。辐照度（irradiance），通常又称为光通量密度，是单位非透视缩小曲面面积上所入射的光通量，用W/m²来度量。

在图形学中，我们关心表面入射光与该表面反射光和透射光的关系。考虑图16-40，入射光的辐照度为

$$E_i = I_i(\bar{N} \cdot \bar{L})\, d\omega_i \tag{16-34}$$

其中I_i为入射光的面辐射强度，$\bar{N} \cdot \bar{L}$为$\cos\theta_i$。因为辐照度是以单位面积表示的，而面辐射强度以单位透视缩小曲面面积表示，所以乘以$\bar{N} \cdot \bar{L}$即将其转换为以单位非透视缩小曲面面积表示。

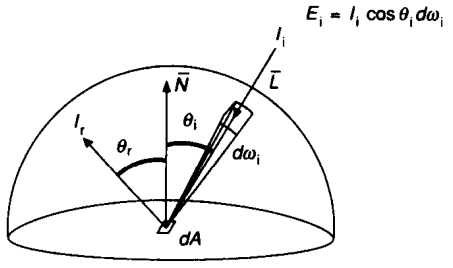


图16-40 反射光的面辐射强度和入射光的辐照度

在求I_r（反射光的面辐射强度）时仅仅考虑I_i（入射光的面辐射强度）是不够的，必须同时考虑E_i（入射光的辐照度）。例如，若两束入射光线有着相同的辐射强度（W/sr），但有着较大立体角的入射光相应地有较大的E_i，进而相应地导致物体看起来更亮一点。我们把某个方向上反射光的面辐射强度与相应的入射光的辐照度（光通量密度）的比称为双向反射率ρ，它是一个与入射光和反射光角度有关的函数

$$\rho = \frac{I_r}{E_i} \tag{16-35}$$

因此，将式(16-34)中的E_i替换，我们得到

$$I_r = \rho I_i(\bar{N} \cdot \bar{L})d\omega_i \tag{16-36}$$

图像中一像素的入射光的辐照度（图像辐照度）与对准该像素场景所发出的面辐射强度（场景面辐射强度）成比例。该比例因子是所用成像系统的一个函数。

如我们所见，在计算机图形学中，通常把双向反射率看成是漫反射分量和镜面反射分量的组合。因此有

$$\rho = k_d\rho_d + k_s\rho_s \tag{16-37}$$

其中ρ_d和ρ_s分别是双向漫反射率和双向镜面反射率，而k_d和k_s则是前面引入的漫反射系数和镜面反射系数，$k_d + k_s = 1$。请注意式(16-37)仅仅是一个有用的近似，并不适用于所有表面。例如，月球表面的ρ值，在入射方向上达到峰值[SIEG81]。这便解释了在为什么月圆之夜，当太阳、地球、月亮几乎在一条直线上时月亮看起来为均匀亮度的圆盘。相反，如果月亮是一个朗伯曲面，那么它的中心将比四周反射更多的光。

除了考虑直接光源的光照效果，我们还应该考虑从别的表面反射过来的光产生的光照效果，到目前为止，所有光照明模型都假定环境光从各个角度均匀入射，与观察者位置无关，且不被附近任何物体所遮挡。接下来将讨论如何去除这些限制。现在我们先保留这些假设，并把环境光表达为$\rho_a I_a$，其中ρ_a是无方向性环境光的反射率，而I_a是入射环境光亮度。

于是，n个光源的光照方程为

$$I_r = \rho_a I_a + \sum_{1 \le j \le n} I_{ij}(\bar{N} \cdot \bar{L}_j)\, d\omega_{ij}(k_d \rho_d + k_s \rho_s) \tag{16-38}$$

760
~
763

用式(16-38)重新解释式(16-15)的光照模型，我们把漫反射率看成物体的漫反射颜色，而镜面反射率则通过物体的镜面反射颜色与$\cos^n\alpha$的乘积来表示。我们已经注意到关于镜面反射率等式的一些不足之处。下面我们便来研究如何去掉这些不足之处。

16.7.1 表面模型的改进

由应用物理学家提出的Torrance-Sparrow模型[TORR66；TORR67]是一个关于反射面的物理模型。Blinn首先把Torrance-Sparrow模型应用于计算机图形学，给出详细的数学细节并在［BLIN77a］中将其与Phong模型进行了比较，而Cook和Torrance[COOK82]则在该模型的实现中，首次模拟了反射光的光谱组成。

在Torrance-Sparrow模型中，物体表面被看成一组各向同性的平面微面元，每一个微面元都是非常光滑的反射面。这些微面元的几何特征和分布特征以及光线的方向（假设光线从无穷远处发出，因此所有光线都是相互平行的）决定了镜面反射的亮度和方向是I_P（点光源的亮度）$\bar{N}$、$\bar{L}$和$\bar{V}$的函数。实验测量表明，实际的反射效果和用此模型来预测的反射效果相当一致[TORR67]。

Cook和Torrance使用以下公式计算双向反射率中的镜面反射分量：

$$\rho_s = \frac{F_\lambda}{\pi} \frac{DG}{(\bar{N} \cdot \bar{V})(\bar{N} \cdot \bar{L})} \tag{16-39}$$

其中D是微面元方向的分布函数，G是几何衰减因子，它代表各微面元之间遮挡和阴影效果，而F_λ则是用菲涅耳公式计算出来的菲涅耳项（将在后面介绍）。该公式在镜面反射时表示了每个光滑微面元的反射光与入射光的关系。分母中的π用来解释表面的粗糙程度（要想了解该公式是如何推导的请参见[JOY88，pp227-230]）。$\bar{N} \cdot \bar{V}$项使得公式与观察者在一块按透视缩小曲面的曲面面积内所看到的表面积（当然也是微面元的数目）成比例，而$\bar{N} \cdot \bar{L}$项则使得公式与光在一块按透视缩小曲面面积内所看到的表面积成比例。

16.7.2 微面元分布函数

因为微面元被看成完美镜面反射面，因而模型只需考虑那些法向与16.1.4节介绍的中间向量一致的微面元。仅占比例D的微面元有这种方向。Torrance 和Sparrow在他们开始的工作中假设D呈高斯分布。Blinn则使用Trowbridge和Reitz分布[TROW75]，而Cook和Torrance则使用Beckmann分布函数[BECK63]。Cook和Torrance指出Beckmann分布有着很好的理论基础并且无随意的常数，这与Torrance和Sparrow还有Bllinn所采用的分布函数不同。对于粗糙表面，Beckmann分布函数为

764

$$D = \frac{1}{4m^2\cos^4\beta} e^{-[(\tan\beta)/m]^2} \tag{16-40}$$

其中β是$\bar{N}$和$\bar{H}$的夹角，m为微面元斜率的均方差[⊖]。当m很小时，微面元的斜率只轻微偏离平面的法线。这样正如我们所期待的那样，反射光具有很高的方向性（图16-41a）。当m很大时，微面元的斜率变得更大，由此产生的粗糙平面将反射光线分散开去（图16-41b）。为了描述具有多种粗糙程度的表面，Cook和Torrance 使用一个分布函数加权和：

$$D = \sum_{1 \le j \le n} w_j D(m_j) \tag{16-41}$$

其中权值w_j的和为1。

⊖ Hall[[HALL89]提及在[COOK82]一文的分母中少了“4”。

16.7.3 几何衰减因子

本模型考虑了一些微面元相互遮蔽的情况。Torrance和Sparrow还有Blinn都讨论了G的计算方法，考虑了以下三种不同情况。如图16-42a所示微面元的所有入射光皆被反射出去。图16-42b所示微面元接受了所有光线，但其部分反射光线却被其他微面元所遮挡。这些被遮挡的光线正是形成漫反射的原因。图16-42c展示的是一个部分入射光被遮挡的微面元。几何衰减因子G的取值从0（完全遮蔽）到1（无遮蔽）。

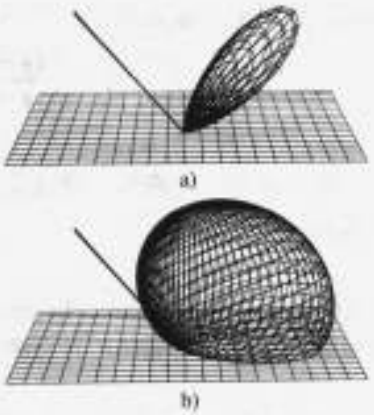


图16-41 Beckmann微面元分布函数。a)m = 0.2，b)m = 0.6（经许可摘自［COOK82］。由Robert和Cook提供，康奈尔大学计算机图形学项目。）

765

为了简化分析，假设微面元形成对称于表面平均法向量$\overline{N}$的V形槽。在图16-42a中，所有的入射光都反射给观察者，因此，G_a为1。在另两个例子中，被遮挡的光线比例为m/l，其中l是微面元的整体面积，而m则是其反射光线被遮挡的微面元面积之和（图16-42b）或被自身遮挡了入射光的微面元面积（图16-42），因此$G_b = G_c = 1 - m/l$。Blinn给出了一个图16-42中反射光比例的三角推导公式：

$$G_b = \frac{2(\overline{N}\cdot\overline{H})(\overline{N}\cdot\overline{V})}{(\overline{V}\cdot\overline{H})} \tag{16-42}$$

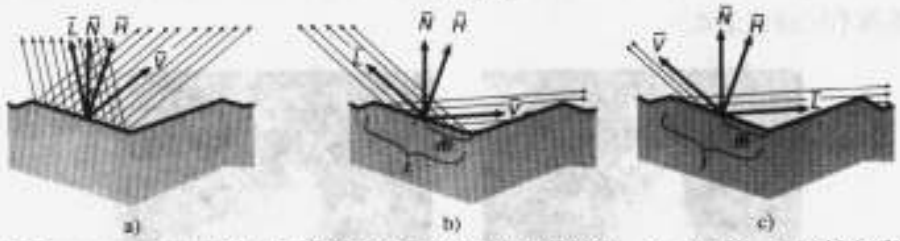


图16-42 Torrance-Sparrow模型中光线从表面微面元反射的情况。a) 无干扰，b)反射光受到部分遮挡，c)入射光受到部分遮挡。（选自［BLINN77a］。）

下面的公式是图16-42c中情形中的反射光比例，请注意它与图16-42b中情形相同，除了$\overline{L}$和$\overline{V}$交换了位置：

$$G_c = \frac{2(\overline{N}\cdot\overline{H})(\overline{N}\cdot\overline{L})}{(\overline{V}\cdot\overline{H})} \tag{16-43}$$

分母之所以没有改变是因为$\overline{H}$定义为向量$\overline{V}$和$\overline{L}$的中间向量，有（$\overline{V}\cdot\overline{H}$）=（$\overline{L}\cdot\overline{H}$）。

G取以下三个值中的最小值：

$$G = \min\left\{1, \frac{2(\overline{N}\cdot\overline{H})(\overline{N}\cdot\overline{V})}{(\overline{V}\cdot\overline{H})}, \frac{2(\overline{N}\cdot\overline{H})(\overline{N}\cdot\overline{L})}{(\overline{V}\cdot\overline{H})}\right\} \tag{16-44}$$

16.7.4 菲涅耳项

非偏振光的菲涅耳公式描述一个非导性物质表面上反射光的比例为：

$$F_\lambda = \frac{1}{2}\left(\frac{\tan^2(\theta_i-\theta_t)}{\tan^2(\theta_i+\theta_t)} + \frac{\sin^2(\theta_i-\theta_t)}{\sin^2(\theta_i+\theta_t)}\right) = \frac{1}{2}\frac{\sin^2(\theta_i-\theta_t)}{\sin^2(\theta_i+\theta_t)}\left(1+\frac{\cos^2(\theta_i+\theta_t)}{\cos^2(\theta_i-\theta_t)}\right) \tag{16-45}$$

766

其中，θ_i为相对于$\overline{H}$的入射角（比如，$\cos^{-1}(\overline{L}\cdot\overline{H})$），和以前一样，$\theta_t$是折射角：$\sin\theta_t$ =

$(\eta_{i1}/\eta_{i2})\sin\theta_i$，其中$\eta_{i1}$和$\eta_{i2}$是两种介质的折射率。菲涅耳公式也可写成

$$F_\lambda = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left(1+\frac{[c(g+c)-1]^2}{[c(g-c)+1]^2}\right) \tag{16-46}$$

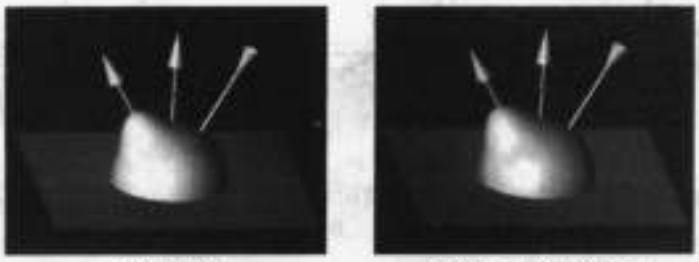
其中$c=\cos\theta_i=\overline{L}\cdot\overline{H}$，$g^2=\eta_\lambda^2+c^2-1$，$\eta_\lambda=\eta_{i1}/\eta_{i2}$。

在一个可传导光的介质里，光线会逐渐减弱，这就有必要引入$\overline{\eta}_\lambda$，即材料折射率的复数表示，定义为：

$$\overline{\eta}_\lambda = \eta_\lambda - i\kappa_\lambda \tag{16-47}$$

其中κ_λ是该材料的*消光系数*（coefficience of extinction），该系数度量光在此材料中每经过单位路径长度所衰减的量。为简化传导介质的反射计算，κ_λ可以假设为0，则$\overline{\eta}_\lambda$即为等同于η_λ的实数值。

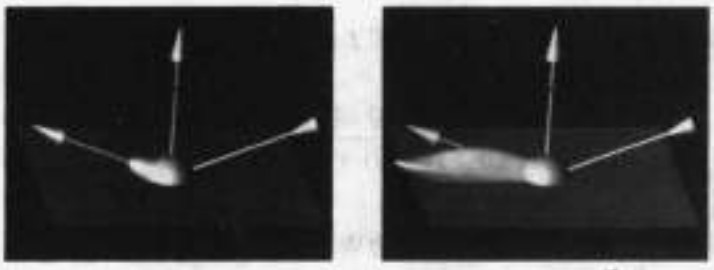
Blinn生成了图16-43和图16-44，以比较Phong模型和Torrance -Sparrow模型的效果。他做了些简化假设，即镜面反射部分仅仅与入射光的颜色有关，而且观察者和光源都位于无穷远处。图16-43和图16-44分别显示了入射角为30°和70°时表面反射照明效果。在每幅图里，垂直的箭头代表表面的法线方向，入射箭头代表光线的入射方向，出射的箭头代表一个纯反射平面的反射方向。在每幅图中，圆球状部分是漫反射分布而突出的部分则是镜面反射部分。对图16-43所示入射角30°的情形，两个模型所产生的结果非常接近，但对图16-44所示入射角为70°的情形，Torrance -Sparrow模型产生更强烈的镜面反射，而且*峰值*发生在大于入射角的角度上。这种可以在实际中观察到被称为*偏离镜面反射峰值*（off-specular peak）的现象，同样也是由Blinn所生成的图16-45则表明当光源从视点位置逐渐向两边移动，然后移向金属球背面时，两种模型所产生的截然不同的视觉效果。



a)Phong模型　　b)Torrance-Sparrow模型

图16-43 当入射角为30°时，Phong模型和Torrance-Sparrow模型的比较。（经犹他大学许可，由Blinn提供[BLIN77a]。）

767



a)Phong模型　　b)Torrance-Sparrow模型

图16-44 当入射角为70°时，Phong模型和Torrance -Sparrow模型的比较。（经犹他大学许可，由Blinn提供[BLIN77a]。）

1. 镜面反射颜色偏移

和Blinn一样，Cook和Torrance 使用Torrance -Sparrow表面模型来确定镜面反射项。但是和Blinn不一样的是，他们把镜面反射光的颜色看成是物体与入射光线相互作用的函数，既依赖于入射光的波长，也依赖于它的入射角。该方法是正确的，因为菲涅耳方程，即式(16-45)，本身会导致镜面反射颜色的偏移，而这种偏移基于入射光与微面元法向 $\overline{H}$ 的夹角，如图16-46所示。



a)Phong模型　　b)Torrance-Sparrow模型

图16-45 从不同角度照射下的金属球的Phong模型和Torrance-Sparrow模型的比较。在光源从背部照射时差别最为明显（下面的几行）。（经犹他大学许可，由Blinn提供[BLIN77a]。）

当入射光线与 $\overline{H}$ 相向时，$\theta_i = 0$。因此$c = 1$，$g = \eta_\lambda$。将这些代入式(16-46)，得到$\theta_i = 0$时的非涅耳项

$$F_{\lambda 0} = \left(\frac{\eta_\lambda - 1}{\eta_\lambda + 1}\right)^2 \tag{16-48}$$

768

当入射光与微面元表面相切时，此时$\theta_i = \pi/2$，因此$c = 0$。将式(16-46)进行变量代换，得到$\theta_i = \pi/2$时的非涅耳项：

$$F_{\lambda \pi/2} = 1 \tag{16-49}$$

因此，若光线垂直于表面，则$F_{\lambda 0}$和镜面反射系数ρ_s都是表面折射率的函数，而折射率本身又随波长而变化。当光线与物体表面相切时（且当以180°角对着光观察时，因为仅仅那些法线为 $\overline{H}$ 的微面元才会被考虑），则$F_{\lambda \pi/2}$和镜面反射率都是1。对于所有θ_i值，除了$\pi/2$以外，镜面反射率都依赖于η_λ。对于金属材料而言，本质上所有的反射发生在材料表面且都是镜面反射。仅仅在镜面反射发生最强烈的角度上，镜面反射才不会被物体材质所影响。注意和Phong模型中镜面反射项不同，它总是不受物体本身颜色的影响。Cook和Torrance指出单色的 Phong模型镜面反射项很好地模拟了塑料，塑料的颜色就是由嵌在其透明基底内部的颜料微粒所产生的。这些微粒的存在引起了漫反射，是光在塑料内无规律弹射的结果。这种漫反射是一个关于颜料和光线颜色的函数。但是镜面反射仅仅发生在透明物体的表面，因此不受颜料颜色的影响，这也就解释了为什么采用Phong光照模型绘制的物体看起来像塑料。

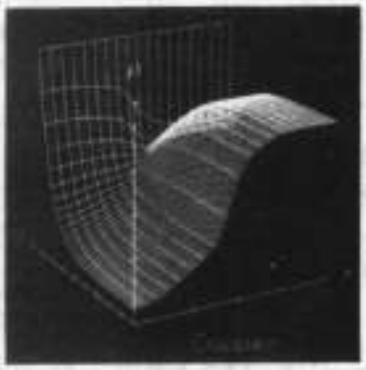


图16-46 铜镜的非涅耳项，它是关于入射光的波长和入射角角度的函数。（由Robert和Cook提供，康奈尔大学计算机图形学项目。）

如果不同波长光线的折射率已知，那么就可以将其直接用于菲涅耳公式。但一般地，它们是未知的。而且多种材料对入射角$\theta_i = 0$情况下不同波长的光的反射系数得到了测量，并记录在文献

[TOUL70; TOUL72a; TOUL72b]里。在这种情况下，每一个η_λ都可以用式(16-48)来决定如下：

$$\eta_\lambda = \frac{1 + \sqrt{F_{\lambda 0}}}{1 - \sqrt{F_{\lambda 0}}} \tag{16-50}$$

所得的η_λ值可以用于菲涅耳公式求解出任意θ_i下的F_λ值。Cook和Torrance并没有对每一个λ值采用这种计算方法，而是通过为η_{avg}计算$F_{avg\theta_i}$来简化高昂的色彩偏移运算。这里η_{avg}代表相应于平均法向反射系数的平均折射率。他们用所计算得到的$F_{avg\theta_i}$值在$\theta_i = 90°$和$\theta_i = 0°$的材料颜色之间
769 进行插值求取颜色模型的各分量。因为$F_{\lambda\pi/2}$总为1，在$\theta_i = 90°$时材料颜色通常就是入射光的颜色。因此，当光线以90°的入射角与材料表面相切时，反射光的颜色即是入射光线的颜色。在RGB颜色系统下，我们称$\theta_i = 0°$时材料颜色的红色部分为Red_0，而入射光颜色的红色部分$Red_{\pi/2}$。通过入射光光谱F_0与图13-22中的颜色匹配曲线乘积的积分来求取Red_0。$Red_{\pi/2}$可以通过将矩阵M的逆作用于式(13-18)求取。用下面的近似公式计算该种材料在入射角为θ_i时材料颜色：

$$Red_{\theta i} = Red_0 + (Red_{\pi/2} - Red_0)\frac{\max(0, F_{avg\theta_i} - F_{avg0})}{F_{avg\pi/2} - F_{avg0}} \tag{16-51}$$

于是Red_{θ_i}可以替代式(16-39)中的F_λ值。因为这个近似值考虑了入射光的光谱，必须修改式(16-38)，给镜面反射项乘以一个与波长无关的亮度比例因子，而不是乘以入射光的光谱值。Hall[HALL89]提出了一种在已知$F_{\lambda 0}$条件下求取$F_{\lambda\theta_i}$的近似插值方法。因为$F_{\lambda\pi/2}$总为1（如式(16-51)中的$F_{avg\pi/2}$），所以

$$F_{\lambda\theta_i} = F_{\lambda 0} + (1 - F_{\lambda 0})\frac{\max(0, F_{avg\theta_i} - F_{avg0})}{1 - F_{avg0}} \tag{16-52}$$

彩图III-3给出了两个用Cook-Torrance模型绘制的两个铜光瓶，两者都使用了铜的双向反射系数作为漫射项。第一幅图使用了塑料镜面的反射系数来模拟镜面反射项，所示结果相似于用式(16-14)所表示的原始Phong光照模型所产生的结果。第二幅图则采用了铜质镜面的反射系数来模拟镜面反射项。请注意，如何计算镜面反射光颜色对入射角大小和表面材质的依赖以产生更为逼真的金属表面图案画面。

通常，无论对非光导体还是光导体，环境、漫射和镜面分量都是该物质的颜色。混合物体，例如塑料，它的漫射和镜面分量通常有不同的颜色。金属通常没有漫射，但它的镜面光颜色将在该材料颜色和θ_i接近90°时光源的颜色之间变化。基于这种观察，人们提出了一种使用式(16-15)对Cook-Torrance模型的粗略近似方法，其中$\theta_{s\lambda}$的取值取决于对θ_i的一个查找表的插值。

2. 进一步的工作

Kajiya[KAJI85]进一步推广了Cook-Torrance光照模型，推导出各向异性的光照模型。在这种模型中，反射性质不再关于表面法线对称。这种模型可以更准确地模拟毛发和打光的金属光的反射
770 方式。它们的微结构更具方向性。为了做到这一点，Kajiya扩展了凹凸映射方法，不仅扰动表面法向量，同时扰动表面切向量和由切向量和法向量的叉乘生成的副法向量。所有这些量形成了一个坐标系统，该坐标系统决定了该表面相对于各向异性光照模型的方向。图16-47所示表面被映射到一个表示线交叉编织物的各向异性的纹理上。通过计算凹凸的遮挡和屏蔽效果，Cabral、Max和Springmeyer[CABR87]提出了一种决定一个由任意凹凸映射所定义的表面涂层的G和ρ值的方法。

Blinn以及Cook和Torrance所采用的菲涅耳公式只有当入射光为非偏振光时才是正确的。但是光从一个表面反射后，其偏振状态便改变了。而且一个表面的ρ值是入射光偏振状态的函数。Wolff和Kurlander [WOLF90]扩展了Cook-Torrance模型，把偏振因素考虑在内。他们的方法所产生的图显示光在两次或多次物体间反射后的两种最明显的现象。第一，非光导体都有一个称为

Brewster角的入射角，在这个角上入射光在反射时被完全偏振化，或者因为偏振方向不对而没有任何反射。如果初始非偏振光的物体间的镜面反射发生在两个绝缘体表面以致于每次反射的入射角都恰好等于Brewster角，并且一个表面的$\overline{N}$和$\overline{L}$决定的平面垂直于另一个表面$\overline{N}$和$\overline{L}$所决定的平面，那么就没有光被第二个物体反射过来；如果角度和方向是变化的，我们可以产生可觉察到的但不过大的衰减现象。第二，有色导体（如铜、金等金属）将会对不同波长的光产生不同的偏振效果。因而从一个绝缘体的表面反射出一个有色导体时，反射光具有稍许不同于不考虑偏振的情况下的颜色。

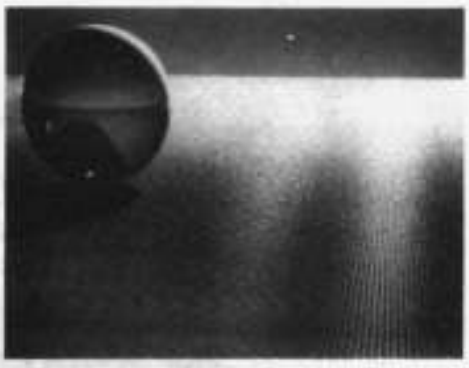


图16-47 各向异性纹理。（由加州理工学院Kajiya[KAJI85]提供。）

771

16.8 扩展光源模型

到目前为止，讨论的光源都是点光源。相比之下，*扩展光源*或者*分布光源*实际上具有一定面积，因此投射“柔和的”的阴影，包含那些光源被部分遮挡了的区域，如图16-48所示。光源的阴影中光线被完全遮挡了的阴影是全影（umbra）；只有部分光源被遮挡的阴影是半影（penumbra）。点光源的所有阴影都是全影。模拟扩展光源最直接的方法就是将其近似为一簇紧密相连的点光源[VERB84；BROT84]。但是这个方法运算很大，而且结果不尽人意，除非是非常多的点光源而且无镜面反射。

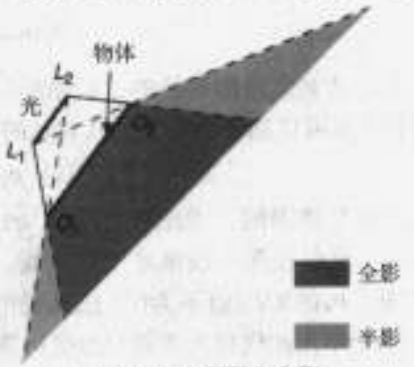


图16-48 全影和半影

Nishita 和Nakamae [NISH83；NISH85a；NISH85b]提出了阴影体的一种扩充，用以模拟具有朗伯亮度分布的线光源、凸多边形和凸多面体光源。他们的方法采用一个对象精度算法来决定凸多面体所引起的阴影体。如图16-49所示，决定了由光源上每一个顶点和多面体所形成的阴影体，半影阴影体是包含所有这些阴影体的最小凸多面体（它们的凸外壳），而全影阴影体则是这些阴影体的交集。将这些阴影体与物体的各个面相交以决定该面在半影还是在全影阴影体的区域。在一个由光源和任意多面体所确定的全影阴影体内的点不受该光源影响。决定半影区内一点的颜色涉及在该点所能看见的该光源那部分的计算。在[CHIN90]中介绍了一种基于BSP树的算法。

光线跟踪法（16.12节）和辐射度方法（16.13节）的改进很好地实现了对扩展光源的模拟。

16.9 光谱采样

当光线从一个表面反射时，入射光线的光谱能量分布P_λ将会被该表面的光谱反射函数ρ_λ所修改。该函数决定了不同波长λ的光被反射的比率。因此，反射光的光谱能量分布为$P_\lambda\rho_\lambda$。类似地，穿过一个物体的光线同样也被光谱透射函数T_λ所修改，一旦确定了落入观察者眼中的一个特定点（如一个特定的像素）的光谱能量分布，那么就可以利用式(13-18)来确定该光对应的CIE XYZ值，而且这个结果还可以通过式(13-24)的矩阵M的逆来计算RGB值。

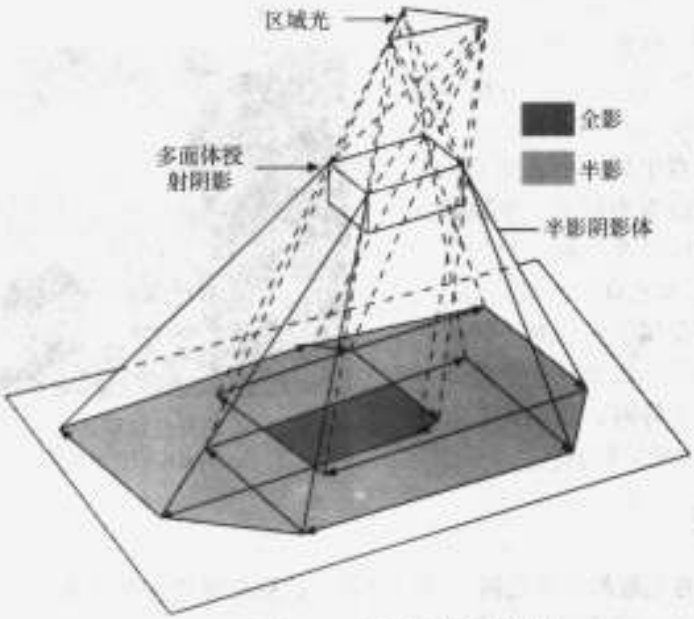


图16-49 半影和全影阴影体

有人尝试着假设用等价的三色激励RGB或XYZ值替代ρ_λ和τ_λ。如果这个想法成立的话，那么，乘积$P_\lambda\rho_\lambda$可以用下列形式的P_λ和ρ_λ的三色激励的和来代替：

$$R\rho_R + G\rho_G + B\rho_B \tag{16-53}$$

这显然是错误的，考虑图16-50中的P_λ和ρ_λ。P_λ分布是均匀的，除了600 nm附近一小段以外，同样ρ_λ仅仅在那一段里才反射光线。于是它们的乘积$P_\lambda\rho_\lambda$处处为0，因此物体表面呈黑色。另一方面，P_λ的XYZ值不为0（这只要把式(13-18)的结果做M矩阵的逆运算便可知。类似地，在式(13-18)中用ρ_λ代替P_λ并做M的逆运算，求得ρ_R、ρ_G、ρ_B的三个非零值，因此式(16-53)中的乘积也非零。这显然是个错误的结果，说明了ρ_λ和τ_λ一般并不能用一个相等的三色激励值来替换。正如Hall在[HALL83]中指出的那样，这种方法是失败的，因为实际上它是在颜色空间里进行采样，因此也易于产生所有由于采样不足造成的问题。

772
|
773

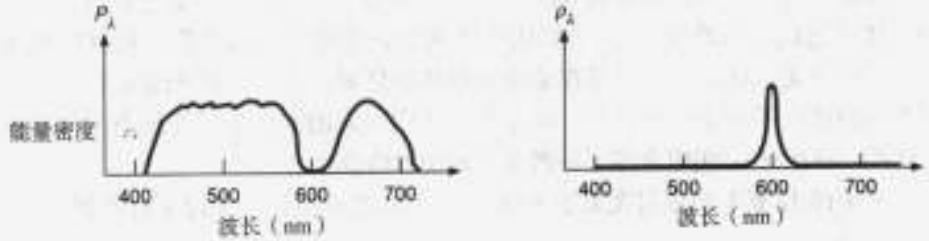


图16-50 P_λ和ρ_λ，它们的乘积处处为0

可以通过将光源光谱分布或表面的反射或透射属性表示为一条曲线来得到更准确的表示。该曲线通过对可见光谱内足够多的采样点进行插值得到。彩图III-9显示了D6500校准白光源所照射下的两个重叠滤镜的图像，它们由不同的颜色模型生成。两个滤镜没有公共的波长段，因此在两者重叠处没有任何光线可以透射过来。彩图III-9a是控制图，由360 nm到830 nm的光谱段上每纳米采样一次生成。彩图III-9b和彩图III-9c则分别由CIE和RGB颜色空间的三基色样本生成。

彩图III-9d是用9个光谱值近似的彩图III-9a的光谱。请注意，彩图III-9d比其他图更加接近控制图。

16.10 相机模型的改进

到目前为止，我们模拟由针孔相机模型产生图像：不论其位置如何，每一个物体清晰地没有任何扭曲地投影到图像中。实际的相机（和眼睛）是透镜，从而引入了各种变形和聚焦效果。

1. 景深

物体由于离镜头的距离不等，它们聚焦程度也就不同，这个效果称为景深。一个镜头有一个焦距值F，在这个距离上的物体图片可以清晰地会聚成像。如果一个点不在这个焦点上，它的像将落在距离一个大于或小于F的平面上。一个不在焦点上的点在焦距F平面上的像通常是一个圈，称为“模糊圈”。

Potmesil和Chakravarty[POTM82]采用一些后处理技术来模拟景深的一些效果和真实镜头的其他一些性质，如彩图II-38所示。他们的系统首先采用传统的针孔模型绘制方法来产生画面，该针孔模型绘制方法不仅生成每个点上的亮度，还包括每个点的z值。每一个采样点将变成一个模糊圈，其大小和亮度分布将根据其z值、透镜和光圈大小来定。输出图像中的每一个像素的亮度都是覆盖了该像素的“模糊圈”的亮度的加权平均值。因为该图像最初是从位于投影中 774
心的一个单一点计算得到的，这种技术的结果仍然只是一种近似。真实透镜的聚焦性能使得光线不再穿过针孔打到透镜并聚焦成像。从这些光线中观察到的场景有所不同，比如，一些透过针孔的光线到不了表面的某些部分。Potmesil和Chakravarty模型生成的图像遗失了这些信息。

2. 运动模糊

运动模糊是由于相机快门打开并在一定时间保持这种状态而导致画面出现拖影和模糊的现象。为了取得这种画面效果，就需要在时空过程中解决可见面的问题，并确定在一个指定像素中有哪些物体是可见的以及什么时间可见。为此，Korein和Badler[KORE83]描述了两个相对的方法：一种解析方法使用连续方程模拟物体在时间变化中发生的变化，另一种依靠时间超采样的简单的基于图像精度的方法。在此时间超采样方法中，为时间采样点生成不同的图像。运动模糊图像即是这些图像的加权和创建的，实质上即用一个时间滤波器对它们进行卷积运算。例如，如果有n个图像，每个图像的权重为$1/n$，这就相当于一个盒式时间滤波器。采样间隔越近，取得的图像越真实。但时间超采样与空间超采样一样也会遇到走样问题，除非在时间上各采样点间隔足够近，否则最终图像就会呈现出一系列不连续的多次曝光效果。Potmesil和Chakravarty[POTM83]将他们的景深方法扩展以模拟实际相机的快门运动。正如我们将在16.12节中所看到的，分布式光线跟踪中所使用的随机采样技术为透镜效果、运动模糊和空间反走样的合成提供了一个统一的框架。彩图II-39和彩图III-16充分说明了这一点。

16.11 全局光照算法

光照模型以光源直射光和经由一点所在的表面及其他表面的反射和透射的光共同决定该点的颜色，这种非直接反射和透射的光通常被称为*全局光照*。与此对应，*局部光照*是那些直接从光源到达为其计算明暗的点的光。我们用环境光项近似全局光照，环境光被认为对于所有物体上的所有点是不变的，不依赖该物体或观察者位置，也不受近旁可能阻隔环境照明光线的物体的影响。此外，我们已经通过阴影、透明性和反射图观察到了一些有限的全局光照效果。

在现实环境中，多数光线并不是直接来自光源。有两种不同的强调全局光照作用的算法被用来生成图像。16.12节讨论可见面光线跟踪算法的扩充，以交错进行可见面的判定和明暗计

算来描述阴影、反射和折射。因此全局镜面反射和透射为物体表面补充了局部镜面光照、漫射光照和环境光照。与此对应，在16.13节讨论的各种辐射度算法完全将可见面判定和明暗计算
775 分开，首先以一个独立于视点的阶段模拟环境与光源的所有相互作用，然后才使用传统可见面判定和插值明暗处理为各目标视点生成不同的画面。

依赖于视图的算法（如光线跟踪算法）和独立于视图的算法（如辐射度算法）的差别是很重要。依赖于视图的算法在给定视线方向离散视图平面成点集并在这些点计算光照方程以决定这些点的光照。与之相对应的是，独立于视图的算法将环境表面离散，为任何点任意视线方向的光照方程计算提供足够的信息。依赖于视点的算法适合处理与观察者位置密切相关的镜面反射现象，但在模拟图像大片区域变化不大或不同视点方向图像变化不大的漫射现象需要进行大量额外计算。另一方面，独立于视点算法对于处理漫射现象非常有效，但需要大量存储资源以获取与镜面反射有关的足够信息。

最终，所有这些方法都试图解决Kajiya[KAJI86]所指的绘制方程，该绘制方程根据光从一点照射到另一点的亮度以及从其他所有照射到第一点并被反射到第二点的光的亮度，来表示光从一点到另一点的转换。同样，从其他所有点转换到第一点的光可以递归由该方程表示，Kajiya将此绘制方程表示为

$$I(x, x') = g(x, x')\left[\varepsilon(x, x') + \int_S \rho(x, x', x'')\, I(x', x'')dx''\right] \tag{16-54}$$

其中[3]，x, x', x''是环境中的点。$I(x, x')$是从x'到x的亮度。$g(x, x')$是几何形状项，当x与x'相互不可见时为0，可见时为$1/r^2$，这里r是它们之间的距离。$\varepsilon(x, x')$是指由x'发射到x的光的亮度。在视点处x的$g(x, x')$ $\varepsilon(x, x')$的初始化计算即完成了x点球面上的可见面判定。方程对所有面S上的所有点进行积分。$\rho(x, x', x'')$是从x''反射并经由x'到达x的光的亮度（包括镜面反射和漫反射）。因此，绘制方程表明从x'到达x的光包括了它自身发射的和它所反射的从其他表面来的光，这些面自身发光并且递归地反射从其他表面来的光。

如我们所见，判别解绘制方程的方法是否有效，一定程度上取决于该方法如何处理这些剩余项和递归方法，它所支持的漫反射和镜面反射是怎样结合上，以及取决于它如何模拟曲面之间的可见关系。

16.12 递归光线跟踪

这一节中，我们将把基本光线跟踪算法（在15.10节讨论的）推广到处理阴影、反射和折射中去。该基本光线跟踪算法使用先前介绍的任一光照模型为与视线相交的最近一点的物体计算其所在像素的颜色。在计算阴影时，我们发出一条从该交点到每个光源的一条新的射线。图16-51中显示了只有一个光源的情况下的情形，该图引自Appel的文章[APPE68]，该文章是计算机图形学领域发表的有关光线跟踪算法的第一篇论文。如果这一束阴影光线中的任一条在该光线上与一物体相交，那

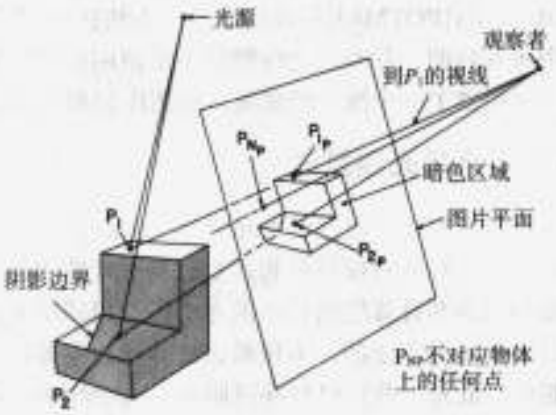


图16-51 确定物体上一点是否在阴影中。（由IBM T.J. Watson研究中心Arthur Appel提供。）

么在该点处该物体处于阴影中，并且明暗处理算法将忽略该阴影射线所对应的光源。图16-52显示了两张Appel使用这个算法用笔式绘图仪绘制的图画。根据各像素的亮度，他通过在视屏网格各像素点上放置不同大小的“+”来模拟半色调模式。为弥补网络的粗糙性，他用可见线画出了可见面边和阴影的边。

由Whitted [WHIT80]和Kay [KAY79a]开发的光照模型从根本上把光线跟踪推广到镜面反射、折射透明性。彩图III-10就是早期应用该算法生成的。除阴影线，Whitted递归光线跟踪算法从相交点处有条件地生成出反射线和折射线，如图16-53所示。这些阴影线、反射线和折射线通常称成次级光线，以区别从眼中发出的主光线。如果物体是产生镜面反射的物体，那么反射线相对于该表面的法向以方向为$\overline{R}$反射出来，该反射方向的计算可参见16.14节。如果物体是透明的且不发生任何内部反射，那么折射线将沿着T方向以Snell定律所决定的折射角进入该物体，Snell定律详见16.5.2节（注意您的入射光线可能与这些小节中的入射光线方向相反）。

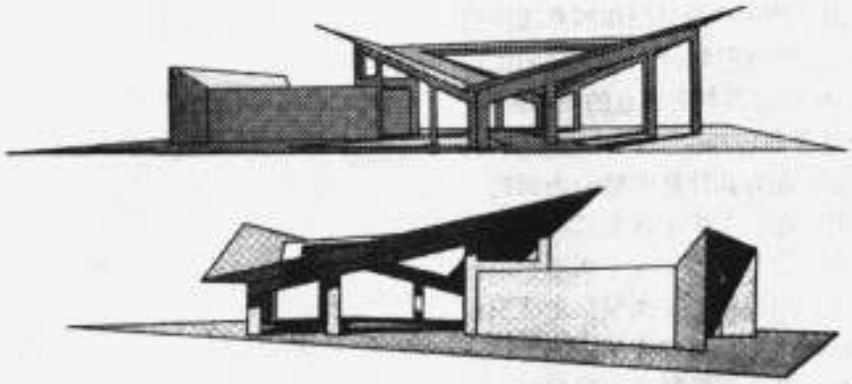


图16-52 用光线跟踪绘制的早期画面。（由IBM T.J. Watson研究中心Arthur Appel提供。）

如图16-54所示，每条反射线和折射线可能依次递归地产生阴影线、反射线和折射线。这些线就形成图16-55所示的光线树。在Whitted的算法中，在镜面反射线和折射线不再与任何物体相交时，或达到用户定义的光线跟踪最大深度或系统已耗尽所有存储资源时，光线停止跟踪，光线树由底而上地进行估算，树中的父节点的亮度由子节点的亮度计算得来，彩图III-11a和彩图III-11b就是采用递归光线跟踪算法得来的。

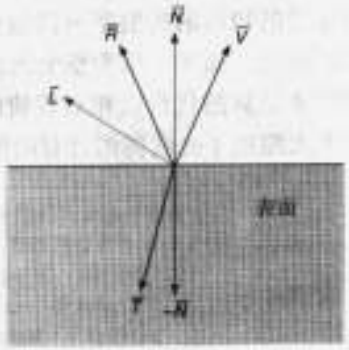


图16-53 从相交点所产生的反射线、折射线和阴影线

Whitted光照方程可表示如下

$$I_\lambda = I_{a\lambda}k_aO_{d\lambda} + \sum_{1<i<m} S_i f_{att_i} I_{pi\lambda}[k_dO_{d\lambda}(\overline{N}\cdot\overline{L}_i) + k_s(\overline{N}\cdot\overline{H}_i)^n] + k_sI_{r\lambda} + k_tI_{t\lambda} \tag{16-55}$$

其中，$I_{r\lambda}$是反射线的亮度，k_t是在0~1范围内取值的透射系数。$I_{t\lambda}$是折射光线的亮度。$I_{r\lambda}$和$I_{t\lambda}$的值通过递归地用式(16-55)对反射光线和折射光线相交的最近点进行计算来决定。Whitted用所计算得到的光线的I_λ乘以光线所经路程长度的倒数来近似计算亮度随距离的衰减。Whitted将S_i表示为该阴影线所交物体的k_t的连续函数，而不是将其作为一个delta函数。因此一个透明的物体在它投下阴影的那些点处将比非透明物体遮挡少一些光。

776 ↓ 778

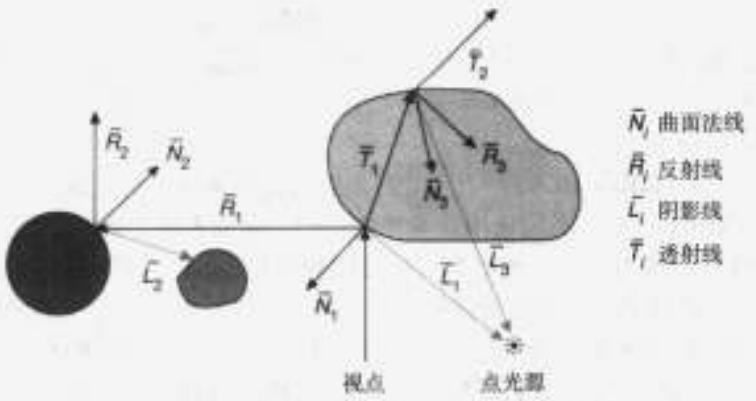


图16-54 光线递归地产生其他光线

图16-56给出了简单递归光线跟踪算法的伪代码。RT_trace过程判定与一光线相交的最近点，并调用RT_shade过程判定该点的明暗色调。首先RT_shade过程判定相交点的环境光颜色。然后，向所有位于在为其计算明暗的表面的一边的光源发出阴影光线以决定该光源对颜色的贡献。非透明物体完全遮挡了光，而透明物体将调节光的亮度，如果我们在光线树上所跟踪的深度不太深，那么为反射物体和透明物体递归地调用RT_trace以处理反射线和折射线。因为需要两种介质的折射率以决定折射方向，那么光线所穿过的物质的折射率可以包含在该光线中。RT_trace过程只将光线树保存到能判定当前像素点的颜色。如果可以为整个图像保存光线树，那么可以改变物体表面的特性，并且可以以光线树的重估算的代价以相对较快的速度重新计算一幅新图像。Sequin和Smyrl[SEQU89]给出了一些大大缩短了光线树的计算时间和减少光线树的存储空间的技术。

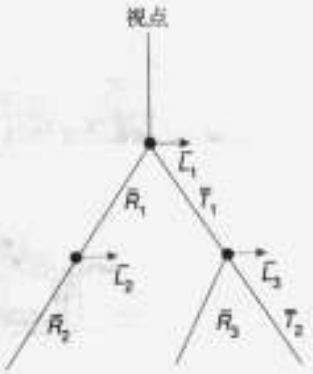


图16-55 图16-54的光线树

779

```
选择投影中心以及视图平面上的窗口
for (图像的每条扫描线) {
    for (扫描线上的每个像素pixel) {
        确定从投影中心穿过像素pixel的光线ray
        pixel = RT_trace (ray, 1);
    }
}

/* 计算与物体的交点并计算最近交点的光照值 */
/* depth是光线树的当前深度 */

RT_color RT_trace (RT_ray ray, int depth)
{
    确定光线与某个物体object的最近交点intersection

    if (命中物体) {
```

图16-56 没有反走样效果的简单递归光线跟踪算法的伪代码

```
        计算交点处物体的法向normal;
        return RT_shade ( object, ray, intersection, normal, depth ) ;
    } else
        return BACKGROUND_VALUE;
}  /* RT_trace */

/* 计算物体上一点的光照值，跟踪阴影光线、反射光线和折射光线。 */
RT_color RT_shade (
    RT_object object,          /* 相交的物体 */
    RT_ray ray,                /* 入射光线 */
    RT_point point,            /* 与其光照值的交点 */
    RT_normal normal,          /* 交点处的法向 */
    int depth)                 /* 所处光线树的深度 */
{
    RT_color color;            /* 光线的颜色 */
    RT_ray rRay, tRay, sRay;   /* 反射光线、折射光线、阴影光线 */
    RT_color rColor, tColor;   /* 反射光线颜色和折射光线颜色 */

    color = 环境光项;
    for ( 每个光源 ) {
        sRay = 从交点到光源的光线;
        if ( 法向与到光源的方向的点积为正 )
            计算被不透明物体和透明物体遮挡的光的量，
                并用该值乘漫射项和镜面反射项，然后将其加入color
        }
    }

    if (depth < maxDepth) {      /* 如果depth太深返回 */
        if ( 物体是反射物体 ) {
            rRay = 从交点向反射方向发射的光线;
            rColor = RT_trace (rRay, depth + 1);
            将rcolor与镜面反向系数相乘，并加入color
        }

        if ( 物体是透明物体 ) {
            tRay = 从交点向折射方向发射的光线;
            if ( 没有发生全反射 ) {
                tColor = RT_trace (tRay, depth + 1);
                将tColor与透射系数相乘，并加入color
            }
        }
    }
    return color;                /* 返回光线的颜色值 */
}  /* RT_shade */
```

图16-56（续）

图16-54给出了光线跟踪算法处理折射时的一个基本问题，阴影光线 $\bar{L}_3$ 并不以它到光源的路线折射。事实上如果我们以其穿过该大物体的点的方向将 $\bar{L}_3$ 进行折射，则它不会到达光源。此外，在决定一条光线的折射路径时，只为每条光线使用了一个折射率。我们将在后面讨论怎样修补这些不足之处。

光线跟踪算法特别容易产生由于有限数值精度所引起的问题。当我们计算与次级光线相交的物体时，这些问题就突现出来了。在计算可见物体上与光线相交点的x, y, z的坐标后，x, y, z

坐标被用来定义次级光线的起始点，并为其确定参数t（见15.10.1节）。如果刚刚与光线相交的物体又和一新光线相交，由于数值精度的限定，该物体往往会有一个小的、非零的参数t。如果不加处理，那么这种错误的相交点将导致视觉误差。例如，如果该光线是阴影线，那么物体可被认为它自己挡了光，导致不正确的“自阴影”的斑点表面。对这种由阴影线导致的问题的一种简单解决方法是把生成次级光线的物体当成一个特殊情况，不在其表面上进行相交测试。当然，当物体确实支持自身遮挡或者透射光线穿过一个物体并为在该物体内部被反射时，这种处理不再可行。有一个更普通的解决方法，该方法为每次相交计算abs(t)并将其与一个小的误差容忍值相比较，当abs(t)低于该误差容忍值时，就舍弃它。

780
~
781

Whitted发表在*SIGGRAPH'79*[WHIT80]的论文和他用所描述的算法所做的电影重新引起了人们对光线跟踪算法的兴趣。递归光线跟踪算法带来令人印象深刻的效果，例如阴影、镜面反射和折射透明效果等这些先前很困难也不可能获得的效果。此外，一个简单光线跟踪算法十分容易实现。因此人们投入了大量的努力致力于提高该算法的效率和图像的质量。在这里我们给出了这些问题的一个概观，并在18.11.12节讨论了几个利用了光线跟踪算法内在的并行性的并行硬件实现方法。更多的细节请参阅 [GLAS89]。

16.12.1 递归光线跟踪算法的效率考虑

15.10.2节讨论了怎样用范围、层次结构和空间划分来限定光线-物体相交计算的数量。因为某些原因，在递归光线跟踪算法中这些普通的提高效率的技术甚至比在可见面光线跟踪算法中更重要。首先，在图16-55中我们一眼就会发现随着光线树的深度的加深需要处理的光线数以指数增长。如果每条光线生成一条镜面反射线和折射线，在树的深度为n时，在最坏情况下，这棵光线树将是一个有$(2^n - 1)$条线的完全二叉树。此外，每条与物体相交的反射线或折射线又为m个光源的每一个生成出一条阴影线，那么这个光线树就可能有$m(2^n - 1)$条线阴影线。更糟的情况是，由于光线来自不同的方向，传统的基于多边形的各种有效算法，如对视见体进行物体裁剪和相对于视点进行背面的删除等，不能再用于递归光线跟踪算法。不可见的物体，包括背面，都可能通过可见面折射或反射。

1. *分项缓存*

一种加速光线跟踪的方法就是在判定对视点直接可见物体时根本不采用光线跟踪算法。Weghorst、Hooper和Greenberg[WEGH84]描述了怎样用相同的观察规范对整个场景使用耗费较少的可见面判定算法（如z缓存算法）来创建*分项缓存*。但他们是在分项缓存的像素中记录最近物体的标识，而不是在每个像素上决定其明暗色调。那么，仅仅只有这些物体需要用光线跟踪算法为该像素判定视线与物体的精确相交点，并由此生成进一步的光线。

2. *反射图*

光线跟踪在其他情况下也可避免使用。Hall[HALL86] 给出了怎样把光线跟踪和在16.6节讨论的反射图相结合。其基本思想是对次级光线作比初级光线少的工作。在判定物体是否直接可见的基础上把在一场景中不能直接可见的物体分成两组，光线跟踪被用于判定更可见物体的全局照明的贡献，同时把做了适当准备的反射图编成索引，以处理不可见物体。判定一个物体间接可见程度的一种方法就是测量，从这个间接可见物体的中心看直接可见物体时所形成的立体角的大小。如果这个立体角大于某一阈值，那么，该物体很可能包括在被反射线跟踪时的环境中（这个环境也包括直接可见物体）；否则，该物体只能在反射图中可见。当使用光线跟踪时，如果一反射线不与在反射线环境中的物体相交，那么，该光线被用来索引反射图。Hall同时指出由于反射图和折射图通常被严重扭曲，我们用相对于视线生成的低清晰度物体来绘制由折射

782

线和由反射线所看见的物体也能取得满意的效果。

3. 自适应光线树深度控制

虽然光线跟踪方法能用于描述高度反射物体，但是大部分场景中这样的物体通常并不多，一个很高的递归深度通常导致对这场景中大部分进行了无用处理。Hall[HALL83]介绍了光线树的自适应深度控制。在这种控制下，如果一条光线对像素的亮度增量低于某一预期设定的阈值，那么该光线不会被计算。假设该光线的子光线的亮度为1，并由此估计这条光线的最大贡献值，这样也就允许估计本光线对父光线的贡献。当光线树建立后，由根节点上溯到本光线的所有光线节点的最大贡献，相乘起来就可以得到本光线对像素的最大贡献。例如，图16-55中假设生成 $\overline{R}_1$ 和 $\overline{R}_2$ 的平面的k_s值分别0.1和0.05，在第一个相交面，我们估算对 $\overline{R}_1$ 光线的像素的最大贡献值为0.1，在第二个相交面，我们估算对 $\overline{R}_2$ 光线的像素的最大贡献值为$0.1 \times 0.5 = 0.005$。如果该值小于我们设定的阈值，那么我们就不用计算 $\overline{R}_2$。尽管自适应光线树深度控制在许多场景中已经取得很好的效果，但它也容易导致失败。虽然一条不用计算的光线对一个像素的明暗的影响或许微乎其微，但是一个像素可能受到的独立的且只能产生细微影响的光线的数量相当大。

4. 光缓存

我们注意到，对于每条反射线和折射线在射到一个物体时都会生成m条阴影线。然而，阴影线特殊性在于每一条线都射向一个相对小的物体部分。Haines和Greenberg [HAIN 86]介绍一个光缓存的观点用来加快阴影线的运算。光缓存就是关于光源的中心立方体，并且该立方体的边与世界坐标轴平行，如图16-57a所示。每个面被一个规则的方格栅格覆盖，每个方格伴随一个通过它能够从光源可见物体表面的深度排序的链表。这些表通过扫描转换场景中的物体填充，把物体扫描转换到以光源为投影中心的光缓存的每个面上。无论重叠面是多少，扫描转换算法将一个物体加到由该物体的投影所覆盖的每一个方格的链表上。图16-57b显示一方形区域相交面的集合。通过判别阴影通过光缓存的方格来处理计算一条阴影线。只需测试光线与表面的有序方格序列（链表）的交。因此，光缓存实现了一类光线三维视图的三维空间划分。

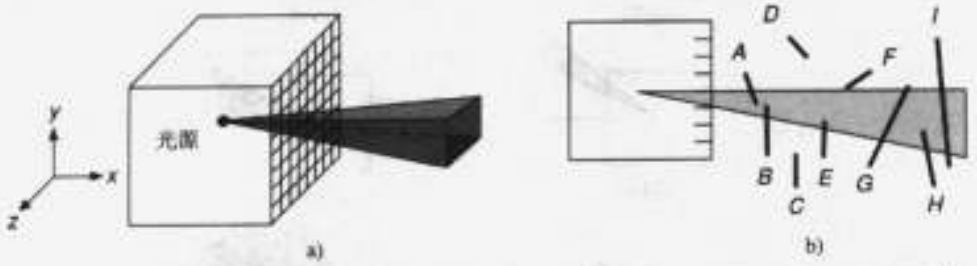


图16-57 以光源为中心的光缓存。a) 由一个方格定义的方形锥体，b)阴影部分显示方形锥体和各面相交的二维情形。方格的深度排序链表中次序为A, B, E, G, H, I

可能还有很多有效的加速方法，例如任一投影到光缓存上的物体完全覆盖了一个方格（如图16-57b中的G），那么该物体在该方格的链表中有一个完全遮挡的记录。从光源看，如果任一物体比完全覆盖的物体还远，那么这一物体将从链表中删除，在这个例子中，H和I被删除。无论何时从相交点出发的光线用方格链表来测试时，如果在这一相交点有一完全覆盖的记录位于比光源更近的位置，那么测试将马上终止。此外，背面剔除可用于避免增加任何表面到一链表中去。这些表面可以是闭合的，或者是与光线相反的表面。彩图III-12便是采用光缓存技术生成的。

Haines和Greenberg也提出用物体的相关性去判别阴影，并且这种判别甚至不需要用光缓存。与每个光源相关的一个指针被初始化为空。从一个光源看来，若一个物体被一不透明的表面遮

蔽，该光源的指针被设置为这个遮蔽表面。接下来，对于这个光源计算一条阴影线，这条阴影线首先与指针所指的物体进行相交测试。如果阴影线与该相交物体相交，那么相交面的测试将终止；否则，指针被置为空，并且继续测试。

5. 光线分类

在15.10.2节中讨论的空间划分方法提供了一个判定哪个物体在三维空间中的给定区域的方法。Arvo和Kirk[ARVO87]把这种观念推广到根据光线所交物体来划分光线，该方法就是光线分类。一条光线可以通过五维光线空间的位置来指明，这个五维空间是由光线起始点在空间中的三维位置及在球坐标中的二维方向确定的。在光线空间中一条光线被定义为一点，而一个有限子集被定义为光线族或光束。光线分类适应性地把光线空间分成为多个子集，每个子集都与该子集所包括的物体相关联（例如，子集中有其光线相交的物体）。我们仅仅需要获取一条光线所在的光线空间子集相关的物体形成的列表，就能判定与该光线相交的候选物体。

Arvo和Kirk用轴线相一致的以光源为中心的立方体中六个面的任一面的（u, v）坐标系代替球坐标来指明方向。他们用五维二叉树的六个副本细分光线空间；对于一个轴线的方向，只需要一个副本，因为一个（u, v）已指明的光线在立方体面的位置，而不是面自身。正如图16-58a所示，在二维中一组光线起始点的子集（用（x, y）表示的长方形）与一组光线方向（在四个面之一上的u间隔）结合定义了定义该光束的一个部分有界的多边形。在三维空间中，如图16-58b所示，一组光线起始点（用（x, y, z）表示的长方体）与一组光线方向（用（u, v）表示的长方形）结合定义了定义该光束的一个部分有界的多面体。用物体（或它们的范围）可能与这样的体（例如，用基于BSP树的方法生成，见12.6.4节）相交来判别这些物体是否包含在该多面体中。首先，每棵未细分的二叉树表示在三维环境区范围中全部物体的位置以及穿过该二叉树指定面任一部分的所有方向。每棵二叉树原来是与环境中的所有物体相关联的。

783
|
784

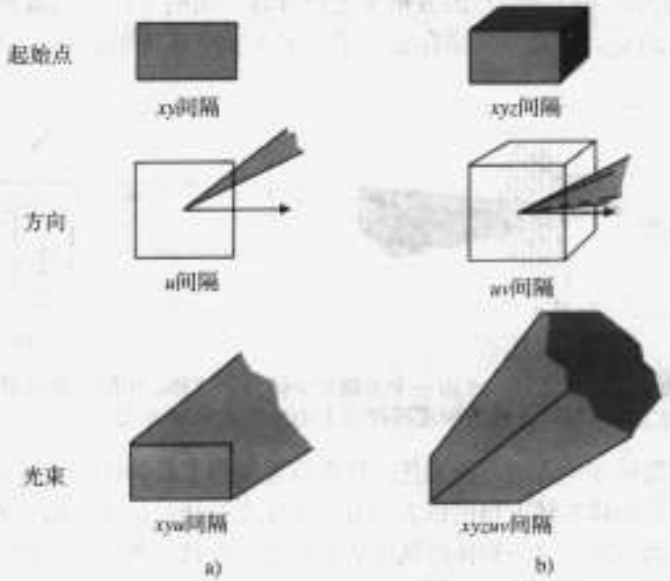


图16-58 光线分类：a)二维空间光束，b)三维空间光束。（选自[ARVO87]。）

Arvo和Kirk在跟踪所有光线的同时用惰性计算法来建立所有树。当一条光线被跟踪时，它的方向决定二叉树的根，光线与根在（u, v）坐标系下相交。如果有超过某一阈值数的物体在一节点上存在，在该节点上，该二叉树将沿该节点5个轴方向中任一方向平等地细分。随着每

次细分，仅该光线所在的子节点被计算，该节点从它的父节点中继承那些在该子节点中保存的物体。当细分停止时，只有那些与光线的划分相关的物体才能返回相交信息，作为光线跟踪的划分数据结构减少了关于不与光线相交的物体的那部分计算量。

16.12.2 一个更佳的照明模型

对于式(16-55)可以做多种推广。Hall[HALL83]提出了一种光照模型，在该光照模型中，用依赖于波长的菲涅耳反射项来计算镜面反射光。加入了透射的局部光的附加项考虑了直接来自光源的透射光的贡献，同样也可乘以菲涅耳透射项。这些全局反射线和折射线应进一步考虑光线穿过的介质的透光度。Hall模型表达如下 785

$$I_\lambda = I_{a\lambda}k_a O_{d\lambda} + \sum_{1 \le i \le m} S_i f_{atti} I_{p\lambda i} [k_d O_{d\lambda}(\overline{N} \cdot \overline{L}_i) + k_s(\overline{N} \cdot \overline{H}_i)^n F_\lambda + k_s(\overline{N} \cdot \overline{H'}_i)^n T_\lambda]$$

$$+ k_s F_\lambda I_{r\lambda} A_r^{d_r} + k_s T_\lambda I_{t\lambda} A_t^{d_t} \tag{16-56}$$

其中，d_r是反射线穿行的距离，d_t是透射线穿行的距离，A_r是该反射线在材料中单位长度的透射率，A_t是该透射线在材料中单位长度的透射率，$\overline{H}'$是对于可以将光线直接从光源折射到观察者的微面元法向量，它可以用下式计算：

$$\overline{H}' = \frac{\overline{V} - (\eta_{avg\theta_t} / \eta_{avg\theta_i})\overline{L}}{(\eta_{avg\theta_t} / \eta_{avg\theta_i}) - 1} \tag{16-57}$$

其中$\eta_{avg\theta t}$是光线所通过物体的平均折射系数，T_λ是该材料的菲涅耳透射率项。注意，k_t在这没有用；这里k_s用F_λ或T_λ来缩放。因为所有的光线要么被反射，要么被折射（也可能）被吸收，所以$F_\lambda + T_\lambda = 1$。

彩图III-13给出由Whitted的模型和Hall的模型绘制的同一场景。彩图III-13b中的玻璃球的颜色显示了用透射率项得到的过滤效果。

16.12.3 区域采样的不同方法

传统的光线跟踪的最大缺点之一就是这种技术在一个固定大小的栅格里用点采样。Whitted[WHIT80]建议用视点和像素的四角定义的锥体代替每一条从视点到该像素的光线来完成未加权区域采样。在这个环境中这些锥体可能与一些物体相交，它们中一些片段可能递归地与它们相交的物体产生折射和反射。然而纯粹的实现这种设想特别复杂，因为这要求精确地计算锥体与遮挡物体的相交面，而且，当这些锥体被递归地从相交曲面反射和折射时，需要判定锥体中碎片的变化。因此，这种方法激发了几种反走样算法，同时这些算法可以利用相关性来节省绘制时间。

1. 锥跟踪

锥跟踪就是把线光线拓广为圆锥，是由Amanatdes][AMAN84]提出的，一个圆锥体就是从眼睛出发并以足够围住一个像素的角度通过这个像素。在圆锥体所经路径上，通过为最接近锥体的顶点的物体的子集计算近似的部分遮挡值来计算与圆锥体相交的物体。由球面镜和球面透镜的光学法则，反射和折射圆锥由一个关于相交物体表面曲率和相交面积的函数来决定。通过进一步扩大新的折射和反射圆锥的锥角生成在折射和反射现象上的散射效果，通过把光源作为一个球形光源生成扩展光源的模糊阴影效果。阴影圆锥生成在一个基面之上，这个基面就是光源的截面。从而光源的亮度可通过没有被阻隔物阻隔的圆锥的截面的比例计算，彩图III-14用锥跟踪生成；3个球面越来越粗糙，并且都投射了模糊的阴影。 786

2. 光束跟踪

光束跟踪是由Heckbert和Hanrahan在[HECK84] 提出的，它是在多边形环境中的对象精度

算法，该算法用锥体光束代替线光线跟踪。正如Whitted所建议的，Heckbert和Hanrahan通过由观察棱锥体定义的单个光束开始的相关性来替代穿过每个像素的跟踪光束。用相对于观察棱锥体顶点从近及远选择次序，判定在多边形环境中，这个观察棱锥体光束与哪些多边形相交。如果一个多边形相交于观察棱锥体光束，那么它就是可见的，且该多边形也可用如在19.14节中所述算法从这个观察棱锥体光束减去这个多边形部分。对于每一个可见的多边形片段，由反射和折射产生两个棱柱。算法是递归计算的，并像在光线跟踪算法中那样以终止判据来终止递归。通过适当的变换，这个多边形环境可以转化为每个新光束的坐标系。尽管光束跟踪模拟反射完全正确，但是因为折射使光线弯曲，所以折射不是线性变换，并且折射光束仅仅是近似计算。光束跟踪生成关于多边形的一棵对象精确的光束树，该光束树用多边形扫描转换算法递归地绘制。使用局部光照方程，绘制每个多边形，接着考虑父光束的镜面反射和透射，则多边形折射子光束和反射子光束尽可能的被绘制出来并取平均值。光束跟踪以更复杂的算法、限定的物体几何形状和不正确地计算折射为代价，利用相关性，取得相对于传统光线跟踪法更快的运算速度。彩图III-15用这种算法生成。

通过使用Atherton-Weiler-Greenberg的阴影算法（16.4.2节）的变形，光束跟踪也能适应于阴影的生成。从每个光源的视点出发跟踪光束来判别从光源处直接可见的所有表面，并且所产生的多边形被加入到数据结构中作为仅仅影响明暗处理的光细节多边形。这样算法产生的效果与用传统光线跟踪生成的阴影相似。

3. *光线束跟踪*

Shinya、Takahashi和Naito[SHIN87]用称为*光线束跟踪*的方法来解决用锥跟踪和光束跟踪的问题。*光线束*（pencil）是组成*中轴*光线的光丛，该中轴光线被附近*轴旁*光线所包围的。各个轴旁光线由四维向量代表，该四维向量代表该轴旁光线与附近轴旁光线的关系。其中两个维表示轴旁光线与垂直于中轴光线的平面相交，另两个维表示轴旁光线的方向。在多数情况下，仅一条中轴光线和一个固定角度就足够代表一个光线束，如果光线束都有足够小的立体角，那么反射和折射可以用4×4矩阵表达的线性变换来很好地近似。Shinya、Takahashi和Naito开发了误差估计技术来判定合适大小的立体角用于光线束中。在光线束与一物体相交的边缘处，传
787 统的光线一定能被跟踪，然而在此情况下轴旁变换是无效的。

16.12.4 分布式光线跟踪

我们已讨论过的方法用投射实体光束替代用极微小的光线采样，以避免对固定点采样的走样问题。与此相反，由Cook、Porter和Carpenter[COOK84b]提出的*分布式光线跟踪*是基于一种随机方法来超采样在“噪音”和走样之间进行权衡[COOK86]。正如我们所见，执行反走样空间采样的能力也能运用于对于场景和场景中物体的其他方面性质的采样并生成诸如运动模糊、景深、扩展光源和粗糙表面的镜面反射等效果。在该技术名中的“*分布式*”就是说明了随机地分布光线来采样生成上述效果的量。这种基本概念早已应用于包括光线跟踪的其他算法中[COOK87]。

1. *随机采样*

如14.10节中描述的那样，用低于*奈奎斯特频率*的固定间隔采样信号时，走样出现了。即使我们用超采样和滤波器计算像素的值，这种现象依然存在。然而，如果样本间隔不固定，那么定义十分严格的频率走样光谱将被噪音所替代，这种技巧使得观察者比起接受锯齿状失真这一类易见的规则走样来更容易接受。

因为纯粹随机采样将在某些地方采样过密而有些地方被忽视，所以仅仅用相等数目的透过一场景上的随机点光线来代替光线跟踪的由视点出发的固定栅格是不够的。Cook[COOK86]提出最小

距离的泊松分布，也就是在任一两个样本之间的距离都大于某个最小值的采样。但是计算这样的分布是非常费时的，即使提前计算这样的分布并用滤波器来确定每个样本对相邻像素的贡献也是如此，而且需要非常巨大的查找表来存储这样的信息。相比较，通过用一个小的随机距离间隔的元素位置偏移，代替有固定距离的采样栅格的元素位置，而取得对于最小距离泊松分布的一个满意的近似结果。这种技术就是抖动。在二维图像平面采样中，在一个固定栅格中每个样本都用两不相关的随机量抖动，在这两个不相关的随机量中，一个是x一个是y，两个都用足够小的方差来生成，而且这个误差导致采样不与原来位置重叠（见图16-59）。图16-60显示了一个最小距离泊松分布以及一个抖动的规则分布。事实上，如果抖动的数量比滤波器宽度小，考虑到没有抖动样本的位置时，则滤波器可以被提前计算。Cook、Porter和Carpenter发现4×4子像素栅格已能充分满足大多数情况。泊松分布和抖动采样的分析可见文献[DIPP85]，包括判断是否安置新的采样的位置的采样统计分析和关于执行适应性随机采样的策略都可参见文献[DIPP85；LEE85b；KAJI86；MITC87]。



图16－59 抖动采样。在二维规则网格中的每一个样本被两个微小的互不相关的随机量抖动。×是原始位置，●是抖动后点的位置

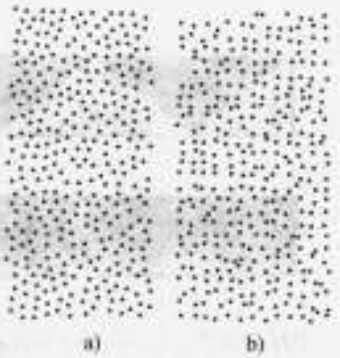


图16－60 a)最小距离的泊松分布，b)抖动后的规则分布。（经加州大学伯克利分校的Mark A. Z. Dippé和Earl Wold许可使用。）

788

图16-61比较了带有抖动和不带抖动的固定间隔的采样频率，其中采样频率有低于奈奎斯

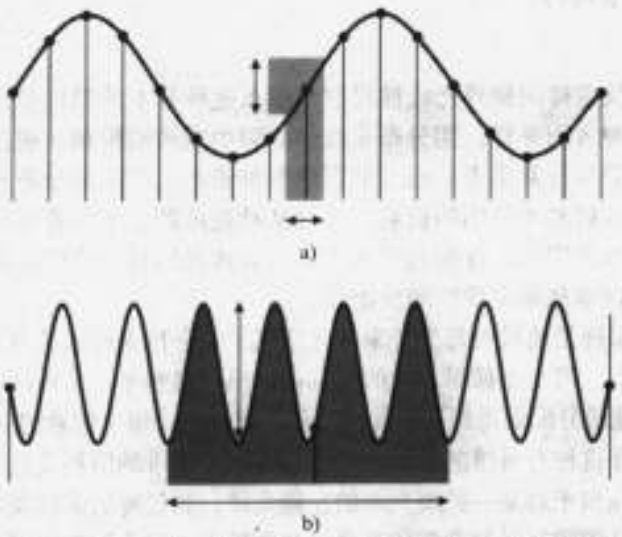


图16-61 使用固定（均匀）间隔采样，外加对采样频率的抖动。a)采样频率高于和b)低于奈奎斯特频率。每个采样点的位置如点（•）所示。水平箭头表示采样点位置抖动的范围，垂直箭头表示采样值可变化的范围。（选自[COOK86]。）

特频率的和高于奈奎斯特频率的。在图16-61a中，以高于奈奎斯特频率进行采样，虽然捕获信号的形状很好，但带有一些噪音。在图16-61b中，如果采样区间内有整周期数，那么采样的幅度是完全随机的。如果不是整周期的，那么在波形的一些部分有比别的地方更多的被采样机会，于是就产生了走样和噪音相结合的结果。频率越高，噪声与走样的比就越大。图16-62展示的是多个相等距离的三角形的梳子，其中，每个三角形宽是（$n+1$）/n个像素宽，在通过固定距离采样点采样时，产生走样的图像；另一当采样点是抖动的时，产生噪音图像。

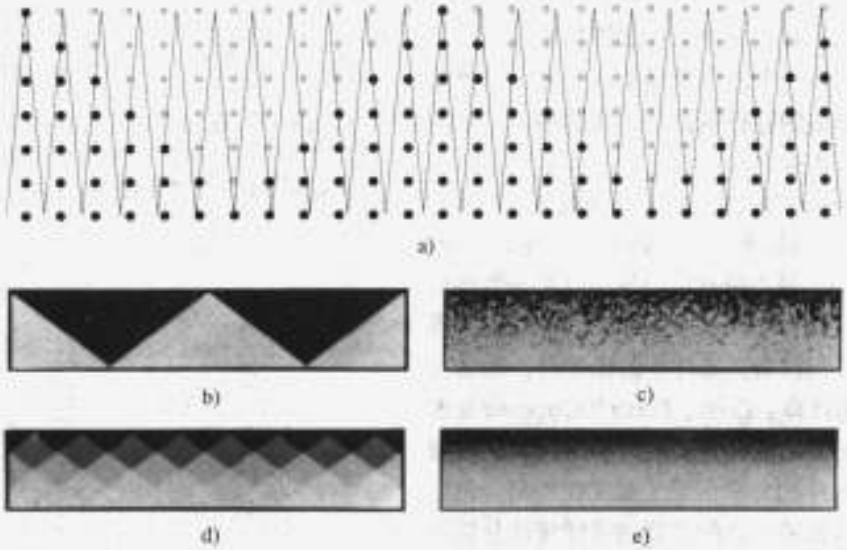


图16-62 走样与噪声。a) 带有固定间隔三角形的梳子，每个三角形的宽度为（$n+1$）/n个像素，每个像素采样一次。○表示落在梳子外边的采样点；●表示落在梳子里边的采样点。b)有200个三角形的梳子，每个三角形的宽度为1.01个像素，高度50个像素。每个像素一个采样点，规则栅格。c)每个像素一个采样点，并做±1/2个像素的抖动。d)每个像素16个采样点，规则栅格。e)每个像素16个采样点，并做±1/8个像素的抖动。（图像b)－e)由Lucasfilm公司Robert Cook提供。）

2. 其他域采样

如果需要在空间反走样的额外光线被投射，那么这种基本的随机采样技术同样也可用于分布光线来对环境的其他方面采样。用分布光线会立即生成运动模糊。通过分布在相机透镜的面积内的光线，可以将景深计算出来。通过根据透射和镜面反射函数分布光线来模拟在粗糙表面所产生模糊化的镜面反射和半透明的折射。通过从被遮蔽的点上对着看扩展光源所产生的立体角来分布光线生成柔和的阴影。在所有情况下，分布式光线跟踪使用随机采样来扰乱所跟踪的光线，这些光线还可以单独地完成空间反走样。

对非空间维度的采样是通过对每个子采样光线赋予一个待采样值的范围来实现的，然后通过抖动确定确切的采样点。每个子像素对应的范围可以通过将整个待采样区间分割为与子像素数量相同的子区间，然后随机分配而得到。因此，对任一维来说，每个像素的第i子像素都对应于某一维的同一个范围。确保这种对每维的分配方法不会导致任两维的值相关是非常重要的。例如，如果时间上较早的采样倾向于对某一扩展光源的左侧采样，而较晚的采样倾向于对该光源的右侧采样，那么在我们所描述的较早时刻遮挡住光源的右侧就不会对阴影的投射产生影响。在时域采样的例子中，每一个相交的物体必须首先移到相应的位置，在此点和采样光线相交。Cook[COOK86]建议对于物体整个运动路径计算一个包围盒，这样在包围盒测试时省去了移动物体的开销。

通过对均匀分布的样本加不同权值，可以模拟在某一维上的加权分布，图16-63a表现了这种分布。然而，用重要性采样是更具吸引力的选择，在那些权值较大的地方，采样点在比例上更多。如图16-63b所示，通过将加权滤波器分割为等面积的区域，并为每个区域赋予相同数量的等权值的采样点来实现重要性采样。每个区域的抖动的数量同样与其大小成比例变化。彩图III-16就是用分布式光线跟踪技术生成的，在图中显示了5个台球的运动模糊效果和用扩展光源生成的半影。请注意那四个运动台球的模糊阴影和反射。

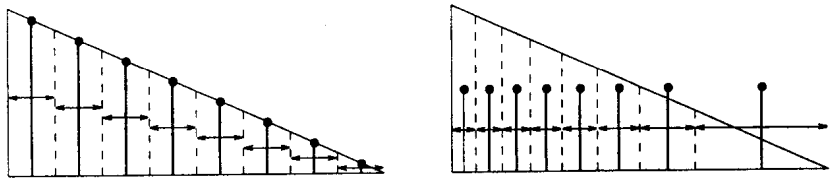


图16-63 重要性采样是通过将权函数划分成等面积区域来实现的。水平轴表示采样次数；垂直轴表示权因子。点标记采样位置；箭头表示抖动范围。a)均匀分布，权值不相等的采样。b)重要性采样：非均匀分布，权值相等的采样

789 ~ 791

3. 路径跟踪和积分方程方法

Kajiya [KAJI86]提出了一种分布式光线跟踪的变体，称为路径跟踪。每条光线不再生成一棵二叉树，而只在每个交点引发出的一条反射线或一条折射线，从而形成一条直线路径，该路径沿着一条光线直到一个光源。究竟是生成反射线还是折射线由每个像素点对各种光线的期望分布决定。Kajiya也把这种算法进行扩展以开发出绘制方程（式(16-54)）的解，称为积分方程方法。该方法考虑了光线到达某一点的所有路径。他用方差减缩方法来计算以每个交点处的镜面反射系数、漫射系数和透射系数为基础的一个随机变量，该随机变量用于判定一条在交点生成的光线是镜面反射线或是漫反射线还是折射光线，而光线的方向则通过采样来决定。此外，对于射向光源上一点的一条阴影线也可用方差减缩法来选择。因为漫射光线被跟踪，这种方法可模拟漫射线在物体之间的漫射，这种效果将在16.13节中讨论。彩图III-17就是用这种积分方程方法生成的。其中除了地板和绿玻璃球外，所有物体都是灰色的。灰色物体反射由玻璃球所聚集的绿光和从地板上漫反射的淡红光，这些现象无法用传统的光线跟踪（或用路径跟踪）生成。

Kajiya 的两种方法都是独立地估计每个像素上的漫反射，即使从一个像素到下一个像素漫反射变化往往相对较小。Ward、Rubinstein和Clear [WARD88]为光线跟踪器增加了一个递归漫反射阶段，递归漫反射阶段中的光线被用于跟踪一定数量漫射光束，光束经过从一个表面到另外一些表面的多次反射。他们不是独立地计算每个像素的漫反射值，而是缓存了所有已计算的值。当处理一个像素时，对于在该像素交点“附近”的缓存中的每个漫反射值，他们都估计该值的照明梯度作为采用该值的误差。如果这个误差是可接受的，那这些缓存值的加权平均将被作为该像素的新值，否则通过采样半球的跟踪光线计算新的漫射值，并将该值装入缓存中。该值又可以被用于计算同一场景中的别的视图。彩图III-18显示了用这种技术生成的具有不同漫反射光束数的一系列图像。彩图III-19给出绘制的会议室与真实照片的比较。

16.12.5 从光源出发的光线跟踪

由于光线跟踪中，所有的光线都是从眼睛出发的，这样会导致一个严重的问题：阴影线只投射到直接光源上，反射和折射的间接光源，诸如平面镜和透镜，无法产生正确的阴影效果。例如，由平面镜反射的光线无法生成阴影，以及由于阴影线沿直线向光源投射而导致的透明物

体的阴影与其折射不符。

我们仅需要运用传统的光线跟踪器反向跟踪从光源到眼睛的光线就能取得这些效果。这个概念被称为*反向光线跟踪*（backward ray tracing），表明它以通常光线跟踪的相反方向运行，但它也称为*前向光线跟踪*（forward ray tracing），以强调它沿着从光源到眼睛的实际路径。我们
792 称它为*从光源出发的光线跟踪*（ray tracing from the light source）以避免混淆。

然而，单纯地从光源进行光线跟踪将导致新的问题，因为大多数的光线最终将到达不了成像面，更不用说最终穿过聚焦透镜和针孔模型成像。但是从光源进行光线跟踪可以用来为通常的光线跟踪补充光照信息。Heckbert和Hanrahan[HECK84]建议改进他们提出的光束跟踪阴影方法（16.12.3节）来实现这一设想。如果递归跟踪光源的光束树，则该树第一层下的所有层次代表了所有非直接照射的多边形片段。将这些多边形片段以细节多边形的形式加入数据库中将使得非直接镜面反射光照的模拟成为可能。

Arvo[ARVO86]实现了一种光线跟踪器，在该跟踪器中，首先使用预处理步骤将光线从光源发射到环境中去。每根光线被赋予了一定的初始能量分额，这些能量的一部分存储在它与漫反射物体的每次相交中。为了弥补相对较少的光线投射到物体上，他将每个表面投射到一个由计数器组成的均匀网格上，其中各计数器将被存储的能量进行累加。每根光线的能量对射到该网格周围的四个计数器贡献按双线性形式分配。这之后进行传统的光线跟踪。在此过程中，相交点处在预处理阶段插值得到的能量与可见光源的亮度一起被用来计算漫反射值。不幸的是，如果一条光线打到一个物体的轮廓边界处对视点来说为不可见的一面，该光线将影响到可见面的阴影处理。注意，这两种从光源出发的光线跟踪方法都使用纯镜面反射几何将光线向两个方向进行扩散。

16.13 辐射度方法

尽管光线跟踪方法成功地模拟了景物表面的镜面反射和规则折射的透明效果，但该方法利用无方向的环境光代替所有其他光照的贡献。辐射度方法对热辐射的发散和反射的描绘基于热能工程模型，它为物体间的多重反射提供了一种更加精确的处理方法，避免了对漫射光线直接处理。算法首先由Goral、Torrance、Greenberg和Battaile [GORA84]以及Nishita和Nakamae [NISH85]提出，这些算法假设光能在一个封闭的环境中是守恒的。每一个表面发出或反射的能量被说明为从其他表面反射或被其他表面吸收的能量。单位时间里离开一个表面的能量称为*辐射度*，是该表面释放的能量以及其反射或透射从其本身或其他表面来的能量的总和。因此，计算一个环境中表面辐射度相关的方法被称为*辐射度方法*。与通常的绘制算法不同，辐射度方法首先以与视点无关的方式计算光在环境中的相互作用，然后只需要可见面判定和明暗插值的开销就可绘制一个或多个视图。

16.13.1 辐射度方程

在以前的明暗处理算法中，光源总是与它所照射的表面分开处理。与之相反，辐射度方法
793 允许任何表面发射光线，因而，所有光源一致地被表示为具有面积。设想把场景划分为有限数量的离散面片n，每一面片大小固定并且在该表面上均匀发射和反射光。如果把每一个面片看成是一个不透明的朗伯漫射的发射体和反射体，则对于表面i，

$$B_i = E_i + \rho_i \sum_{1 \leq j \leq n} B_j F_{j-i} \frac{A_j}{A_i} \tag{16-58}$$

B_i、B_j分别为面片i和j的辐射度，以能量／单位时间／单位面积（即W/m²）度量。E_i是其自身发射的能量，与辐射度具有相同的度量单位；ρ_i是面片i的反射系数，是一个无度量单位的量；F_{j-i}是

无度量单位的形状因子或构造因子，用来表示从面片j辐射并到达整个面片i的那部分光能，该因子的设定须将两个面片的形状以及相对方向和遮挡物的存在考虑在内。A_i和A_j是面片i和j的面积。

式(16-58)表明了离开物体表面单位面积的能量是辐射光和反射光之和。反射光由所有入射光之和乘以反射率计算得到，而入射光则是场景中离开所有面片的光的和乘以到达接受面片单位面积的比率。$B_j F_{j-i}$是离开面A_j单位面积以及到达A_i的所有光能，因而，乘以一个面积比A_j/A_i就决定了所有离开A_j以及到达A_i单位面积的光能。

自然地，漫射环境中的形状因子之间存在着一个简单的相互关系：

$$A_i F_{i-j} = A_j F_{j-i} \tag{16-59}$$

所以，式(16-58)可以简化为

$$B_i = E_i + \rho_i \sum_{1 \leq j \leq n} B_j F_{i-j} \tag{16-60}$$

移项有

$$B_i - \rho_i \sum_{1 \leq j \leq n} B_j F_{i-j} = E_i \tag{16-61}$$

因此，环境中面片间光能的相互作用可被表示为一组联立方程：

$$\begin{bmatrix} 1-\rho_1 F_{1-1} & -\rho_1 F_{1-2} & \cdots & -\rho_1 F_{1-n} \\ -\rho_2 F_{2-1} & 1-\rho_2 F_{2-2} & \cdots & -\rho_2 F_{2-n} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ -\rho_n F_{n-1} & -\rho_n F_{n-2} & \cdots & 1-\rho_n F_{n-n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \cdot \\ \cdot \\ \cdot \\ E_n \end{bmatrix} \tag{16-62}$$

注意，必须计入一个面片对它自身反射光能的贡献（例如，面片可能是凹的）。所以，通常情况下，联立方程组对角线上的每一个因子并不是1。联立方程组(16-62)应该对所有光模型中考虑的每个波段进行求解，因为ρ_i和E_i都是依赖于波长的。但是形状因子是独立于波长的，并且 794
是严格的几何函数，因而在光线及表面折射率改变的时候不需重新计算。

联立方程组(16-62)可用Gauss-Seidel迭代法求解[PRES88]，生成每一个面片的辐射度值。然后这些面片可用传统的可见面算法从任何当前视点进行绘制。计算得到的各面片各波段的辐射度就是该面片的亮度。不用微面之明暗处理方法，我们可以通过顶点所在面片辐射度值计算得到顶点辐射度,以对面片进行插值明暗处理。

Cohen和Greenberg [COHE85]提出用以下方法来确定顶点辐射度：如果顶点为一个表面内部顶点，则共享该顶点的所有面片的辐射度的均值即为该顶点的辐射度值；如果该顶点在边界上，则找到最近的内部顶点v，该边界顶点的辐射度与B_v的均值应该是共享这个边界顶点的那些面片的辐射度的均值。考虑图16-64中的面片。内部顶点e的辐射度为$B_e = (B_1 + B_2 + B_3 + B_4)/4$。边界顶点$b$的辐射度是通过找到最近的内部顶点$e$来计算。注意$b$被面片1和2所共享。因而为确定$B_b$，利用上述定义，$(B_b + B_e)/2 = (B_1 + B_2)/2$求取，得到$B_b = B_1 + B_2 - B_e$。$a$的最近内部顶点也是$e$，而$a$仅仅只是面片1的一部分，所以$(B_a + B_e)/2 = B_1$，因此$B_a = 2B_1 - B_e$。其他顶点的辐射度计算与此类似。

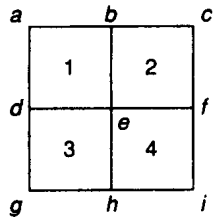


图16-64 用面片辐射度计算顶点辐射度

最初的辐射度方法由Goral等实现[GORA84]。他用轮廓线的积分为无封闭曲面的凸环境计算精确的形状因子，如彩图III-20所示。注意，由于相邻

表面间的漫反射正确的“渗色”效果在模型和绘制图像中都是可见的。为使辐射度方法能够实用，首先应开发计算封闭的曲面之间的形状因子的方法。

16.13.2 计算形状因子

Cohen和Greenberg[COHE85]采用了一种图像精度的可见面算法来近似遮挡曲面间的形状因子。考虑图16-65所示的两个面片，从微分面元dA_i到微分面元dA_j的形状因子为：

$$dF_{di-dj} = \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\, dA_j \tag{16-63}$$

795

对于图16-65中微分面元dA_i和dA_j之间的光线，θ_i是光线与A_i法向量的夹角，θ_j是该光线与A_j法向量的夹角，r是该光线的长度。H_{ij}取0或1，取决于从dA_i是否可看见dA_j。为求从微分面元dA_i到有限面积A_j的形状因子F_{di-j}，需要在面片j上进行积分，因此，

$$F_{di-j} = \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\, dA_j \tag{16-64}$$

最后，A_i到A_j的形状因子是式(16-64)在面片i上的面积均值：

$$F_{i-j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} H_{ij}\, dA_j\, dA_i \tag{16-65}$$

如果假设一个面片上的中心点代表了该面片上其他的点，那么F_{i-j}可以用计算面片i中心的微分面元dA_i的形状因子F_{di-j}来近似。

Nusselt [SIEG81]讨论了可以用投影来计算F_{di-j}，即把A_j的对于dA_i可见的部分投影到以dA_i为中心的单位半球面上，然后把投影区域再垂直投影到半球的底面上，再除以底面面积(图16-66)。其中投影到单位半球面等价于计算式(16-64)中的$\cos\theta_j/r^2$，投影到底面则相应于乘以$\cos\theta_i$，除以单位圆的面积则是考虑到分母中的π。

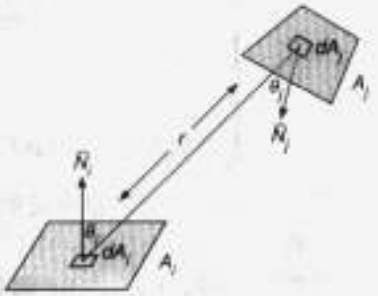


图16-65 计算一个面片和一个微分面元间的形状因子

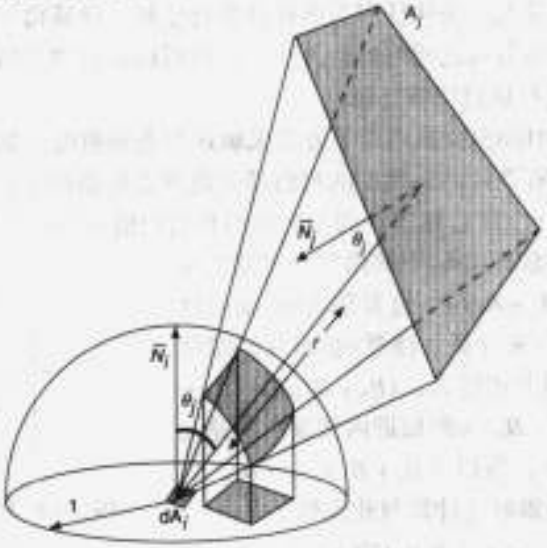


图16-66 用Nusselt的方法确定一个面片和一个微分面元间的形状因子。半球底面上的投影面积和底面面积的比值就是形状因子

除了分析上将每一个A_j投影到半球面上，Cohen和Greenberg开发了另一种有效的图像精度算法，该算法将A_j投影到以dA_i为中心的立方体的上半部分。立方体的顶面是与表面A_j平行的（如图16-67所示）。这个半立方体的每个面都被划分为许多大小相等的方形单元（本书图中所用的分辨率从50×50到数百不等）。其他所有面片被以该立方体的中心和它的五个面所定义的视见体所裁剪，并且每个被裁剪的面片都被投影到该半立方体的相应面上。可用分项缓存算法（16.12.1节）来记录每个单元上最近的相交面片标识。每一个半立方体单元中都对应一个预先计算好的Δ形状因子：

$$\Delta F_p = \frac{\cos\theta_i \cos\theta_p}{\pi r^2}\Delta A \tag{16-66}$$

其中，θ_p是单元p的表面法向量与以dA_i到p的向量之间的夹角，r是该向量的长度，ΔA是单元的面积，如图16-68所示。假设半立方体有自己的（x, y, z）坐标系统，原点在底面中心。对图16-68a中的顶面，我们有

$$r = \sqrt{x_p^2 + y_p^2 + 1} \tag{16-67}$$

$$\cos\theta_i = \cos\theta_p = \frac{1}{r}$$

其中x_p和y_p是半立方体单元的坐标。因此，对顶面来说式(16-66)可以简化为

$$\Delta F_p = \frac{1}{\pi(x_p^2 + y_p^2 + 1)^2}\Delta A \tag{16-68}$$

对垂直于在半立方体的x轴的侧面（如图16-68b所示）有：

$$r = \sqrt{y_p^2 + z_p^2 + 1} \tag{16-69}$$

$$\cos\theta_i = \frac{z_p}{r}, \qquad \cos\theta_p = \frac{1}{r}$$

这里，式(16-66)可简化为

$$\Delta F_p = \frac{z_p}{\pi(y_p^2 + z_p^2 + 1)^2}\Delta A \tag{16-70}$$

因为对称性，ΔF_p的值只需要计算顶面的1/8和单个半侧面的1/4。

我们可以用在A_j半立方体投影的每个单元p的ΔF_p值的和来近似面片j的$F_{dA_i - j}$。（注意，半立方体单元的所有ΔF_p的总和为1。）假设面片间的距离相对面片的尺寸大得多，在用式(16-62)计算面片辐射度时可以用$F_{dA_i - j}$的值作为F_{i-j}的值。彩图III-21a和彩图III-21b是用半立方体算法得到的。因为半立方体算法中大多数的计算都要用到计算分项缓存，所以可以利用现有的z缓存硬件。另一方面，因为所有操作都是图像精度的，所以半立方体很容易产生走样。

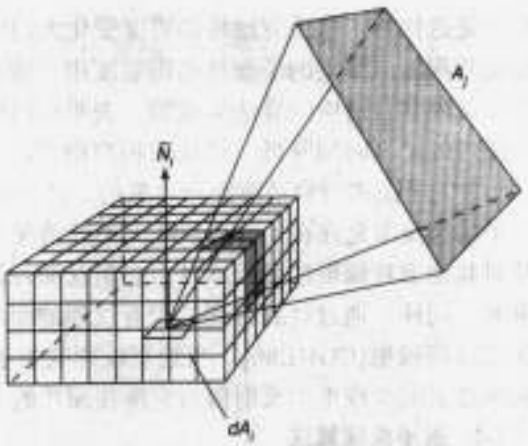


图16-67 半立方体是以面片为中心的正方体的上半部分。（选自[COHE85]。）

Nishita和Nakamae[NISH85a]采用了不同的方法来计算封闭环境下的形状因子，在该方法中他们将面光源的遮挡算法与辐射度算法相结合（16.8节），彩图III-22a和彩图III-22b即用该方法生成。彩图III-22c[NISH86]在这种方法中加入了用一个大半圆球漫射光近似的天空光线模型。将半球沿

纬度均匀但经度不均匀地划分为不同段。与其他发光表面一样，该方法模拟了遮挡物的效果。

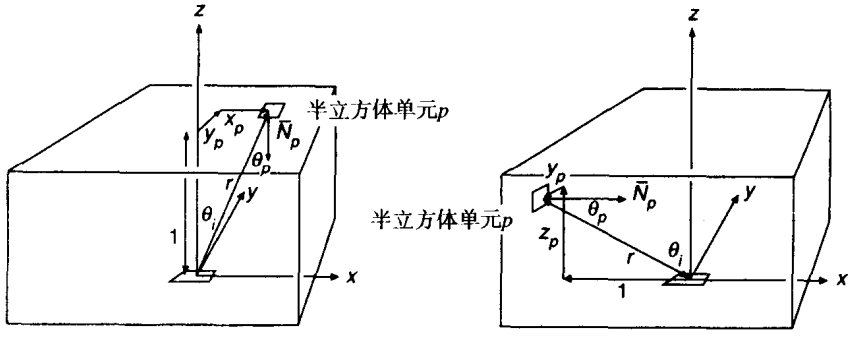


图16-68 Δ形状因子。a)顶面，b)侧面。(选自[COHE85]。)

16.13.3 子结构技术

面片参数化越精确，结果就越好，但所需形状因子的计算是以n^2为增长代价的。为了避免形状因子计算的平方增长，Cohen、Greenberg、Immel和Brock[COHE86]将面片辐射度梯度值较大的区域进一步自适应地划分为子面片。这种子结构过程生成的子面片并不像一般的面片那样。当面片i被划分成子面片时，用半立方体技术来计算从子面片s到其他面片j的形状因子F_{s-j}，但是从面片到子面片的形状因子不需要计算。而一个面片被划分为许多子面片后，以前计算的从这个面片到其他面片的形状因子，被这个更加精确的从它的m块子面片的形状因子的面积加权平均值所代替：

$$F_{i-j} = \frac{1}{A_i} \sum_{1 \le s \le m} F_{s-j} A_s \tag{16-71}$$

面片的辐射度在用以上的方法计算后，面片i的子面片s的辐射度的计算如下：

$$B_s = E_i + \rho_i \sum_{1 \le j \le n} B_j F_{s-j} \tag{16-72}$$

该算法是迭代的，自适应地将辐射度变化大的区域进一步划分为子面片，直到其间的差值达到可接受的程度。最终的子面片的辐射度用于确定顶点辐射度。彩图III-21b即在用户指定的细分方案下使用非自适应的算法生成的，表明在同样数目的面片下，对面片进行自适应的划分将使同一图像的处理时间降低。算法由用户指定的“首次猜测”细分初始化。彩图III-21c是由彩图III-21b中的子面进行自适应细分生成的。注意桌腿的阴影分辨率得到改善。

796 ~ 799

子结构技术允许在不改变矩阵大小的情况下求解式(16-62)以决定面片的辐射度。注意，子面片对其他面片辐射度的贡献是由它所在面片的辐射度很粗略地近似的，但在漫射环境中这并不重要。同样，通过计算得到的带有纹理的面片的单一的反射值的平均，用于辐射度的计算可以实现纹理映射[COHE86]。当绘制纹理映射表面的像素时，该像素的明暗色调应乘以计算得到的该像素的纹理平均反射值与它所在面片的平均反射值的比值。

16.13.4 逐步求精算法

考虑到至此所描述的辐射度求解过程的代价都很高，那么是否可以逐步地逼近该算法的结果呢？是否可以先产生一些虽然不精确但是有用的图像，可以利用它随计算时间的增加逐步提高精度呢？上几节中描述的辐射度方法不能这样做，原因有二：一是必须进行完整的Gauss-Seidel迭代后才能再对面片辐射度进行计算；二是在最初计算所有面片之间的形状因子并将其保留到计算结束，需要$O(n^2)$的时间和空间。Cohen、Chen、Wallace和Greenberg[COHE88]开发

了一种逐步求精的辐射度算法解决了上述两个问题。

考虑至今所描述的辐射度方法。式(16-62)第i行的计算为面片i的辐射度B_i提供了一种估计，B_i由式(16-60)表示，该方程基于对其他面片辐射度的估计的。式(16-60)中求和的每一项代表面片j对面片i的辐射度的影响：

$$B_i\text{归因于}B_j = \rho_i B_j F_{i-j},\ 1 \leqslant j \leqslant m \tag{16-73}$$

因而，这一方法“收集”了从环境中其他部分所发来的光。相反，逐步求精方法将从一个面片来的辐射度“发射”到周围环境中。这样做的直接方法是修改式(16-73)以得到：

$$B_j\text{归因于}B_i = \rho_j B_i F_{j-i},\ 1 \leqslant j \leqslant m \tag{16-74}$$

给定B_j的估计值，面片i对环境中其他部分的贡献可以由式(16-74)的计算决定。不幸的是，这要求对于每一个j都需要知道F_{j-i}，它由不同的半立方体决定。这使得算法具有了原有算法的庞大时空耗费。但利用式(16-59)中的相互关系，可以将式(16-74)改写为：

$$B_j\text{归因于}B_i = \rho_j B_i F_{i-j}(A_i/A_j),\ 1 \leqslant j \leqslant m \tag{16-75}$$

对每个j用等式进行计算仅仅只需用一个以面片i为中心的半立方体计算的形状因子。如果来自面片i的形状因子能很快通过计算得到（例如，用z缓存硬件），那么只要那些面片i的辐射度计算完毕，就可以将它们丢弃，因而，在某个时间内只需要计算并存储一个半立方体和它的形状因子。 800

在一个面片的辐射度“辐射”出去以后，即选择另一面片进行处理。一个面片在其他面片新的光线辐射过来以后或许会被重新选择进行辐射，因而面片i“发射”的并不是它总的辐射度，而仅仅是ΔB_i自从上次发散后面片i所收到的辐射度。算法迭代进行直到达到预定的容差。算法不是随机地选择面片，而是选择那些将产生最大差异的面片，也就是那些具有最高待射能量的面片。既然辐射度是以单位面积计算的，则应选择$\Delta B_i\ A_i$最大的那个面片。初始时，对所有面片$B_i = \Delta B_i = E_i$，仅仅对光源非零。迭代的单步伪代码如图16-69所示。

```
选择面片i;

对每个面片j计算F_{i-j};

for（每个面片j）{
    ΔRadiosity = ρ_j ΔB_i F_{i-j} A_i / A_j;
    ΔB_j += ΔRadiosity;
    B_j += ΔRadiosity;
}

ΔB_i = 0;
```

图16-69 从一个面片“发射”辐射度的伪代码

图16-69中伪代码的每一次执行都会使得另一面片将其未“发射”的辐射度发散到场景中。因此，首次执行以后，仅有光源或直接被首次选中向外“发射”的面片照到的表面才是亮的。如果每迭代一次都绘制一个新的画面，那么第一次产生的画面将相对较暗，接下来的画面将逐渐变亮。为了使早期产生的画面更加合理，我们可以在辐射度中加入一个泛光项。随着循环的每一步迭代，泛光项将逐渐减少，最终趋于零。

估计泛光项的一种方法是以未“发射”的面片的辐射度的加权和来衡量的。首先，以场景中面片的漫反射率的加权和作为场景的平均漫反射率ρ_{avg}：

$$\rho_{avg} = \sum_{1 \leqslant i \leqslant n} \rho_i A_i \Big/ \sum_{1 \leqslant i \leqslant n} A_i \tag{16-76}$$

这个等式可用来计算整体的反射因子R，以考虑面片间能量传播的不同反射路径：

$$R = 1 + \rho_{avg} + \rho_{avg}^2 + \rho_{avg}^3 + \ldots = \frac{1}{1 - \rho_{avg}} \tag{16-77}$$

801

每一面片的未发射的辐射度用面片面积和环境面积的比来加权，该比为从任意微分面元到该面片的形状因子提供了一个近似值。因而，未发射的能量的泛光项的估计值为：

$$Ambient = R \sum_{1 \le i \le n} (\Delta B_i A_i) / \sum_{1 \le i \le n} A_i \tag{16-78}$$

这个泛光项仅仅只是为了显示的目的来提高面片的辐射度值，因而

$$B'_i = B_i + \rho_i Ambient \tag{16-79}$$

图16-70描述了整个算法的伪代码。通过从面片到子面片的辐射度的发散提供了子结构技术以决定子面片的辐射度，因而，只为面片上而不为子面片建立半立方体。当相邻子面片顶点间的辐射度梯度很大时，就将该面片进一步细分，实现面片的自适应划分。彩图III-23是用泛光项进行绘制的，描绘了迭代1、2、24和100次所产生的图像。

```
for(每个面片i){
    ΔBi = Ei;
    for(i中的每个子面片s)
        Bs = Ei;
}

AreaSum = Σ(1≤i≤n) Ai;

Ambient = R Σ(1≤i≤n) (ΔBiAi)/AreaSum;

while(没有会聚){
    选择带有最大ΔBiAi的面片i;
    确定所有面片中的所有子面片的Fi-s;

    /* ΔEnergy被初始化为总能量 */
    ΔEnergy = ΔBiAi;

    /* 发射来自面片i的辐射度 */
    for(每个被i看见的面片j){
        OldΔB = ΔBj;
        for(每个被i看见的j中的子面片){
            ΔRadiosity = ρjΔBiFi-sAi/As;
            Bs += ΔRadiosity;
            ΔBj += ΔRadiosity As/Aj;
        }
        /* ΔEnergy以面片j获得的总能量递减 */
        ΔEnergy -= (ΔBj - OldΔB) Aj;
    }

    使用Bs + ρiAmbient作为面片i的子面片s的辐射度,
        由子面片辐射度确定顶点辐射度;
    if(相邻顶点间的辐射度梯度太大)
        划分子面片并重新从面片i发射给它们;
    ΔBi = 0;

    完成与视图相关的可见面判定和明暗处理

    /* 使用ΔEnergy(由射中的面片吸收的能量)来确定Ambient的前值 */
    Ambient -= R ΔEnergy / AreaSum;
}   /* while */
```

图16-70 采用泛光及子结构技术的逐步求精辐射度算法的伪代码

16.13.5 更加精确的形状因子的计算

虽然利用硬件快速z缓存可以使半立方体算法非常有效，但是这种技术仍存在很多缺点[BAUM89, WALL89]:

- 每一个半立方体的像素只能存一个面片标识。因而当一个面片网格被投影到半立方体侧面的时候会产生走样，就像用z缓存算法进行处理那样。在半立方体中面片会表现出一种本来并不存在的规则性。而且，一个投影到半立方体上很小的面片在投影到图像平面时可能会很大。
- 半立方体的应用假设面片的中心对其他面片的可见性代表该面片整个面片的可见性。如果假设并不成立，表面可以划分为许多子面片，但对该面片仅有单一的细分粒度；对同一面片所看到的不同其他面片，不能细分成不同的层次。
- 若要半立方体方法正确，面片间的距离必须足够大。如果两面片邻接，这一问题就变得严重了。既然所有的计算都是由半立方体中心得出，那么这些形状因子将被低估，因为计算时没有考虑面片的某些相邻部分是邻近的。

由Wallace、 Elmquist和Haines [WALL89]提出的逐步辐射度方法用光线跟踪而不用半立方体来估计形状因子。当一个源面片发射辐射度时，从场景中每一顶点发出到源面片的射线以计算从源面片到该顶点的形状因子。这通过将源面片划分为许多有限的小面来实现，每一个小面都是从顶点发来的射线的目标。如果该射线没有被遮挡，那么该目标是可见的。可以用一个基于一些简单的几何假设的解析表达式来计算该微分面元顶点与具有有限面积的目标之间的形状因子。如果需要，通过用一个独立于清晰度的真实曲面数据库来进行光线的相交计算，使单一射线的测试时间独立于多边形的个数。顶点和整个源面片之间的形状因子是子面片和顶点的形状因子的面积加权平均值。该结果用于计算源面片对该顶点的贡献。这种方法有很多优点。在顶点计算辐射度，这正是最终光滑明暗处理所需要的。可以使用顶点的法向量使多边形网格能够近似曲面。非真实的点光源也可以通过跟踪到光源的射线，并用光照方程来确定各顶点的入射光。源面片划分的小面数目以及是否真正发出射线（例如，进行阴影检测）可由每个顶点单独处理。彩图III-24和彩图III-25是用这种算法绘出的。

802 ~ 803

Baum、Rushmeier和Winget [BAUM 89]提出了一种不同的解决半立方体引起的不精确性问题的方法。他们认识到半立方体计算出的形状因子通常都是精确的；因而，对于每一个面片，他们提出了一种错误分析测试来确定何时进行半立方体算法，何时对面片进一步细分，何时用代价更大但更精确的解析技术来计算形状因子。

16.13.6 镜面反射

至此描述的辐射度方法仅仅处理了漫反射。因而，一个面片的所有发散到其他面片的辐射度可以统一进行处理：从任何方向离开该面片的辐射度仅由该面片的总的辐射度决定，而不受它所获取的入射能量的方向影响。Immel、Cohen和Greenberg[IMME86]把辐射度技术扩展到了模拟镜面反射。不同于对每一个面片计算一个辐射度值，他们将该面片上的半球分割为一组有限的立体角，每一个立体角为入射或发出的能量确立一个方向。考虑到面片的双向反射率，每个发射方向上的辐射度即以该方向上自身发射能量和从各方向入射光的加权贡献的和来计算。最后，用从顶点到眼睛的方向插值最接近的方向的辐射度值来计算从顶点到眼睛方向的该顶点的亮度值，并以它绘制画面。但是这种方法的时间和空间消耗都很大，而且仅能增加镜面模型环境的真实效果。另一种解决的办法是将辐射度方法和光线跟踪结合使用。

16.13.7 辐射度和光线跟踪的结合

考虑辐射度方法和光线跟踪的不同。辐射度方法适用于漫反射，因为漫射表面的双向反射

率在各个方向都是相同的，因而，辐射度计算是独立于视点的。另一方面，前面描述的镜面表
804 面的纯辐射度方法是不实用的，因为一个表面的镜面反射高度依赖于观察者（或另一个表面）的观察角度。因而必须计算许多额外信息，因为没有提供所期望的视角的任何信息。另外，这些定向信息被离散化并且进行插值计算以实现特定视线方向。不仅插值使得不可能模拟高光反射，而且这种样点的离散处理的采样方法将导致走样。

相反，光线跟踪能够很好地计算镜面反射，因为视点是预先知道的。虽然传统的光线跟踪不能描述全局漫射现象，但是在16.12.4节中讨论的一些改进方法可以做到。正确解决表面的漫反射问题需要考虑所有和它有能量交换的表面。简单地说，需要辐射度方法。

看起来应该把光线跟踪法和辐射度方法结合起来以利用光线跟踪处理镜面反射的能力和辐射度处理漫射的能力。但是，简单地将漫射辐射度方法求解的像素值与镜面光线跟踪求解取和是不够的。例如，漫射辐射度方法不能处理从镜面表面射到漫射表面的光。考虑漫反射到漫反射、漫反射到镜面反射、镜面反射到漫反射以及镜面反射到境面反射的能量转换是非常必要的。

Wallace、Cohen 和Greenberg[WALL87]描述了一种两步法。该方法第一步使用独立于视点的辐射度方法，第二步使用依赖于视点的光线跟踪方法，从而将两种方法结合起来。像前面所提到的那样，在第一步必须像考虑漫反射一样把镜面反射考虑在内。如果仅允许那些完美的像镜子一样的镜面反射存在，这可以通过将面片从镜面平面进行反射来实现。每一个镜面反射面片都被当成“镜中世界”的一个窗口来处理。从一个面片到这些镜面反射的形状因子充分考虑了做镜面反射的面片的镜面反射效果。

在依赖于视点的第二步中，对每个相应于视图中一个像素的物体表面上的一点建立一*反射截头锥体*。如图16-71所示，这个反射截头锥体由一个小的z缓存组成，垂直于反射方向并覆盖一个小的入射立体角，该立体角对表面的双向反射率非常重要。面片被z扫描进该截头锥体，并用Gouraud插值明暗处理在投影中插值计算面片第一步计算得到的漫反射亮度。从每一个可以看一个镜面反射表面的该截头锥体上的像素递归地跟踪一条光线并在每个交点上产生一个新的反射截头锥体。每个截头锥体顶点的值加权以模拟该表面的ρ_s。因而第二步用镜面反射来结合第一步得到的辐射度结果值。透明性则是通过在折射方向上建立一个透射截头锥体来描述的。书中的封面彩图I-9即用这种算法绘制。

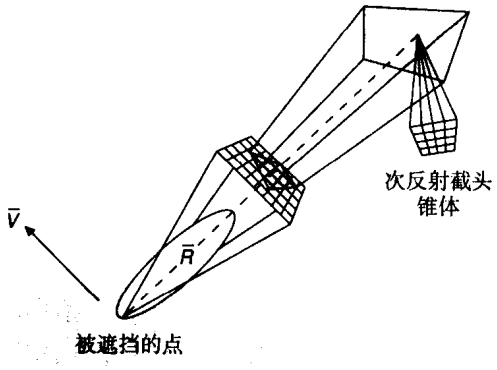


图16-71 反射截头锥体。（选自[WALL87]。）

第一步中使用的“镜中世界”的方法仅仅处理了严格的镜面反射并且导致形状因子计算的增加。Shao、Peng和Liang [SHAO88]进一步改进了两步法，允许在第一步中有类似Phong的双向反射功能，而不需要进行面片复制。

Sillion和Puech[SILL89]扩展了两步法，使之能够模拟任意数量的镜片反射和折射的*扩展形状因子*。不同于增加镜面反射形状因子，他们采用了递归光线跟踪来计算形状因子。彩图III-27说明了为什么在第一步的漫射中必须考虑镜面反射。彩图III-27a是传统的漫射辐射度方法的结果。（靠近镜子的桌子由高灯内部漫反射的光照亮。）经由第二步的纯光线跟踪方法改善的第一步传统的漫射产生了彩图III-27b，它包括了镜子的镜面反射，彩图III-27c展示了Sillion和

puech提出的两步法结果。和彩图III-27b一样在第二步中使用了光线跟踪的方法，但是在第一步用了扩展形状因子方法。每一个表面在第一阶段都扮演了照明者的角色，使用扩展形状因子意味着漫发射的能量将镜面的相互反射也考虑在内了。注意，第一步中光从镜子镜面反射到桌面和花瓶背面，彩图III-28是一个更复杂的包括一个反射球的例子。

16.14 绘制流水线

我们知道了可见面判定、光照和明暗处理的各种方法，现在可以考虑将这些处理步骤装入第7章介绍的标准图形流水线中（图7-26）。为简单起见，除特别说明外，我们均假设场景是由多边形构成的。在第18章中将更详细地讨论如何用硬件实现这些流水线。

16.14.1 局部光照绘制流水线

1. *z*缓存和Gouraud明暗处理

对标准流水线最直接的修改是用*z*缓存可见面判定算法来绘制有Gouraud明暗处理的多边形如图16-72所示。*z*缓存算法的优点是对图元的出现顺序没有要求。因而，就可以像以前一样，通过遍历数据库得到图元并通过模型变换将其变换到世界坐标系中。



图16-72 *z*缓存和Gouraud明暗处理的绘制流水线

当建立模型时就可能为图元定义了相应的表面法向量。由于光照处理需要用到表面法向量，必须用附录中所讨论的方法将法向量进行正确的变换。我们不能试图用正确变换后的顶点来重新计算新的法向量而忽略了原来存储的法向量。和物体同时定义的法向量可能代表了其表面的真正几何属性，或者可能具体指定了用户所定义的表面混和效果，而不仅仅是由近似多边形网格中相邻面的法向的平均。

下一步是要除去完全落在窗口外面的图元以及反向面剔除。现在进行这些枝节去除操作是因为我们不希望在接下来的光照处理中做无谓的工作。由于采用Gouraud明暗处理，需要在各个顶点对光照方程进行计算。这一处理必须在观察变换（包括*skew*或透视变换）前的世界坐标系（或任何与其尺寸相同的坐标系）中进行，以保持光线和表面间正确的角度和距离。如果物体没有提供顶点的法向量，则在对顶点进行光照计算前必须予以计算。剔除工作和光照处理通常是在光照坐标系下完成的，该坐标系通过对WC坐标系进行刚体变换得到的（比如，用标准的PHIGS工具创建视线方向矩阵而得到的VRC）。

接着物体通过观察变换变换到NPC，并由视见体进行裁剪。将顶点的齐次坐标除以其相应的*W*分量，并将物体映射到视口。如果一个物体部分被裁剪，那么必须为裁剪过程中产生的新顶点计算正确的亮度值。在这一刻，裁剪后的图元被交给*z*缓存算法，由其进行光栅化。在此过程中，隔行扫描转换时要对每个像点的*z*值和亮度值进行插值计算。如果确定像素是可见的，其亮度值可以进一步由深度提示效果（式(16-11)）修正，这里就不再说明。

虽然这一流水线看上去简单直观，但是必须处理许多新问题以确保高效正确的实现。例如，考虑曲面处理所产生的问题，诸如必须进行网格化的B样条曲面片。应该在变换到屏幕大小可定的坐标系后才能进行网格化，这使得网格大小可自适应地调整，并且限制了变换的数据量。另一方面，网格化的图元必须在与世界坐标系同构的坐标系中进行光照处理。Abi-Ezzi[ABIE 89]讨论了这些问题，提出了更有效但更复杂的结合反馈循环的绘制流水线表示。这一流水线用到的光照坐标系是世界坐标系WC的等距（如刚体或欧氏）变换得到的，但计算上接近于设备坐标系（DC）以允许网格化更为有效地进行。

805 - 807

2. z缓存和Phong明暗处理

上述简单的流水线若要适应Phong明暗处理则必须加以修改，如图16-73所示。因为Phong明暗处理对表面法向量而不是亮度值进行插值，所以光照处理不能在绘制流水线的早期执行。相反，每一物体要先进行裁剪（用插值法计算出新产生顶点的法向量）、观察变换并交给z缓存算法处理。最后，利用在扫描转换时用插值计算得出的法向量进行光照处理。因此，每一点和其法向量必须反向映射至与世界坐标系等距的坐标系中来计算光照方程。

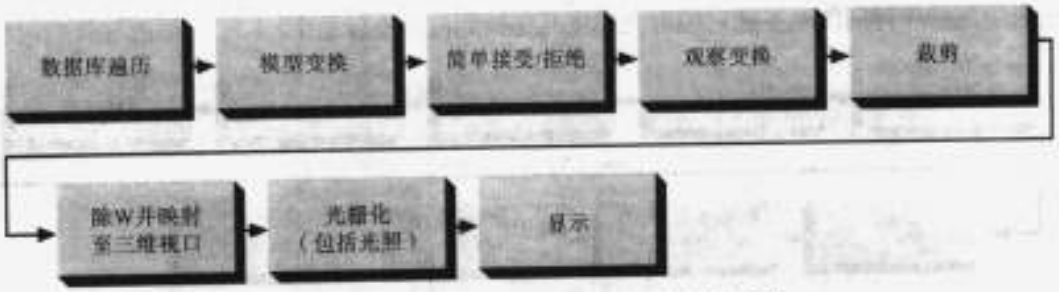


图16-73 z缓存和Phong明暗法的图形绘制流水线

3. 列表优先级算法和Phong明暗处理

使用列表优先级算法时，经由数据库遍历得到的并由模型变换处理后的图元被放入一个独立的数据库中，如BSP树，作为初始可见面判定的部分。图16-74描述了采用BSP树算法的流水线，其中初始可见面判定是与视点无关的。如我们第7章中所述，应用程序和图形软件包可以有各自不同的数据库。这里我们看到绘制也需要自己单独的数据库。在此情况下，多边形被分割，就必须为这些新产生的顶点确立正确的明暗信息。现在可以遍历这个绘制数据库生成从后向前排列的图元序列。建立这种数据库可以用来生成多幅图像，因而把它看成一个单独的流水线，该流水线的输出是一个新的数据库。从该数据库中取出的图形基元被裁剪、规格化，并交给该流水线的余下阶段。这些阶段的构造和z缓存流水线构造类似，区别在于，它们惟一需要执行的可见面判定过程是必须保证每个多边形正确地覆盖与它相交的已扫描转换的多边形。如



808

图16-74 列表优先级算法和Phong明暗处理的绘制流水线

果使用诸如Weiler-Atherton一类产生完全可见的图元的对象精度算法，那么这个简单的覆盖能力也不需要了。

16.14.2 全局光照绘制流水线

以上的讨论中忽略了全局光照。正如我们以前注意到的，加入全局光照效果需要考虑正在绘制的物体与世界中其他物体的几何关系。我们在很多例子中已经看到，一种解决的方法是在扫描转换之前为一个特定的视点计算所需信息并存入一个表中（如反射图和阴影图）。这使得在处理当前物体的时候不需要遍历存放其他物体的整个数据库了。比如，在阴影处理中，由于阴影只依赖于光源的位置而不依赖于观察者的位置，通过对场景预处理，为物体表面加入阴影细节多边形，从而可以使用一种不同于传统绘制流水线的方法。

1. 辐射度

漫射辐射度算法为如何利用传统流水线来实现全局光照效果提供了一个很有价值的例子。这些算法处理物体并且为它们赋予一系列与视点无关的顶点亮度。这些物体用修改后的z缓存和Gouraud明暗处理的流水线进行处理，如图16-75所示，该流水线不需要光照处理阶段。

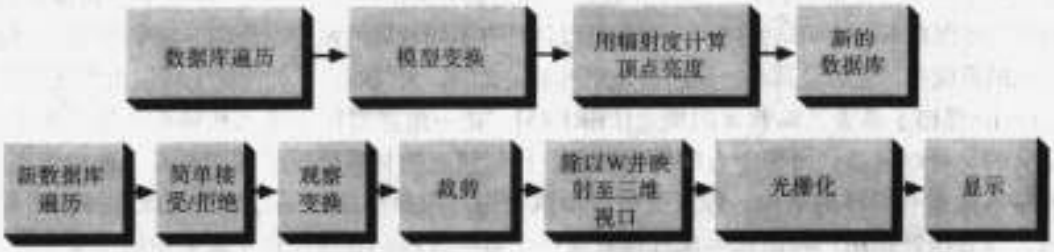


图16-75 辐射度和Gouraud明暗处理的绘制流水线

2. 光线跟踪

最后，我们考虑光线跟踪。它的流水线如图16-76所示，是最简单的，因为每个像素可见的物体以及这些物体的光照完全在世界坐标系中决定，一旦物体从数据库中获得并经过模型变换，它们就被装入光线跟踪器的世界坐标系数据库中，它可以使用15.10.3节和16.12.1节中的技术实现，以支持高效的光线相交计算。

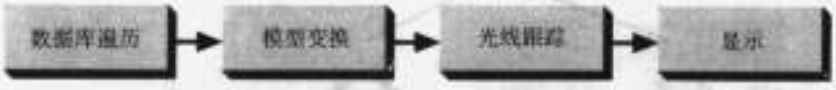


图16-76 光线跟踪的图形显示流水线

16.14.3 设计灵活的绘制法

我们已经看到许多不同的光照和明暗处理模型，选择使用哪一种模型取决于我们所关心的问题：提高效率和真实感，还是获得有趣的视觉效果。简单来说，没有一种模型能让所有的用户满意，因而人们提出一些设计方法以使得光照和明暗处理算法能更加容易实现和使用。

1. 模块化

一种直接的方法是将一些绘制系统中光照模型和明暗处理模型模块化，该模块通常称为明暗处理模块。Whitted和Weimer[WHIT82]提出，通过建立向明暗处理模块传递参数的标准机制，可以在同一系统中调用不同的明暗处理模块。甚至可以在运行的时候基于物体的属性决定应选用哪一个明暗处理模块。他们的系统用扫描线算法进行扫描转换，并将结果表示为各条扫描线上跨度的链表。每一个跨度包含它的端点的一系列信息，包括它们的x和z值，以及法向量、亮

度等附加信息。明暗处理模块在跨度内对特定值进行插值计算（因为通常使用一个扫描线z缓存，利用插值计算的z值以决定可见面，所以明暗处理模块可以被宽松地加以应用）。

Doré图形系统[ARDE89]的设计为程序员提供了更多的灵活性。该系统以一系列对象为用户提供了一种表达场景数据库的标准方法。这些对象具有绘制、拾取及计算包围体等功能。显示列表及其遍历功能组成了一个公共核心，该核心使得该系统与不同绘制系统的接口更加简便。一个程序员可以使用各种标准的对象、方法和属性，或在此框架下自己进行设计。

2. 专用语言

相对于在程序语言一级为用户提供可扩展性，设计一种专用于图形任务的专用语言也是可行的。Cook已设计了一种专用语言[COOK84a]，在该语言中，明暗处理模块表达为树的形式，称为明暗处理树。明暗处理树是由节点构成的，每个节点从其子节点处获得参数并为父节点计算参数。这些参数是光照方程中的项，如镜面反射系数、表面法向等。其中一些节点，如漫反射、镜面反射、平方根等都内建于明暗处理树所用的语言中。另一些可以由用户定义并在需要的时候动态装载。所有节点都可获得光源的信息，图16-77是描述铜材质的一棵明暗处理树。因此，一棵明暗处理树描述了一个特定的明暗处理过程并通过一种与其分离的造型语言与一个或多个物体相联系。不同物体可以有不同的明暗处理树，因而可以将具有不同效果的模型混合在一起生成图像。同样，在Cook的系统中，光源及其参数是通过光照树来定义的；大气效果（如薄雾）可以由大气树定义。

Perlin提出了像素流编辑器的概念[PERL85]，把一组像素作为输入和输出。像素并不是严
810 格定义的，可以包括该图像中点的任意信息，比如该点的材质标识或法向量等。输出像素不需要和输入像素有同样的结构。像素流编辑器执行用户用高级语言编写的针对像素操作的程序。因而，该方法鼓励用户将图像生成过程看成一个多遍过程，各遍操作生成的中间结果中包含不同的信息。

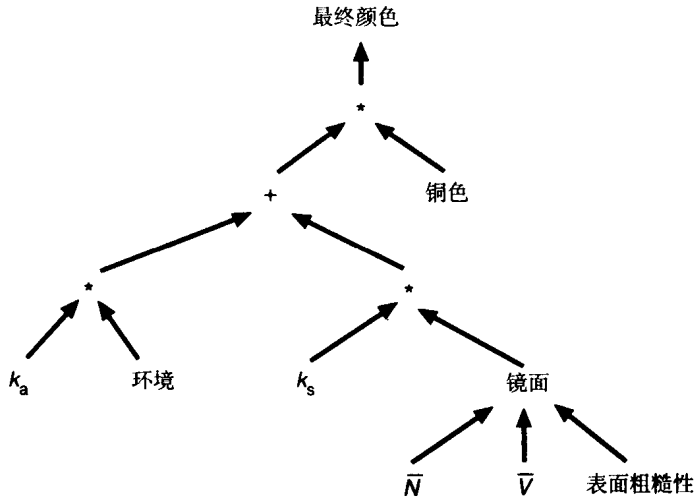


图16-77 铜材质的明暗处理树（选自[COOK84]）

可以通过设计一个既允许用户以一种专用语言编写他们自己的明暗处理模块又允许将这些模块分别与不同的物体相结合的绘制系统，从而将明暗处理树与像素流编辑器的灵活性结合起来。这种方法在RenderMan Interface中被采用[PIXA88；UPST89]，该系统提供了这样一种明暗处理语言的场景描述规范。RenderMan定义了绘制过程中一系列的关键点，在这些关键点可调用用户或系统定义的明暗处理模块。例如，最普通的一种明暗处理模块称为表面明暗处理模

块，它对于给定的光源和物体表面上的一点及其方向，能够返回指定方向的反射光。因而一个用户提供的表面明暗处理模块可以实现一种完全不同于至此所讨论的所有光照方程的新形式。其他明暗处理模块包括大气明暗处理模块，（它可以改变通过两点间的光线的颜色）和（另一个体现明暗处理模块概念的延伸的例子）投影明暗处理模块，该明暗处理模块允许用户自己定义投影，以实现不同于平行投影和线性透视投影的新投影形式。

3. 一个实例

Cook、Carpenter和Catmull的Reyes绘制体系结构[COOK87]（用于绘制彩图II-24至彩图II-37、彩图D和彩图F）提供了一个怎样构造绘制系统的有价值的例子。Reyes将所有物体分割成为小的、固定明暗的、每边尺寸大约是1/2个像素的微多边形。这种方法，被称为切割(dicing)，沿着物体的自然边界进行的，如一个面片被沿着平行于它的坐标系（s, t）方向切割。 811
切割是在透视变换前，基于对投影后微多边形的大小的估计进行的。15.9节的Catmull面片细分算法中，细分面片直到它们成为像素大小，Reyes与之相似也将物体细分到足够小。每一种物体都有一个与之相关联的过程以决定是否对它进一步细分为其他图元或进行分割。如果没有任何办法可供对其直接切割，或者切割将导致太多的微多边形，或者微多边形的最终投影大小差别太大，则对物体进一步细分。这一递归的细分最终必须产生可直接对其进行切割的物体。为了避免用解析法对物体相对于视见体进行裁剪，当Reyes进行物体细分时，只有那些至少有一部分在视见体内的物体被保留下来。对由于物体距离太近或位于眼睛后面而产生的透视投影问题，可以通过对离眼较近的平面上的物体进行进一步的细分来避免。

切割一个物体将产生这些微多边形的四边形网格，这些微多边形在世界坐标系中进行明暗处理。因为微多边形足够小，每一个可以赋给一个明暗色调值，避免了16.2.6节所讨论的明暗插值。既然面片是沿平行于它的坐标系(s, t)进行切割，那么可高效地使用第17章中讨论的一些纹理映射技术。切割和明暗处理都可以利用递增算法。Reyes的全局光照效果依赖于本章介绍的映射技术。

通过一个子像素z缓存实现可见面判定，并通过抖动这些子像素的中心以完成随机采样。覆盖子像素中心的最近的微多边形是该子像素的可见微多边形。为了避免为整幅图像存储微多边形网格、子像素z以及亮度，Reyes使用了空间划分方法。图像被划分为长方形区间并在其间根据物体的左上角进行排序。分区以从左到右、从上到下的顺序处理。在物体被细分或切割的过程中，所产生的子物体或微多边形被置于与它们相交的分区中。因此每个分区只需要足够的z缓存空间，而分区所需的其他空间则在其处理后被释放。

16.14.4 逐步求精方法

考虑到我们观察画面的时间有限,可以对我们所讨论的流水线做一种有价值的改善。我们不再一次就生成一幅画面的最终图像，而是首先较粗略地绘制该图形画面，然后逐步求精完善它。例如，初步的图像可以没有反走样、简单的物体模型、简单的明暗处理。当用户观察这一图像时，可以在空闲的时钟周期来改进质量[FORR85]。如果有规则决定下一步怎样做，则求精过程可以自适应地发生。Bergman、Fuchs, Grant 和Spach[BERG86b]已经开发了这样一个系统，用不同的启发式规则来决定怎样安排时间。例如，仅仅当顶点的亮度值超过一个阈值时，才用Gouraud明暗处理来代替常量明暗值。光线跟踪[PAIN89]和辐射度[COHE88]算法都适用于逐步求精。 812

16.15 小结

在这一章中，我们遇到了很多不同的光照模型，一些是为了效率而提出的，而另一些则试

图考虑光线和物体表面相互作用的物理机制。我们已看到了怎样把插值技术应用于明暗处理中，既能使需要进行光照方程计算的顶点数目降到最低，又能以多边形网格近似曲面。我们对照了局部光照方法和全局方法，局部法分离地处理表面点，并利用光源直接进行光照计算，而全局法支持环境中物体间的折射和反射。我们注意到，在这些方法中，有的用环境的完整几何描述来计算全局效果，而其他的则采用更简单的描述，例如反射图。

如我们在本章一直强调的那样，不同的光照算法和明暗处理对相同的场景和相同的观察规范能产生不同的图像。选用哪种算法取决于很多因素，包括绘制图像的目的。尽管通常情况下，为了效率而不得不牺牲真实感，但算法和硬件的改进很快会使物理上正确的全局光照模型的实时实施成为现实。然而，当效率不再成为问题时，我们仍然愿意生成一些没有纹理、阴影、反射或折射效果的图像，因为在一些情况下，这种选择仍然是传递给观察者所需信息的最好方法。

习题

16.1 (a) 在Phong光照模型中使用不同项$(\overline{N}\ \ \overline{H})^n$和$(\overline{R}\ \ \overline{V})^n$生成画面时，描述所生成的画面的不同。

(b) 证明，当图16-12中的所有向量共面时，$\alpha = 2\beta$。

(c) 证明在通常情况下这种关系并不正确。

16.2 证明：沿多边形各边和扫描线插值顶点信息的结果，其结果在三角形的情况下，与方向无关。

16.3 假设存在多边形A、B、C，以与观察者距离递增的顺序与同一投影线相交。证明：如果多边形A和B透明，为它们投影的相交区域上的像素计算的颜色依赖于对公式(16-25)求值时多边形A和B的计算顺序（A和B是作为多边形1和2还是作为多边形2和1）。

16.4 考虑使用纹理映射来修改或代替不同的材质属性。列出所有您认为可以通过单独映射属性和混合映射属性所产生的效果以及如何消除由于纹理映射所产生的走样问题。

16.5 虽然使用反射图要求预先计算周围环境的光照，但是可以由记有物体标识和物体表面法向的反射图代替。请叙述使用这样的反射图有何不利？

16.6 解释如何使用反射图来模拟不同粗糙程度物体表面的反射。

16.7 叙述你认为可以通过推广应用Warn的挡板和锥体概念所能模拟的物体表面其他光照效果。

16.8 假设图16-64中显示的一组面片，继续重复增加面片5和6，并假设面片的辐射度值分别为$B_1 = B_2 = 2$，$B_3 = B_4 = 4$，$B_5 = B_6 = 6$。证明：B_h、B_e的值分别为5和3。然后证明B_b的值为1。
813 这是一个合理的值吗？注意从h到e保持线性。如果在面片行列中增加更多的面片，将发生什么情况？假设您以线ac为界增加这些面片的镜像多边形，并计算辐射度，那么，B_b的值将为2。这是否矛盾？解释您的结论。

16.9 根据15.10节和16.12节的内容实现简单的递归光线跟踪。

16.10 使用16.12.1节讨论的一些技术使习题16.9中的光线跟踪算法效率更高。

16.11 将习题16.10的光线跟踪扩充为分布式光线跟踪。

16.12 基于图16-70给出的辐射度算法的伪代码，实现辐射度逐步求精算法。用半立方体的方法计算形状因子。忽略子结构，而仅计算面片与面片的能量交换。同时，忽略环境光的计算以便于编程和视觉跟踪。通过验证所有Δ形状因子的和为1（近似）来检测你的半立方体算法。

为了显示你的图形，你必须实现多边形可见面算法（或许你的半立方体算法要用到）或使用现存的某个图形系统。如果没有明暗处理图形硬件，那么使用恒定明暗处理多边形将改进交互性（并将使编程和调试变得更容易）。

16.13 解释为什么图16-72给出的绘制流水线中的光照处理必须在裁剪之前进行。

16.14 实现一个局部光照模型的实验平台。将记有每个像素的可见多边形指针、该可见面法向、该可见面到视点的距离以及到一个或多个光源的距离和向量方向的图像存入一个材质属性表中。该平台中允许用户改变光照模型、光源的亮度和颜色以及物体表面的属性，并在每次改变时绘制画面。使用式(16-20)和光源衰减（式(16-8)）以及深度提示（式(16-11)）。

16.15 在你已实现的可见面判定算法中加入阴影算法。比如，如果已建立了一个z缓存算法系统，你可以将16.4.4节所讨论的两遍z缓存阴影算法增加进来。（如果您可以得到一个使用硬件z缓存的图形系统，那么很容易对后处理过程加以改变。解释如何利用习题16.14中为每个像素存储的特别信息来设计一个阴影后处理过程，以产生正确的明暗处理和高光效果。）

16.16 使用16.6节中描述的曲面表面的反射映射和平坦表面的映射方法，在可见面算法中增加物体间的反射的计算。

814

第17章　图像处理和存储

在这一章中，我们探索一些能有效地对图像进行处理和存储的方法。首先考虑常用的图像操作种类，请记住，我们正在处理的图像可能其本身就是最终所需的结果，也可能被用于产生后续的图像，就像在第16章中描述的环境映射那样。

我们立刻会想到几种图像操作，其一是将一幅图像叠放在另一幅图像上面，使它们融合起来，称为合成。合成的一个应用是在动画中，我们需要让一个人物在复杂的背景前移动，而背景保持不变。与每次都绘制背景不同的是，我们仅绘制一次背景，再绘制多帧人物在黑色背景上移动的图像，然后将这些图像逐一叠加在背景上合成起来，产生人物在背景上移动的效果。在这种合成操作中，为了使人物的轮廓在背景上没有锯齿形状，反走样就变得非常重要了。区分图像的背景和内容也是必要的，在这个例子中，人物在其上绘制的是黑色背景，人物本身是内容。

一般地，需要合成的两幅图像尺寸是不一样的，因此在合成之前，需要对它们进行平移、缩放和旋转。我们甚至需要使一幅图像变形，使其看起来像是一幅透视图，或者使其看起来像是画在一片橡皮上，然后进行拉伸一样。虽然绘制时可以采用几何变换来达到这些效果，但是这通常太困难或太耗时了，以至于根本不实用。事实上，如果图像是通过对照片进行光学扫描得到的，或者创建该图像的原始程序或参数丢失了，这样的绘制方法甚至是不可能的。

我们有时希望对图像进行各种滤波，用以产生伪彩色，模糊图像，或增强颜色、亮度的不连续性。这些滤波经常用于对卫星图像和CT数据（其中，一个点的亮度反映身体中组织的密度）的处理中。例如，亮度很小的改变可能预示着正常组织和癌变细胞的分界线，我们希望突出这些分界线。

图像实际上是很大的数据集合，一幅1024 × 1024的图像，如果每个像素使用n位表示，就会占据n/8 MB的存储空间（在一个每字节8位的机器上）。正如第4章所述，许多图形处理系统都有着巨大的内存用于存储图像（帧缓冲器）。如果图像存储器可以被其他程序访问，那么它可以用来作为一个程序的输出，然后作为另一个程序的输入，甚至它可以作为两个不同程序的输出。例如，当我们使用一个像素绘图程序去调整所绘制的结果图像中的单个像素时，这种情况就会发生。这种对图像存储器的使用（以及存储器严格的结构，它构成了一个被多种程序使用的数据库格式）被Blinn[BLIN85]称为“帧缓冲器协作”。

当一幅图像被存储于辅助存储器时，通常的做法是对被存储的数据（而不是图像）进行压缩。已经开发出来的方法有很多种。例如，如果图像中包含的颜色有大量的重复，第4章中所描述的查色表方法能有效地减少图像的存储空间。当然，只有用于显示图像的帧缓冲器支持查色表时，才能采用这种方法。在17.7节，我们将讨论几种更复杂的方法。

接下来，我们首先重新解释一下图像的含义，然后描述图像处理的一些最基本的操作：滤波和几何变换。然后，讨论如何为图像的每一个像素存储额外的数据，这些数据将用于图像的合成。再接着，我们讨论一下几种图像存储格式；最后，描述一些可以在图像层面上（而不是在建模或绘制时）产生的特殊效果。

17.1 什么是图像

如第14章所述，图像（在最基本的层面上）可以看成一个数组，数组的每个值代表了图像中每个像素点的属性（在位图中，值是单个的二进制数字）。这些数值经常是实数范围的定点表示，例如，我们常使用0到255的整数表示0.0到1.0之间的实数。通常，这个值代表了图像（灰度图）中一个点的亮度值或该点某彩色分量的亮度。数组的大小称之为图像的宽度和高度，数组的每个像素的位数称为图像的深度。

然而，通常我们并不仅仅把图像看成一个数组，而是用它表示一幅抽象图像，这个抽象的图像是关于一个连续变量函数，它在每个位置上都有值[⊖]。我们考虑的图像（有时也称数字图
816 像或离散图像）是离散变量的函数，对每一个数对$[i, j]$，有一个值被赋予像素$[i, j]$。如第3章和第14章所描述的那样，选择最好的离散图像来表示抽象图像是很困难的。本章中，我们会讨论重建抽象图像以便对其采新样。当然，我们并不是真正地重建，因为这意味着必须产生无穷多个点的值。但是我们可以重建出一幅抽象图像中任一单个点的值——实际上，我们能够重建需要采样的有限多个点的值。

如果我们通过对抽象图像采样生成一幅离散的图像（见第14章），然后从数字图像重建出抽象图像，那么原抽象图像与重建出的抽象图像可能会不同。如果原抽象图像不含有高频成分，那么重建出的图像会和原抽象图像一样。此时，我们就说重建出的图像是真实的。另一方面，如果原抽象图像含有高频成分，那么采样后的图像不能精确地表示它，重建出来的图像就会和原图有差别。

图像的另一个特点也很重要。虽然滤波理论告诉我们，如何才能最准确地用离散的图像来表示抽象图像，但是它基于这样的假定，即抽象图像上点的值是实数，而且数字图像上点的值也是实数。然而，在位图中，图像像素是二值的。在更复杂的像素图中，值有可能是小的二进制数（例如，每像素4位），或者在一个很大的数集中连续变化。这种值的离散化会导致图像处理上的很多严重问题，例如，怎样才能最好地压缩一个位图？如果4个像素——2个黑色和2个白色——将被压缩成1个，那么这1个像素应该是黑色的还是白色的？值的离散化会引起许多结果，我们将讨论其中已知的并且重要的几个，但是值得注意的是，仍然有很多结果我们尚不知道。

17.2 滤波

假设我们对产生的图像不进行反走样处理，例如，用光学扫描仪在一个点阵上采样一幅图画，然后将结果读入存储器，我们如何让这幅画看起来更好？这幅图像中肯定会出现我们不希望的锯齿现象。对某些原图像来说，我们能创建的每一幅图像都是正确的（这里，正确的含义是，在低通滤波之后，我们能够准确地表示原图像的一个采样）。如果某种方法改善了一幅图像的质量，同样的方法有可能会毁掉别的图像。因此，我们将要描述的方法应该只用于需要光滑处理的图像。如果锯齿状本来就是图像内容的一部分，那么它应该保留在那里，后置滤波只会使图像变得模糊。（最终，一幅楼梯的图像看起来应该怎样？）

假设我们想使图像光滑、没有锯齿，怎么做呢？一个简单的方法是使用每个像素和周围像素
817 的平均值来代替这个像素本身，这个过程称为后置滤波。它适用于离散图像，而不适用于抽象图像。使用后置滤波，阶梯附近的像素被融合了，从而消除了阶梯形状，如图17-1所示，滤波的效果被放大了。就像我们在第14章中看到的那样，这正是盒式滤波器产生的效果。更复杂的滤波器

⊖ 我们真正讨论的是一个函数，它的定义域是欧氏平面上的一个矩形，而不是该平面上的离散网格。

能够产生更好的效果。在讨论其他滤波器之前，让我们先来看看这种简单滤波方法的缺点。



图17-1 阶梯被盒式滤波平滑了

如果我们对一幅尖桩篱栅的图像进行点采样，尖桩以及它们之间的空隙宽度相等，尖桩是白色的，背景是黑色的。尖桩这样来放置，使得每9个尖桩及它们之间的空隙共占据10个像素的位置。采样出来的图像是怎样的呢？如果第一个像素正好落在第一个尖桩的最左端，那么第一个像素是白色的，紧接着的5个像素是黑色的，接下来的第6个到第10 个像素将落在尖桩上，因而是白色的，接下来的5个像素是黑色的，以此类推（如图17-2所示）。

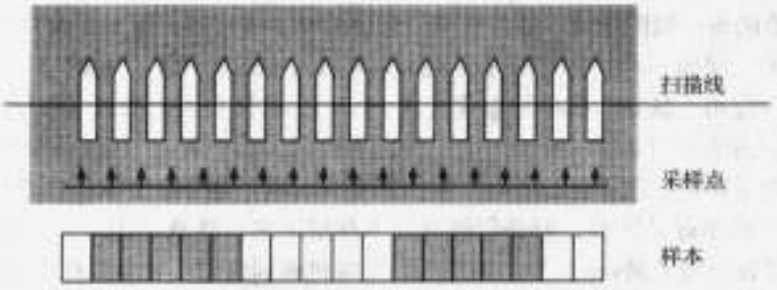


图17-2 一个采样图中的走样

现在，我们来看看前面所描述的后置滤波对这种情况会产生什么样的效果呢？它只能使第6个像素和第7 个像素之间变得光滑，图中仍然是大块的黑色区域连接着大块的白色区域。它不可能解决图像中隐含的所有问题。很明显，后置滤波对于走样来说不是一个很好的解决办法。另外，由于后置滤波会使图像中的其他边模糊（甚至是本来应该存在的），导致结果图像很不好看。

这个问题采用缩小图像的办法可以部分地解决：假想原图像被一个网格覆盖，网格的每一个小方块包围了4个像素，取4个像素的平均值作为目标图像中一个像素的值，这样我们就将一幅$2n \times 2n$的图像缩成$n \times n$的图像了。结合后置滤波，我们在间隔的扫描线上间隔地选取像素。注意，这里所涉及的滤波计算较少，我们仅需要对最终图像中的像素进行滤波。也就是说，仅对输出图像中出现的像素进行滤波，计算原图像中相应点周围像素的加权平均值。当然，最后的图像只有原图像的1/4大小。

回忆一下第14章的分析，我们来看看这种方法是如何工作的。如果原始图像采用2ω的频率进行采样，那么原图像中任何频率低于ω的信号将被精确表示，对于频率大于ω的信号（假设为$\omega + \phi$），采样得到的信号在频率为$\omega - \phi$处发生走样。如果用宽度为2的盒式滤波器对采样后的信号滤波，它能大量地（但不是完全地）过滤掉信号中频率高于$\omega/2$的成分（因为盒式滤波器的傅里叶变换是sinc函数，当频率增大时，它衰减的非常快。）在相间的像素上进行重采样，能够获得的有效采样频率是ω；在这个采样率下，所有频率低于$\omega/2$的信号都能被精确表示。但是进行滤波之后，所余下的频率也就是这么多了。如果原始信号中存在频率为$\omega + \phi$的成分，并且ϕ非常的大（大于$\omega/2$），那么这些成分的走样将发生在频率小于$\omega/2$的部分，从而永久地存在于最终的图像中。但是对于ϕ较小的信号，其走样将发生在频率大于$\omega/2$的部分，此时，通过超采样和后置滤波能减少走样。

我们知道，像直线段、矩形或者其他几何图形都具有明显的边界，在边界处存在任意大的频率，所以这种方法无法准确地表示它们。我们最多能用后置滤波来提高一幅极为糟糕的图像的质量，而这也是以图像变得模糊为代价的。

其他的滤波器（如sinc滤波器、Catmull-Rom滤波器、三角形滤波器等）都能产生比纯粹的盒式滤波器更好的效果。第14章中对这些滤波的分析在这里也适合。作为一个方便的经验准则，Whitted建议，虽然对一幅高分辨率图像进行滤波会导致明显的模糊，但是如果对一幅2048 × 2048大小的图像进行后置滤波，然后采样成512 × 512的图像，通常能得到较好的结果[WHIT85]。

现在，我们考虑时域上的一个类似问题：在一个关于运动马车的动画中，一个轮子的辐条快速地转动，一次通过屏幕上一个像素（这是对下面情况在时域上的类比：一个物体被划分成条状，并且着上快速变化的颜色）。辐条通过一个点的频率远大于30次/秒——典型的视频记录速度。时域走样是这样一种明显的现象：辐条看起来静止不动或者反向转动。当然，我们已经习惯了在电影中看到这种现象。但是，在电影中，我们实际看到的是模糊的反向转动的车轮，因为在采集电影的每一帧图像时，摄像机快门开启的时间非常短（但不是无穷小），大约只有分配给该帧图像的时间的一半，余下的时间被用来将胶片向前转动了。在时域上，快门有效地将一个盒式滤波器应用于场景。模糊是盒式滤波引起的，走样是因为滤波器很窄。所有的盒式滤波器加起来仅仅占电影时长的一半——余下的因为快门关闭而丢失了。这种影响对计算机图形学是明显的：为了获得电影质量的动画，我们需要在时域上做（至少）盒式滤波——*前置滤波*。后置滤波消除一些不好的影响，但是仍然有许多存留下来。注意，为了得到真正准确的图像，我们实际上应该在时域上做sinc滤波。如果电影摄像机做到这点（或者具有更宽的盒式滤波器），马车的轮子看起来就与生活中一样了——它们看起来是向前转动的，并且有一定的模糊。

818
~
819

17.3 图像处理

现在来看另外一个问题：怎样增强或减弱图像的某些特征？这个问题事实上属于*图像处理*领域，而不是计算机图形学领域，但是讨论一些相关的基本思想是值得的。通过扫描图像并检查相邻区域像素颜色值的改变，我们可以进行*边缘检测*和*边缘增强*。在相邻点的值相差足够大的地方，我们可以使这种差别进一步增大。如果图像中有*噪声*（也就是说，如果对像素值加入了随机扰动），那么可以通过前一节讨论的滤波技术*平滑*掉噪声。如果噪声是随机的，那么滤波（计算相邻像素的平均值）的确可以去掉噪声，最起码可以将噪声的高频部分滤掉。

另外一个图像处理的常用技术是*取阈值*，它用来突出图像中特定颜色值的点。例如，在一幅灰度图像里，可以这样来做，将低于某一个颜色值的像素变成是黑色的，高于某一个颜色值的像素变成是白色的，以便于明显地区分出它们的边界。第20章中讨论的Marching-cubes算法给出了另外一种取阈值的机制（在三维空间中）：它显式地将两个区域的边界构造成一个曲面（或在二维空间中，是一条曲线），然后采用适当的反走样方法绘制边界，得到一个更光滑的边界。关于该内容的更多信息请参见[GONZ87；SIG85]。

17.4 图像的几何变换

假设我们想对一幅图像进行几何变换，如平移、旋转、放缩以及其他非线性操作，我们该怎么做呢？

只有当一幅图像被作为另一幅更大的图像的子图像时，考虑对它进行平移才是有意义的。假设我们要平移一块$n \times k$的像素块（*源*），其左上角的坐标是(a,b)，平移后左上角的坐标是

(c,d)(目标)。这个变换很简单，我们只要将像素从原位置拷贝到新的位置，并且（如果我们愿意的话）用背景色覆盖所有不在目标区域内的原图像区域就可以了（如图17-3所示）。注意，必须使拷贝按适当的顺序进行，以免当源区域与目标区域发生重叠时覆盖掉还未拷贝的图像。假定a、b、c、d都是整数，这个方法工作的很好。

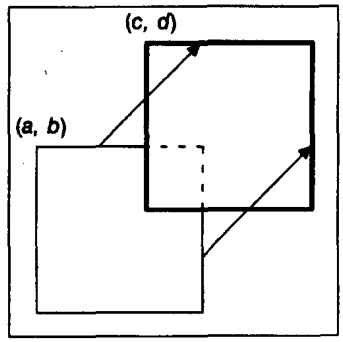


图17-3 图像的简单平移

但是如果起始位置和终止位置的坐标都不是整数怎么办呢？我们就需要重新构造源图像的抽象图像，对它进行平移，然后对平移后的图像采样。按部就班地这么做是不可行的——我们当然不希望在每一个可能的位置重构抽象图像，然后仅仅从中（几乎无限多个点）选择少数几个点。对于缩放和旋转变换也存在类似的问题。然而，人们已经开发出了一些算法来完成这样的操作，这些算法在计算方面将是正确的。Weiman开发了一些算法，可以对图像缩放和错切一个有理数因子[WEIM80]。旋转可以通过上面一些算法的巧妙组合来实现（见习题17.1）。习题17.2描述的是如何找到一个类似的平移算法，能将图像平移任意有理数。

17.4.1 基本几何变换

Weiman指出，一幅灰度图像以以下形式代表了一幅抽象图像：抽象图像被分成许多方块（一个方块代表一个像素），此方块中的平均颜色值就是这个像素的值。他于是假定，可以通过画一幅只包含灰度方块的图真实地进行图像重建，其中每个方块的灰度就是像素值。对于一个放大因子是p/q的缩放变换来说，只要把原来的q列变为p列，然后对图像进行区域采样就可以重构这幅图像了。滤波理论告诉我们，这种对采样结果图像的属性以及相应放大算法所做的假定从任何角度说都是错的：一幅抽象图像中高于Nyquist频率的成分不可能被采样到，因此，一幅从采样图像重建出的抽象图像不包含高频的成分。一幅包含相邻方块（它们的值不相等）的图像是具有许多高频分量的图像的极好例子，从而，上面所描述的显然不是一个好的重构方法。最后，滤波理论还指出，当把这样一幅图像转化为像素图时，我们应该采用sinc滤波，而不是盒式滤波。然而，如果Weiman的假定是合理的，那么他实现这些线性变换的算法是相当巧妙的。同时，它也是一个非常好的位图缩放变换的基础（见习题17.3）。

假设我们想对一幅图像进行缩放，缩放因子是p/q（这里，$p>q$，p和q是互素的整数）。第一步就是为p/q生成Rothstein码[ROTH76]。Rothstein码是一个二进制数列，它描述了一条斜率是q/p的直线（任何一个扫描转换直线段的算法都可以用来产生类似的代码）。图17-4显示了一条斜率是3/5的直线，其上标记了15个点。这条直线从左向右穿过了一些水平网格线，如果某一列中包含了直线和水平线相交的情况，那么就将此列标记为1，否则标记为0。每一列中包含三个标记，除非位于该列的三个标记中的某一个处于5的倍数的位置，否则将一个该列赋予0值，因为5的倍数的位置正好是直线段与水平线相交的位置。因此，第9个标记与第12个标记之间的列的值是1，因为标记10位于其间。（将位于列左端的标记看成是属于该列的，而右端的不是。）

820 ~ 821

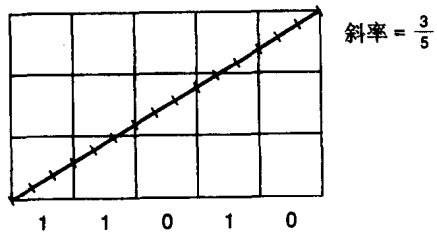


图17-4 一条斜率为3/5的直线段的Rothstein码

可以将Rothstein码看成这样一种机制：它将q个1均匀地分布在p个二进制位中。因此，它就告诉我们如何使源图像中q列均匀地分布在目标图像的p列中。但是，如果用Rothstein码中的1作为向哪里拷贝源图像的指示器，目标图像中的一些列将是空的。但是，Rothstein码可以循环排列，以给出源到目标的不同映射[⊖]，将得到的结果再做平均即可。

这个过程的伪代码如图17-5所示。

```
/* 将一幅图像的宽度放大p/q */
void WeimanExpansion(
        const grayscalePixmap source,      /* 源图像，尺寸是n×k */
        grayscalePixmap target,            /* 目标图像，宽度至少是k*p/q */
        int n, int k,                      /* 原始图像尺寸 */
        int p, int q)                      /* 缩放因子是p/q */
{
    char roth[MAX];                        /* 数组中至少能包含p个元素 */
    int i, j, s;                           /* 循环变量 */

    /* 源图像是n×k，目标图像是n×ceil(k*p/q)[⊖] */
    int targetWidth = ceil (k * p / (double) q);

    Rothstein (roth, p, q);                /* 将p/q的Rothstein码保存在数组roth中 */
    SetToBlank (target, n, targetWidth);   /* 清空目标数组 */
    for (i = 0; i < p; i++) {              /* 算法将循环执行若干遍 */
        int sourceCol = 0;
        Permute (roth);                    /* 对Rothstein码进行循环排列 */
        /* 对目标图像的每一列 */
        for (j = 0; j < targetWidth; j++) {
            if (roth[j] == 1) {            /* 如果Rothstein码指出要拷贝源图像列 */
                for (s = 0; s < n; s++) {  /* 拷贝所有的像素 */
                    target[s][j] += source[s][sourceCol];
                }
                sourceCol++;               /* 转到下一列 */
            }
        }
    }

    /* 因为源图像列一共加了q次，所以除以q */
    for (i = 0; i < n; i++)
        for (j = 0; j < targetWidth; j++)
            target[i][j] = rint (target[i][j] / (double) q);
}  /* WeimanExpansion */
```

图17-5 Weiman放大图像算法

如果缩放因子小于1，我们简单地对以上过程取反即可。p/q的Rothstein码告诉我们源图像中的哪些列应该出现在目标图像中（Rothstein码中的1表示要选中那一列）。同样地，我们最后取Rothstein码循环排列的平均值。

Rothstein码也可以用来描述错切变换。Rothstein码中的1表示源图像中相应的列应该向上平移一个像素，在q列中，一共平移了p个像素，结果纵向错切比例是q/p。同样地，我们应该

⊖ 对一个二进制数列循环排列即是对这个数列重复地做逻辑移位操作。

⊖ 一个数的ceiling是指大于或等于它的最小整数。ceiling(1.6)=2,ceiling(1.1)=2,ceiling(6.0)=6。

循环排列并对结果求平均值。

17.4.2 带滤波的几何变换

Feibush、Levoy和Cook给出了一个更复杂的图像变换算法[FEIB80]。(他们的算法主要用于为曲面进行纹理映射的算法中，详见第16章。)在详细讨论之前，我们先指出这个算法的优缺点。这个算法的优点在于它为边界像素计算出了合理的值：如果一幅图像被旋转了，某些目标像素仅仅被源图像中某个像素部分覆盖，该算法将这些像素作为特殊情况识别出来，并做了相应的处理。它使用一个加权滤波器作用于源图像，计算出目标图像的像素值。这样有助于减少结果图像的走样程度。但是，由于这个滤波器的宽度超过一个像素，如果用这种方法实现图像的恒等变换，将使图像变得模糊。在17.5.3节中，我们将详细讨论图像变换算法的一些缺点[⊖]。

一般来说，可以将图像变换分成两个方面。第一是计算出源图像中的哪一点将被映射到目标图像中的像素中心，第二是计算目标图像中这个像素的值。第一个任务仅仅是代数问题，它涉及求一个变换的值(和逆)。有很多增量方法可以有效地完成这个任务。第二个任务也有不少的解决方法，主要是围绕着选择一个滤波函数对源图像进行滤波。下面所描述的方法采用了这样的滤波函数，它是中心对称的并且大小适中(也就是说，它仅在平面上一个较小的区域取非0值)。

822 ~ 823

算法从一幅源图像(将它看成是位于二维欧氏平面上的，称为*源空间*)、从另一个二维欧氏平面(称为*目标空间*)到源空间的投影映射[⊖]以及目标空间中的一个多边形区域开始。目标图像是目标空间中的像素集合，它们近似组成了一个多边形区域，我们要计算的正是这些像素的值(见图17-6)。注意，这里的映射是从目标图像到源图像的，与通常对数学函数的命名约定相反。

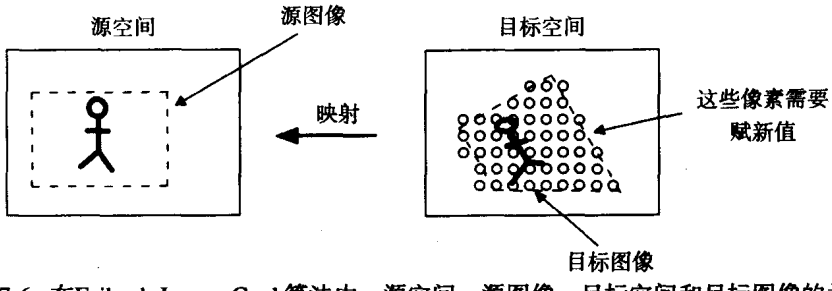


图17-6 在Feibush-Levoy-Cook算法中，源空间、源图像、目标空间和目标图像的关系

我们选择一个对称的滤波函数，只有当(x,y)非常靠近(0,0)时它的值才非0(一般约在2个像素之内)。这个滤波函数的*支撑集*是值为非0的点构成的集合。我们将包含支撑集的矩形平移到目标空间中的每一个像素处。只要矩形与目标多边形相交，这个像素就被看成属于目标图像的。这个被平移的矩形称为*目标像素的矩形包围盒*，而被平移的滤波函数的支撑集称为这个像素的*卷积屏蔽区域*(convolution mask)(见图17-7)。

目标多边形的顶点只变换到源空间中一次，被重复使用，所得到的结果多边形称为*源多边形*。每个目标像素的矩形包围盒都被变换到源空间中，变为一个平行四边形。计算出这个平行四边形的矩形包围盒，然后用源多边形对它进行裁剪(因为用源多边形对一个矩形裁剪比对一个平行四边形裁剪要简单)。源空间中所有位于裁剪结果区域中的像素都会被变换到目标空间中，并且只有落在目标像素的矩形包围盒中的像素才会被保留下来。

824

⊖ Feibush、Levoy和Cook指出，可以采用任何滤波器，但他们用直径为2的滤波器描述了这个算法。采用这个滤波器的算法通常比采用单位面积的盒式滤波器的算法效果好。

⊖ 投影映射在齐次坐标空间用3 × 3的矩阵表示，就像第5章中描述的那样。

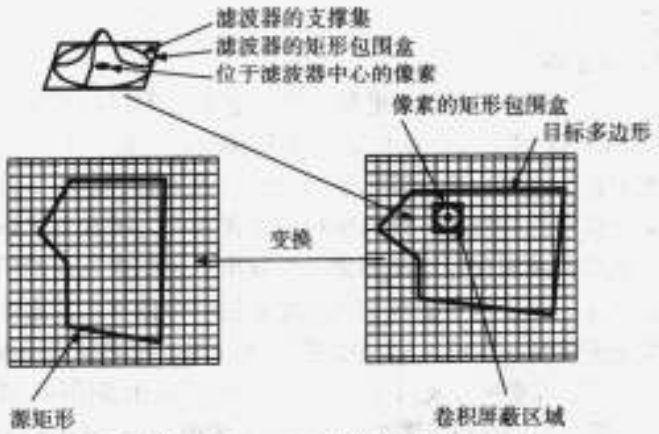


图17-7 Feibush-Levoy-Cook算法中用到的术语

经过变换的像素根据滤波函数进行加权平均，得到的值赋给目标像素。只有当整个像素落在变换后的图像边界之内时，目标像素的值才是正确的。例如，如果图像被旋转了，那么变换后的图像边界就有可能从像素中间穿过（更准确地说，穿过像素的卷积屏蔽区域）。

因此，目标像素并不完全由计算时用到的源像素值决定，那个值仅仅是对目标像素值有贡献，贡献的大小与该像素覆盖的区域面积成正比，该面积可以被精确地计算出来。图17-8显示了源图像经过变换后，它的边界穿过了一个像素的矩形包围盒，并且在该矩形内穿过了该像素的卷积屏蔽区域。为了计算出源像素值对最终目标像素值的贡献，我们进行如下步骤：

1. 用像素的矩形包围盒对图像多边形裁剪（见图17-9）。与矩形包围盒边界相交的点已经计算出来了，用于确定这个像素是否落在目标图像之内。

825 2. 对裁剪结果多边形（在图17-9中，是一个三角形，它的顶点按顺时针排列分别是A、B和C）的每一个顶点，用基准边（base）(绕多边形，按顺时针顺序连接这个顶点与下一个顶点的边)、侧边1（连接这个顶点和像素中心的边）和侧边2（连接下一个顶点和像素中心的边）构造一个三角形。

3. 将滤波函数看做是定义于卷积屏蔽区域之上的三维空间的曲面，那么，一个区域对像素的总的贡献正比于这个区域与滤波函数曲面之间的空间的体积。因此，我们现在来计算每一个三角形上方的体积。如图17-9所示，为了得到一个区域的正确贡献值，有些体积要被加上，有些体积要被减掉。加减的依据是，如果侧边1与侧边2的叉积指向书面里面，则相应的体积应该被加上，否则应该被减掉（见习题17.4）。

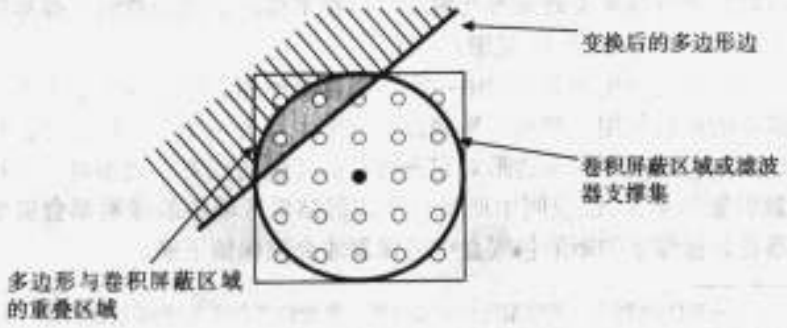


图17-8 在多边形的边界处，对一个像素进行滤波

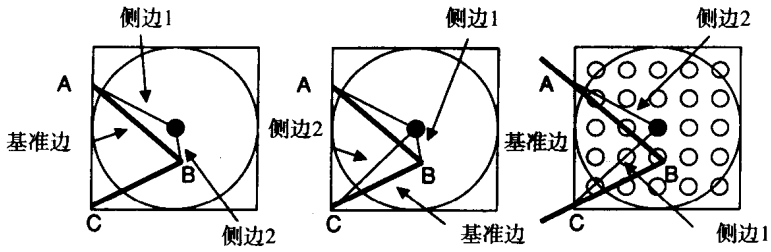


图17-9 滤波一条边的步骤

计算第3步中的体积比看起来要容易，我们可以预先将它们计算好了放在查找表里，在实际的滤波过程中只要从该表中检索出来就行了。习题17.5描述了如何做这样的预计算。

17.4.3 其他图案映射技术

Feibush-Levoy-Cook算法为向多边形上进行图案映射提供了优秀的结果，但是需要在每一个点上计算滤波后的值，从而，在图像中的每一个像素处都需要进行滤波计算。在一个平面的透视图中（收缩到一个灭点），一个像素也许对应源图像中成千上万个像素，因而需要极大的滤波计算。一些技术已经被开发出来，用于使滤波更快（可能准确性稍差一些）。

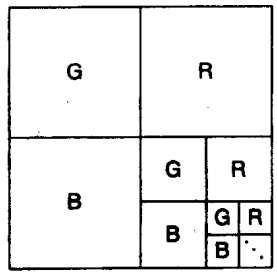


图17-10 一个MIP图。源图像的红、绿、蓝分量占据了MIP图的三个1/4区域，其中的每一个用因子为4的滤波器进行滤波，用三个结果图像填充余下的1/4区域的三个1/4区域。这个过程一直进行到MIP完成

Williams[WILL83]采用源图像创建了一个MIP图（multum in parvo——许多东西放在一小块地方），它占据的存储空间是源图像的4/3倍。如果源图像是512 × 512的24位真彩图像，红、绿、蓝每个分量占8位，那么，MIP是一幅1024 × 1024、深度是8位的图像。源图像中的红、绿、蓝分量各占据MIP图的1/4区域，余下的1/4区域被它们滤波后的图像占据，如图17-10所示。当一个目标像素被一组源像素覆盖时，MIP图中与该组像素最相近的像素即为所求的值。为了平滑所得到的值，可以在不同层之间进行线性插值。

Crow[CROW84]设计了一种方法，使得在相邻排列着的矩形区域上对图像进行盒式滤波很快。对快速图案映射来讲，在很多场合这是足够了的——与变换后的目标像素形状相近的矩形盒被用来计算像素的滤波值。这个方法基于代数恒等式$(x + a)(y + b) - (x + a)y - x(y + b) + xy = ab$。从几何上来解释，就是图17-11中的小的白色矩形的面积等于整个大矩形的面积减去水平阴影矩形的面积和竖直阴影矩形的面积，再加上方格阴影矩形的面积（该区域被减了两次）。通过从源图像创建一个新的图像，使得它的像素(x, y)的值等于源图像中角点为(0,0)和(x,y)的矩形内所有像素值的和，我们得到了一张*求和面积表*。现在我们可以计算角点为(x,y)和$(x + a, y + b)$的矩形内的像素和了，例如，它的值就是$S[x + a, y + b] - S[x + a, y] - S[x, y + b] + S[x, y]$。

Glassner[GLAS86]注意到，如果经变换后的像素与一个矩形不太相似，那么求和面积表可能会使结果非常模糊。于是他开发了一个系统，在这个系统中，矩形中对像素来说是多余的面积自动地被裁剪掉，从而给出了在该点处源图像滤波值的更准确估计。这需要检测与当前矩形区域相关的目标像素在逆变换下所对应的几何形状。

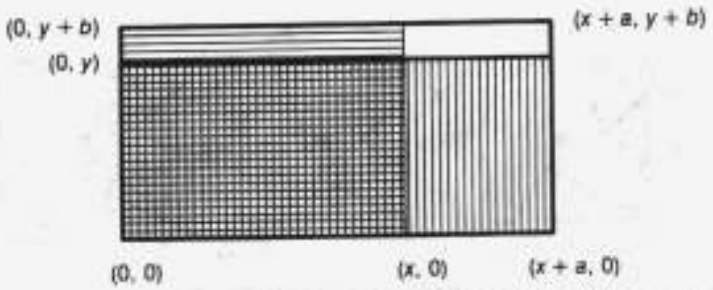


图17-11 图像中的小白色矩形的面积可以这样来计算，即从整个大矩形的面积中减去水平阴影矩形的面积和竖直阴影矩形的面积，再加上方格阴影矩形的面积

826
|
827

Heckbert[HECK86a]设计了一个系统，同时采用了Feibush-Levoy-Cook方法和MIP图。他将目标像素的支撑集（假定它是圆形区域并且由一个二次函数定义）映射到源图像中的一个椭圆形区域（由另一个二次函数定义）。根据这个区域在源图像中所占的面积，为源图像选择MIP图的合适层次，然后对椭圆区域内的像素做加权和。这个加权和即是目标像素的值。这个处理方法就同时具有了Feibush-Levoy-Cook技术的准确性和MIP图系统高效性。关于图案映射结果的比较如图17-12所示。

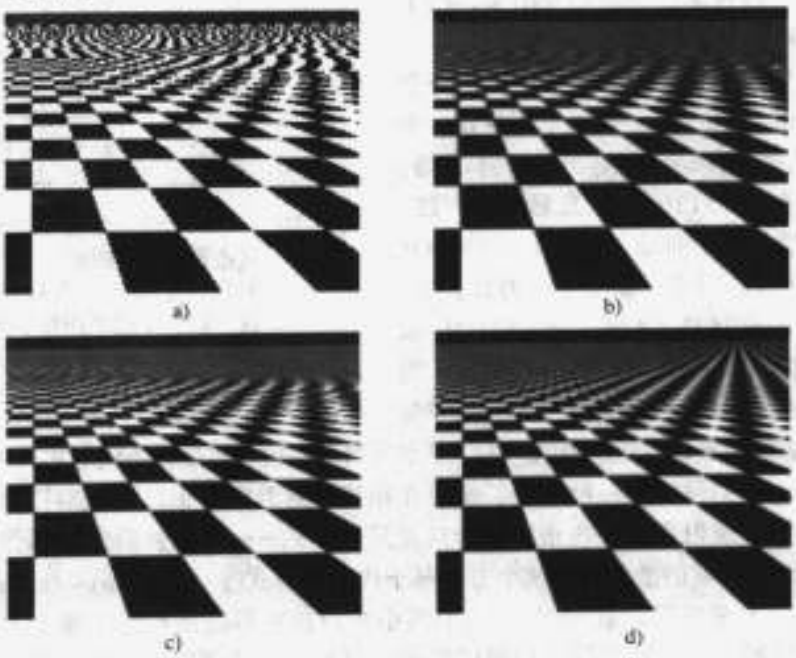


图17-12 a)源图像的点采样，b)MIP图滤波，c)求和面积表滤波，d)采用MIP图和一个高斯滤波器对椭圆形区域做加权平均。（P. Heckbert 授权。）

17.5 多重变换

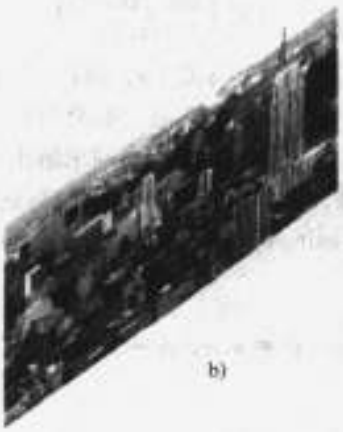
假设我们要对图17-13a中的图像进行垂直方向上的错切变换，如图17-13b所示，然后再施加一个水平方向上的错切变换[⊖]，得到17-13c中的图像。如果我们选择了合适的变换，那么最

⊖ 两个错切变换事实上是错切-缩放变换，第一个变换将一列像素平移，然后在垂直方向上压缩；第二个变换对一行像素做类似的操作。

终得到的结果是将图像旋转了[CATM80]。这种两重变换技术会比直接计算旋转变换快得多，因 828
为它每次可以处理一行像素或一列像素，而对在每一条线上像素的计算可以使用增量方法来完成。同样，在许多场合，为了避免走样而进行的滤波也可以一条线一条线地完成。更重要的是，很多变换都可以通过相继使用多个子变换来完成[CATM80，SMIT87]。这种多重变换技术已经在Ampex数字光学（Ampex digital optics machine, ADO）仪器中实现了[BENN84]，它被广泛地使用于视频制作过程中。关于该方法和其他一些图像变形技术的综述请参见[WOLB90]。



a)



b)



c)

图17-13 一个旋转变换可以表示为一个列保持变换和一个行保持变换的合成。a) 原图像，b) 对图像施加列保持变换之后，c) 将一个行保持变换作用于第二个变换。（哥伦比亚George Wolberg授权。）

两重变换（或多重变换）的实现过程可以被分解成为两个子任务：找出各重的正确变换，这是一个纯粹的代数问题；采用正确的滤波生成新的像素，这是一个反走样问题。既然第二部分依赖于第一个部分的结果，我们就先解决第一个问题，并且以旋转为例来加以说明。 829

17.5.1 多重变换的代数学

为了简化讨论，假设将使用三套不同的坐标。原始图像使用(x,y)坐标，垂直错切后的图像使用(u,v)坐标，最后结果图像使用(r,s)坐标。第一个错切变换记为A，第二个记为B，它们的组合变换是一个旋转变换，记为T。于是就有，

$$\begin{pmatrix} u \\ v \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix}$$

并且

$$\begin{pmatrix} r \\ s \end{pmatrix} = B\begin{pmatrix} u \\ v \end{pmatrix} = B\left(A\begin{pmatrix} x \\ y \end{pmatrix}\right) = T\begin{pmatrix} x \\ y \end{pmatrix}$$

从第5章，我们知道T的公式：

$$\begin{pmatrix} r \\ s \end{pmatrix} = T\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \end{pmatrix}$$

从这个式子，我们可以确定A和B的公式。

变换A是*列保持的*，也就是说，它将原始图像上每一列变换到结果图像的相应列上。因此，如果A将像素(x,y)变换到了(u,v)，那么，$u = x$。换句话说，对某个函数f，A必须是如下形式

$$\begin{pmatrix} u \\ v \end{pmatrix} = A\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ f(x,y) \end{pmatrix}$$

同样地，B是*行保持的*，所以，对某个函数g，B必须是如下形式

$$\begin{pmatrix} r \\ s \end{pmatrix} = B\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} g(u,v) \\ v \end{pmatrix}$$

为了确定A和B的公式，我们需要找到函数f和g。

将上面两式合起来，得到

$$\begin{pmatrix} r \\ s \end{pmatrix} = B\begin{pmatrix} u \\ v \end{pmatrix} = B\left(A\begin{pmatrix} x \\ y \end{pmatrix}\right) = B\begin{pmatrix} x \\ f(x,y) \end{pmatrix} = \begin{pmatrix} g(x,f(x,y)) \\ f(x,y) \end{pmatrix}$$

从这个等式中，我们可以看出s与$f(x,y)$是相等的，于是，s关于x、y的公式给出了$f(x,y)$的公式：$f(x,y)=x\sin\phi + y\cos\phi$。确定$g(u,v)$的公式要复杂一些。我们知道，如果用$x$、$y$来表示，我们可以写成$g(u,v) = x\cos\phi - y\sin\phi$。为了将它表示成$u$、$v$的形式，我们必须求解出$x$、$y$关于$u$、$v$的
830 公式，再进行替代。求解x是容易的，因为前面我们已经注意到$u = x$。求解y就更难一点：$v = f(x,y) = x\sin\phi + y\cos\phi$，于是$y = (v - x\sin\phi)/\cos\phi = (v - u\sin\phi)/\cos\phi$。将这个结果代入到$g(u,v)$关于$x$、$y$的公式中，我们得到

$$g(u,v) = u\cos\phi - \frac{v - u\sin\phi}{\cos\phi}\sin\phi = u\sec\phi - v\tan\phi$$

总之，如果我们定义

$$A\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ x\sin\phi + y\cos\phi \end{pmatrix}$$

和

$$B\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u\sec\phi - v\tan\phi \\ v \end{pmatrix}$$

那么，计算它们的复合式得到

$$B\left(A\begin{pmatrix} x \\ y \end{pmatrix}\right) = \begin{pmatrix} x\cos\phi - y\sin\phi \\ x\sin\phi + y\cos\phi \end{pmatrix}$$

正如所期望的那样。

对于一个任意的变换T，我们也必须做同样的工作。如果

$$T\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} t_1(x,y) \\ t_2(x,y) \end{pmatrix}$$

那么，我们定义$u = x$和$v = f(x,y) = t_2(x,y)$。为了定义$g(u,v)$，我们需要求解y关于u和v的表达式，

也就是说，需要找到一个函数h，使得$(u, v) = (x, t_2(x, y))$等价于$(x, y) = (u, h(u, v))$。如果我们找到了h，则$g(u, v)$的公式为$g(u, v) = t_2(u, h(u, v))$。

整个过程比较难的部分就是寻找h，事实上，在刚才的例子里，$h(u, v) = (v - u\sin\phi)/\cos\phi$，但是如果$\cos\phi=0$时（即$\phi=90°$或270°时），此函数没有定义，$h$是不可能找到的。幸运的是，旋转90°是非常容易的，只要将(x, y)变成$(-y, x)$就行了。因而这不成问题。实际上，我们将看到，如果要旋转一个接近90°的角度的话，最好是先旋转90°，然后再反向旋转一个小的角度。因此，为了旋转87°，我们可以先旋转90°，然后再旋转-3°。从代数学上讲，这两者没什么区别，但是从像素层面上讲，当涉及到滤波时，两者的区别很大。

一个旋转变换可以被分解成三个变换，以避免这个瓶颈问题[PAET86；TANA86；WOLB90]。关于旋转角度为ϕ的旋转变换可以分解为

$$\begin{bmatrix}\cos\phi & -\sin\phi\\ \sin\phi & \cos\phi\end{bmatrix} = \begin{bmatrix}1 & -\tan\phi/2\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \sin\phi & 1\end{bmatrix}\begin{bmatrix}1 & -\tan\phi/2\\ 0 & 1\end{bmatrix}$$

注意，每一个变换只需要执行一次乘法和一次加法。同时，如果$\phi>90°$，我们可以先旋转180°，然后再旋转$180° - \phi$，使得正切函数的参数永远不超过45°。[⊖] 831

为了说明多重变换技术并不局限于旋转变换，让我们来看另外一种变换，它将一个正方形变换为一个梯形。（这种变换在第6章描述的透视变换中出现。）作为一个例子，我们考虑

$$T\begin{pmatrix}x\\ y\end{pmatrix} = \begin{pmatrix}x/(y+1)\\ y/(y+1)\end{pmatrix}$$

像前面一样，我们希望找到函数

$$A\begin{pmatrix}x\\ y\end{pmatrix} = \begin{pmatrix}x\\ f(x, y)\end{pmatrix} \quad 和 \quad B\begin{pmatrix}u\\ v\end{pmatrix} = \begin{pmatrix}g(u, v)\\ v\end{pmatrix}$$

使得 $B\left(A\begin{pmatrix}x\\ y\end{pmatrix}\right) = T\begin{pmatrix}x\\ y\end{pmatrix}$。在这里，$v=f(x, y)=t_2(x, y)=y/(y+1)$。我们需要找到$g(u, v)$，使得$g(u, v)=x/(y+1)$。从$f$的方程中求解出$y$，我们得到$y=-v/(v-1)$。因此（记住$u=x$），得到$g(u, v)=u/(-v/(v-1)+1)=u/(-1/(v-1))=u(1-v)$。两个变换就是：

$$\begin{pmatrix}u\\ v\end{pmatrix} = \begin{pmatrix}x\\ y/(y+1)\end{pmatrix} \quad 和 \quad \begin{pmatrix}r\\ s\end{pmatrix} = \begin{pmatrix}u(1-v)\\ v\end{pmatrix}$$

容易知道，这两个变换合起来就是原始的变换T。

Smith和他的同事[SMIT87]将这一技术进行了推广，可以处理其他映射。使得平移、旋转、缩放和错切变换都很容易。另外，Smith考虑了如下形式的函数

$$T(x, y) = S(m(x)\, h_1(y), m(x)\, h_2(y))$$

其中，S是一个标准的计算机图形学变换（也就是，对一张平面的平移、缩放、旋转和透视变换）并且$m(x)$、$h_1(y)$和$h_2(y)$是任意的。他同时也考虑了映射T，它的构件函数$t_1(x, y)$和$t_2(x, y)$是x、y的双三次函数，在特殊的条件下，T是单射（也就是说，没有两个(x, y)点被映射到同一个(r, s)点）。

17.5.2 利用滤波生成变换后的图像

当图像经过了行保持变换（或列保持变换）后，源图像的像素（源像素）不见得正好被变

⊖ 正切函数在0°附近性质很好，但在±90°处有奇异性。因而，通常都愿意在0°附近计算它的值。

换到了目标图像的像素上。例如，某行中的一个像素有可能被向右平移了$3\frac{1}{2}$个像素。在这种情况下，我们必须将源图像中的几个像素值结合起来才能计算出目标像素的值。我们要做的是，将源图像上某一行像素的颜色值看成是一个函数在一条实线段（该行）上的采样，目标像素值是同一个函数的不同采样。因此，这个过程称为重采样。

理论上，对一个目标像素的理想重采样过程是对一些源像素做加权平均，而这些源像素经过变换后是落在目标像素的附近的。对应每一个源像素的权应该是sinc(kd)，其中d是变换后的
832 像素到目标像素的距离，k是某一个常量。不幸的是，这要求该行上的每一个源像素对每一个目标像素都有贡献。通常，我们并不直接使用sinc滤波器，而是用它的一些近似方法。最简单的是盒式滤波器：每一个源像素被看成是它所在行中的一个区段，变换作用于这个区段的端点。对目标像素的贡献等于变换后的区段与目标像素所代表的区段的重合部分与源像素值的乘积。

采用这种方法，每一个目标像素的值都是源图像上一个小区段（区段的长度依赖于所使用的变换）内像素的加权平均值。在旋转变换被两重分解情况下，(r, s)空间中的一个像素的值是(u,v)空间中一个水平区段内像素的加权平均。图17-14a显示了(r, s)空间中的一个像素，图17-14b显示了(u, v)空间中对这个像素有贡献的像素区段。同时，这些(u, v)像素中每一个又是(x, y)空间中一个竖直区段内像素的加权平均。图17-14c显示了(x, y)空间中的这些像素。注意，从(x, y)到(u, v)的变换是一个错切变换，所以这些竖直区段在(x, y)空间中形成一个菱形，而不是一个正方形。

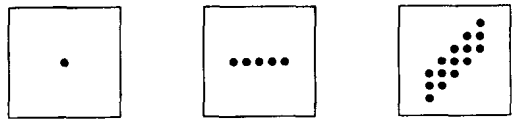


图17-14 在一个两重旋转变换中，对一个输出像素有贡献的所有像素在(x, y)空间中形成了一个小区域。a)(r, s)空间中的一个像素；b)(u, v)空间中对这个像素有贡献的水平像素区段；c)对(u, v)空间中的像素有贡献的(x, y)空间中的像素

从第14章我们知道，对输出像素有贡献的那些像素应该形成一个圆形区域[⊖]（也就是，滤波器应该是中心对称的）。如果菱形与圆形相差太远，那么这种滤波器的效果就会下降，产生较差的结果。这种将一个滤波器分解为两个子滤波器的方法所产生的效果在[MITC88]中有更深入的讨论。

另外，我们还想避免刚才所谈到的瓶颈问题，在这个问题中，源图像中的许多像素对每一个目标像素都有贡献。在错切-缩放操作的情况下，当缩放因子很小时，就会发生这种情况。

因此，当进行两重旋转变换（或其他多重变换）时，我们希望避免过大的错切变换或瓶颈问题。这就是为什么先旋转90°然后再旋转 -3°优于一次性旋转87°的原因。一般来讲，当我们构造一个两重变换时，我们先按行变换再按列变换，或者反过来；也可以在此之前先旋转90°。根据Smith的结果，在解决瓶颈问题时总可以采用这几种方法中的一种，至少对于标准的平移变换、旋转变换、缩放变换和错切变换来说是这样的[SMIT87]。但是，对一些更一般的变换，
833 使用以上技术可能没有什么效果：假设要将图像的一部分旋转90°，其他部分保持不动（例如，将一个很细长的矩形弯成一个四分之一的圆），那么，无论变换的顺序如何，在某些点肯定存在瓶颈问题。

Wolberg和Boult[WOLB89]开发了一种方法，使得一个多重变换的两个不同变换顺序能够一次完成。原始图像先经过一个行保持映射和一个列保持映射，然后旋转90°并颠倒映射顺序。

⊖ 特别是对旋转变换的情况。对一个一般的变换，对输出像素有贡献的源像素应该包含这样一些像素，它们被变换到目标像素中的一个小圆盘上，而它们在源图像中不一定构成一个圆盘。

对两种变换顺序，该方法分别记录了发生在每个像素处的瓶颈现象的程度。

于是，每个输出像素都可以通过两种不同的方法来计算。Wolberg和Boult为每一个像素选择了一个合适的变换顺序，使得发生瓶颈现象的程度较少，从而，图像的某些部分用行-列变换顺序，而其他部分用列-行变换顺序。因为这两套变换在并行处理器上可以同时进行，因此该技术非常适合于硬件实现。

17.5.3 评价变换方法

有几个评判图像变换算法的标准。滤波理论告诉我们，如果原始图像没有频率太高的成分，一幅图像可以由它的采样样本重建。实际上，通过任何一组样本，都可以重建出一些没有高频信息的图像。如果原始图像没有高频信息，那么我们就得到了原始图像；如果它的确含有高频成分，那么采样图像就发生走样，重建出的图像与原始图像可能有区别（见习题17.6）。

那么，我们怎么来评价一个变换算法呢？理想的情况是，一个变换应该有以下性质：

- 平移一个零向量应该保持不变。
- 一系列的平移变换应该和单个的复合平移变换效果相同。
- 以缩放因子$\lambda>1$进行缩放变换然后再以缩放因子$1/\lambda$进行缩放变换应该是一个恒等变换。
- 旋转任意角度，如果这些角度的度数加起来等于360°，那么应该得到一个恒等变换。

许多实际中工作得很好的算法显然不能满足以上任一个条件。Weiman算法只满足第四个条件。Feibush-Levoy-Cook算法在滤波函数的宽度超过一个像素时不满足第一个条件。Even Catmull和Smith的两重变换算法不满足所有四个条件。

其实这并不奇怪。为了对一幅图像进行重采样，我们首先应该使样本与sinc滤波器进行卷积，正确地重建出这幅图像。因此，每一个新的像素都应该是原始图像中所有像素的加权平均。由于所有的方法对使用宽度有限的滤波器都很敏感（影响计算速度），它们最终都使图像变得模糊了。

还有很多图像变换方法这里没有提到，它们每一个都有各自的优缺点，但大部分发生在时间-空间的折衷意义上。这个问题在Heckbert的一篇非常优秀的综述 [HECK86b]中有详细的描述。 834

17.6 图像合成

这一节中，我们讨论如何合成图像，也就是说，将图像合成在一起创建一幅新的图像。Porter和Duff[PORT84]认为，一般说来，合成是一个产生图像的好方法，因为做起来很容易，而绘制图像的各个部分可能会比较困难。通过合成，当图像的一部分需要改变时，不用重新生成整幅图像。更重要的是，当图像的某些部分还没有绘制但是已经被扫描到内存里时，合成可能是将它们加入到图像中的惟一办法。

在17.6.1节中我们将讨论使用α通道进行合成，17.6.2节讨论用帧缓存硬件进行合成，17.6.3节讨论如何人工生成α值，17.6.4节讨论一个图像组装的界面。

17.6.1 α通道合成

在图像合成中我们可以进行哪些操作呢？在合成后的图像中，每一个像素都是从构件图像以某种方法计算得到的。在一幅重叠图中，除了其他一些值（通常是RGB或其他颜色信息）外，还必须给定前景图像透明度。合成图像中的一个像素的颜色值是通过前景图像和背景图像混合得到的，除非前景图像在该点是不透明的，在这种情况下，结果像素的值就取自前景图像。在两幅图像的混合中，结果像素值是两个像素的值的线性组合。在这一节中，我们介绍使用这种组合和透明度的Porter-Duff方法 [PORT84]。

假设我们有两幅图像，一幅是红色多边形，另一幅是蓝色多边形，它们都在一幅透明的背

景上。如果我们将这两幅图叠放在一起，使红色多边形在前，那么，落在前面的多边形内部的所有点是红色的，而落在前面多边形外部但是落在后面多边形内部的点都是蓝色的。但是，如果一个像素落在前面多边形的边界上同时又在后面多边形内部，结果怎样呢（见图17-15）？这里，前面的多边形只覆盖了该像素的一部分，如果用红色显示它的话，就会产生走样。另一方面，如果我们知道前面的多边形覆盖了像素的70%区域，我们就可以用70%的红色和30%的蓝色混合得到一个更有吸引力的结果。

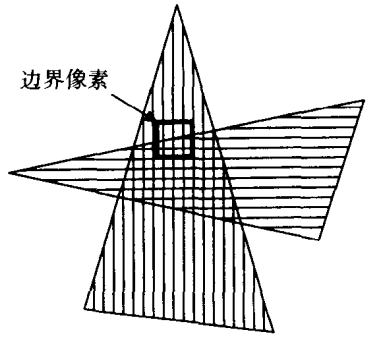


图17-15 边界附近的合成操作：我们如何对这个像素着色？

假设需要生成一幅图像，首先要记录覆盖信息：给定与图像中每个像素相关的颜色一个α值，用来表示像素的覆盖率。对于一幅前景图像来说，其中的许多像素的覆盖率为0（它们是透明的）；而对于其他像素，它们是前景图像的重要内容，其覆盖率通常大一些。

为了以一种合理的方式来合成图像，我们需要图像中每一个被合成像素的α信息。因此我们假定，像素除了RGB颜色值之外，还有表示像素覆盖率的α值。这些α值通常被称为α通道（见17.7节）。一些绘制程序能很容易地产生覆盖率信息，但对已经扫描到存储器中图像来说，这就更困难。我们将在17.6.3节中简要讨论这个问题。

怎样使用α通道来合成图像呢？假设有一个红色多边形覆盖了一个像素的三分之一区域，一个蓝色的多边形覆盖了这个像素的二分之一区域。第一个多边形被第二个多边形覆盖了多少呢？如图17-16所示，第一个多边形有可能被第二个多边形完全覆盖、部分覆盖或者完全没有覆盖。但是，假如我们完全不知道这些信息，那么如何合理地估计第一个多边形被第二个多边形覆盖的程度呢？让我们假定，第一个多边形在像素内所覆盖的区域是随机分布的，第二个多边形也一样；那么，像素中任何一个小点落在红色多边形内的机会是1/3，落在蓝色多边形内的机会是1/2，同时落在这两个多边形内的机会是1/6。注意到这个值正好是1/3的一半，因此，第一个多边形的一半会被第二个所覆盖。下面是我们的一般假设：像素被多边形覆盖的区域是随机分布的，所以第一个多边形被第二个多边形覆盖的比率（在一个特定的像素内）等于整个像素被第二个多边形覆盖的比率。在实际中，这种假设所导致的结果是，当两幅图像同时具有细节信息时，合成后的图像很糟。然而，Porter和Duff说，他们没有这类严重的问题[PORT84]。

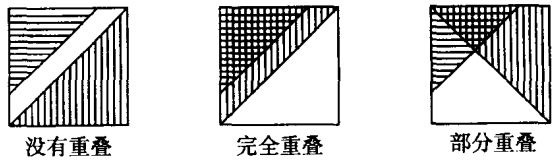


图17-16 在一个像素内，多边形重叠的方式。在图像合成时，前两种情况被看成是例外，第三种情况被看成是常规的

现在，如果按照60-40的比例混合这两个像素，我们怎么计算这个像素的颜色值呢？既然一种颜色覆盖了这个像素的1/3（如果完全覆盖的话，将产生颜色(1, 0, 0)），该颜色对这个像素的贡献就是(1/3)(1, 0, 0)。我们希望将这个结果的60%与另一个结果的40%混合。于是，我们按照如下方式混合了红色像素的RGB三基色(1, 0, 0)和蓝色像素的RGB三基色(0, 0, 1)：

835
~
836

$$0.6\left(\frac{1}{3}\right)(1,0,0) + 0.4\left(\frac{1}{2}\right)(0,0,1) = (0.2, 0, 0.2)$$

这就是结果颜色。

注意，无论何时，当我们合成两个像素时，我们使用α值和像素颜色值的乘积。这就意味着，当我们存储一幅图像时（在一个图像合成程序内），我们应该保存的不是（R, G, B, α），而是（αR, αG, αB, α），这样可以避免在每一次使用这幅图像时都要进行乘法运算。当我们提到一个像素的RGBα值时，我们所指的就是上面这个值。因此，这个结论总是正确的：一个像素的R、G、B分量不大于它的α分量[⊖]。（在很少的情况下，我们可能希望考虑使该条件不成立的像素，这样的像素是发光的。）

假定现在有两幅图像A和B将被合并，又假定我们考虑P这个像素。如果图像A中P像素的α值是α_A，图像B中P的α值是α_B；我们可以问，结果像素的多少是只被A覆盖的，多少是只被B覆盖的，多少是同时被两者覆盖的，多少是两者都没有覆盖的。

我们已经假定了被两者同时覆盖的部分的面积是$\alpha_A\alpha_B$，这表示，$\alpha_A - \alpha_A\alpha_B$只被$A$覆盖，$\alpha_B - \alpha_A\alpha_B$只被$B$覆盖，剩下的部分为1减去上面三者之和，等于$(1-\alpha_A)(1-\alpha_B)$。在合成的图像中，仅被$A$覆盖的区域的颜色可能是$A$，也可能没有颜色；颜色$B$的情况类似。既没有被$A$覆盖也没有被$B$覆盖的区域没有颜色，而同时被两个覆盖的区域可能没有颜色，可能颜色是A，也可能颜色是B。表17-1列举了这些可能性。

表17-1 面积与合成时重叠区域可能的颜色

区 域	面 积	可能的颜色
空	$(1-\alpha_A)(1-\alpha_B)$	0
A	$\alpha_A(1-\alpha_B)$	0, A
B	$\alpha_B(1-\alpha_A)$	0, B
A和B	$\alpha_A\alpha_B$	0, A, B

那么对这个像素来说，到底有多少种可能的着色方法呢？同时被A和B覆盖的区域有三种可能性，仅被A或B覆盖的区域有两种可能性，空的区域只有一种可能性，所以我们总共有12种。图17-17列出了这些可能性。

在这些情况中，最直观的（也是最常用的）是A覆盖B、A在B内和B裁剪A，它们分别表示了如下情况：A将B隐藏在背后；只显示A中落在B内的部分（当B是一个有洞的图像时，这种情况就会出现）；只显示A在B外的部分（如果B是一个框架窗口，这就是伸出到窗口之外的那部分）。

在每一种情况下，我们都可以像前面一样计算出每一种颜色对最终结果贡献的比例。例如，对于A覆盖B的情况来说，图像A中的所有颜色都有贡献，而B图像中只有$1-\alpha_A$的颜色对最终结果有贡献。最终的颜色是$c_A + (1-\alpha_A)c_B$。其中c_A代表A的颜色（已经乘上了α_A）。

一般说来，F_A代表的是图像A中对应像素的颜色对结果贡献的比例，F_B也类似，那么最终颜色值应该是$F_Ac_A + F_Bc_B$。同时，我们必须计算相应的α值：如果F_A是A的显示比例，α_A是A原始贡献，那么考虑覆盖率，图像A的新的贡献应该是$F_A\alpha_A$。对B也是一样。因此，新像素总的覆盖率应该是$F_A\alpha_A + F_B\alpha_B$。

一些一元操作也可以作用于图像，例如，我们可以定义**darken**（变黑）操作，

$$\textbf{darken}(A, \rho) := (\rho R_A, \rho G_A, \rho B_A, \alpha_A) \qquad 0 \leqslant \rho \leqslant 1$$

⊖ 在用定点数表示颜色时，α值小的像素比α值为1的像素的颜色变化范围更大。Porter和Duff说，小α值引起的像素颜色范围不稳定对他们没有什么影响。

在保持同样的覆盖率的情况下，这个操作有效地使一个像素变暗。作为对比，**fade**（淡化）操作符定义如下，

$$\mathbf{fade}(A, \delta) := (\delta R_A, \delta G_A, \delta B_A, \delta \alpha_A) \qquad 0 \leqslant \delta \leqslant 1$$

该操作在保持颜色不变的情况下使一个像素逐渐变得透明（其颜色各分量都乘以δ，因为颜色值总是要预先乘以α值的）。

操作	四元组	图形	F_A	F_B
空	(0, 0, 0, 0)		0	0
A	(0, *A*, 0, *A*)		1	0
B	(0, 0, *B*, *B*)		0	1
*A*覆盖*B*	(0, *A*, *B*, *A*)		1	$1-\alpha_A$
*B*覆盖*A*	(0, *A*, *B*, *B*)		$1-\alpha_B$	1
*A*在*B*内	(0, 0, 0, *A*)		α_B	0
*B*在*A*内	(0, 0, 0, *B*)		0	α_A
*B*裁剪*A*	(0, *A*, 0, 0)		$1-\alpha_B$	0
*A*裁剪*B*	(0, 0, *B*, 0)		0	$1-\alpha_A$
*A*在*B*的上方	(0, 0, *B*, *A*)		α_B	$1-\alpha_A$
*B*在*A*的上方	(0, *A*, 0, *B*)		$1-\alpha_B$	α_A
A Xor *B*	(0, *A*, *B*, 0)		$1-\alpha_B$	$1-\alpha_A$

图17-17 合成操作的可能性。四元组指示了“空”、“*A*”、“*B*”、“二者都有”这四个区域的颜色。该结果从[PORT84]修改得来。（Thomas Porter和Tom Duff授权。）

很自然地，第三个操作符是（本质上是上面两者的复合）**opaque**（不透明）操作符，它只作用于α通道，定义为，

$$\mathbf{opaque}(A, \omega) := (R_A, G_A, B_A, \omega\alpha_A)$$

当ω在0和1之间变化时，像素对背景的覆盖率也发生变化。当然，如果ω足够小，可能会有一个颜色分量的值大于α的值。例如，如果ω为0，我们就得到一个像素，它的α值为0。这样的像素不能遮挡任何东西，但它对合成像素的亮度有所贡献，因此它被称为是发光的。颜色分量会大于α分量这种可能性要求我们在重新计算真实颜色值时，对颜色值进行裁剪，使其落在[0，1]区间

（为了重新计算颜色，我们计算（R/α,G/α,B/α），并调整颜色分量大小，使其最大值为1）。

最后一个一元操作符**plus**（加）很有用。在这个操作下，颜色分量以及α值可以相加。因此，为了从图像*A*平滑过渡到图像*B*，我们使用

fade(*A*, *t*) **plus fade**(*B*, 1 − *t*)

让*t*从1逐渐变到0。通过对以上这些操作进行巧妙的组合，我们可以对图像进行各种各样的合成。

现在我们来看一下前面做的假设所引起的结果：如果一个像素被一个多边形覆盖的比例是F_1，被另一个多边形覆盖的比例是F_2，那么第一个多边形被第二个多边形覆盖的比例也是F_2。如果两个多边形碰巧以一种合理的方式重叠，那么这个假设工作得很好，如图17-17所示。但如果两个多边形是穿过像素的平行细条，如图17-18所示，那么这个假设就不成立。当几何体被合并时，这个问题一般不会经常发生。然而，如果一幅图像被反复地叠加合成，这种情况就可能经常发生。

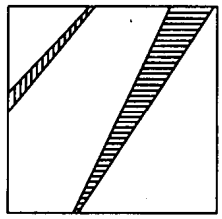


图17-18 在合成时，使重叠假设不成立的一种情况

837 ≀ 839

为了说明如何使用合成技术，我们来描述一下在第20章介绍的Genesis effect（已显示在彩图IV-14中）中帧的合成。在这个例子里，有四幅图像要进行合并：*FFire*，行星前面的粒子系统；*BFire*，行星后面的粒子系统；*Planet*，行星本身；*Stars*，背景星空。整个合成的表达式是[PORT84，p.259]：

(*FFire* **plus** (*BFire* **held out by** *Planet*)) **over darken** (*Planet*, 0.8) **over** *Stars*

行星用来遮挡粒子系统位于它之后的部分，所得结果叠加到前面的粒子系统上，这些再合成到有点变暗的行星上（从而粒子系统使得其后的行星变得朦胧），然后，结果叠加到背景星空。

17.6.2 其他合成方法

让我们来考虑一下其他两种合成方法。第一种通过简单的运算（*A* **over** *B*，*A* **and** *B*，*A* **xor** *B*）在压缩方式下合成图像（例如每个字节表示8个像素的位图）。这些运算非常容易实现：对任意两个字节，我们利用硬件对它们做位运算。如果将这个方法进行扩展，它还可用于处理采用行程编码位图的合成。

另一个合成方法利用硬件帧缓存硬件实现合成。考虑一个简单的例子，我们通常将每像素占3位的位图看成一个单个的图像，它的每个像素用3个二进制位来表示。但是，我们也可以认为它包含两幅图像，其中一幅图像的每个像素占2位，另一幅图像占1位；或者将它看成3幅图像，每幅图像占用其中的1位。无论在哪种情况下，都需要查色表将独立的图像组合起来，并形成最终的合成图像在观察平面上显示出来。例如，如果只显示由每个像素的最高位所决定的图像2，我们就使用图17-19中所示的表。为了显示由每个像素的最低位所决定的图像0，我们就使用这张表，对应低位是1的位置设置为7，其他位置放上0，得到序列：0,7,0,7,0,7,0,7。如果要显示的图像是这3幅图像的和，并且如果每幅图像在处于“开”状态的像素处对亮度的贡献是两个单位的，我们就使用图17-20中的表。如果一个像素的三个位中的一个为1，则在表中放一个2；如果有两个为1，则放一个4；如果三个都为1，则放一个6。

作为另一个例子，我们将每幅图像看成是定义于相互平行的平面上的，如图17-21所示。图像2所在平面距观察者最近，图像0所在平面最远，因此，图像2同时遮挡了图像0和1，图像1仅遮挡图像0。这种优先级可以在查色表中体现出来，如图17-22所示。在这个例子里，图像2以亮度7显示，图像1以亮度5显示，图像0以亮度3显示，以便距观察者越近，图像看起来越亮。

如果没有定义图像，用亮度0显示。

项　号 （十进制）	项　号 （二进制） 2 1 0 图 图 图 像 像 像	查色表的内容 （十进制）
0	0 0 0	0
1	0 0 1	0
2	0 1 0	0
3	0 1 1	0
4	1 0 0	7
5	1 0 1	7
6	1 1 0	7
7	1 1 1	7
		0 = 黑色 7 = 白色

图17-19　用来显示由每个像素的最高位所定义的图像的查色表

项　号 （十进制）	项　号 （二进制） 2 1 0 图 图 图 像 像 像	查色表的内容 （十进制）
0	0 0 0	0
1	0 0 1	2
2	0 1 0	2
3	0 1 1	4
4	1 0 0	2
5	1 0 1	4
6	1 1 0	4
7	1 1 1	6

图17-20　用来显示三个1位图像之和的查色表

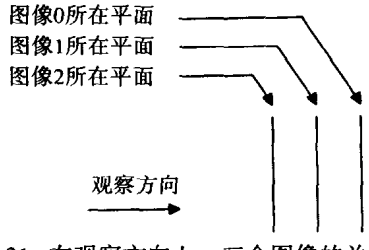


图17-21　在观察方向上，三个图像的关系

项　号 （十进制）	项　号 （二进制） 2 1 0 图 图 图 像 像 像	查色表的内容 （十进制）
0	0 0 0	0没有图像
1	0 0 1	3图像0可见
2	0 1 0	5图像1可见
3	0 1 1	5图像1可见
4	1 0 0	7图像2可见
5	1 0 1	7图像2可见
6	1 1 0	7图像2可见
7	1 1 1	7图像2可见

图17-22　用来对三个1位图像分配优先级的查色表

另一种可能性是用查色表来存储两幅图像亮度的加权和，产生两次曝光的效果。如果用于一个图像的权值随着时间变小，而另一个随时间变大，我们就可以得到淡入、淡出效果，也称为重叠-溶解（lap-dissolve）。当使用彩色图像时，在渐变过程中显示的颜色依赖于我们计算加权和时所采用的颜色空间（见第13章）。

确定如何计算查色表以便获得一些特殊的效果是一件乏味的事情，尤其是当图像数目很多并且查色表包含很多项时。图像合成语言（Image Composition Language, ICL）[FOLE87c]允许编程者将图像（由一个或多个位面组成）定义为变量。被显示的图像用一个合成表达式来描述，合成表达式中包含了算术运算、关系运算和条件运算。采用ICL，重叠-溶解可以用如下表达式来表示：

$$newImage * t + oldImage * (1 - t)$$

其中，变量*oldImage*和*newImage*都是位面上的图像，*t*是一个标量，在0到1的范围内变化。下面这个合成表达式将红色（大括号里的三元组是RGB颜色的表示）加入到图像*c*中颜色值在[0.6,0.8]范围内的部分，并将绿色加入到颜色值在[0.3,0.5]范围内的部分：

```
if (c < 0.8) and (c > 0.6) then c + {1,0,0}
    else if (c < 0.5) and (c > 0.3) then c + {0,1,0}
    else c
endif endif
```

这个简短的合成表达式代替了相当长的一个代码段。

17.6.3 通过填充机制生成α值

在前面的几节里，我们描述了通过α通道来合成图像。但是，如果一幅图像是通过扫描照片得到的，或者是从缺乏该信息的其他渠道得来的，怎么办呢？如果我们能为图像产生α通道，那么我们就能应用这个通道进行图像合成了。即使我们仅仅是将黑色像素的α值赋为0，其他像素的α值赋为1，我们也就可以应用前面的算法了，虽然这将引起边界不光滑的问题。

从第3和4章的内容我们知道，有很多方法能够用新的值来填充区域。如果我们采用一个填充算法不是为了改变像素的颜色值，而是改变它的α值，我们就可以给图像中不同的区域分配α值了。如果图像中的颜色代表前景和背景（例如，一个人站在白色墙壁前面的照片），我们可以用它的原始颜色填充背景区域，同时赋给它较小的α值。Fishkin和Barsky给出了一个算法，它能识别出与某些颜色完全或部分相同的像素组成的区域[FISH84]。这个算法的更详细描述请参见19.5.3节。

840 ~ 842

当然，如果背景不是一个连通的区域，应用种子填充算法就注定要失败。如果我们采用Fishkin-Barsky关于背景与图像中所有像素之间的相似性标准来试图克服这种困难，前景区域中颜色与背景颜色相近的像素可能会被错误处理。（想像前面那个例子中的人，他站在一面白色的墙壁前，他的绿色T恤衫上的白色图案是背景的一部分吗？）如果我们试图用种子填充算法填充每一个分离的背景区域，这个任务可能是没有希望完成的。（再考虑我们例子中的人，假定白色的墙壁从他的卷发透了过来，就会形成成千上万个背景区域。）但是，一个软种子填充算法在这方面迈出了可喜的一步，对简单的图像来说，它要比那些将所有像素的α值赋为0或1的方法改善了很多。

17.6.4 用于图像组装的一个界面

在实际应用中，进行图像合成和几何变换的工具是怎样的呢？像彩图IV-19中的图像是由一组非常复杂的操作描述的。诸如*A* **over** *B*这样的操作可以用一个树结构来描述，其中叶节点是图像，中间节点是操作符，而操作数是它的子节点。但是，在图像被合成之前，它们必须被正确地放置。这些要求导致了一个图像-组装树（image-assembly tree）结构，在这种结构中，每一个中间节点或者是图像变换或者是合成操作，每一个叶节点是一幅图像。

这种图像-组装结构可以用一个很方便的用户界面来实现，其中，用户用鼠标向树中增加或从树中删除一部分，也可以在某些节点处放置一个标记，以便观察由该节点和它的子节点所描述的图像。这个观察结果可以是结构性的，仅仅显示子节点在父节点中的相对位置和大小。在一个颜色个数有限的工作站上，可以对真彩图像进行颜色抖动再显示结果，在一个高档工作站上，结果可以全彩全尺寸地显示。这种结构性的观察会极其有用，因为用户通过拖动鼠标就能指定其中的几何变换，而不需要输入几何变换的精确坐标。当然，允许输入精确坐标的功能也是必要的。

Kauffman[KAUF88a]开发了一个这种类型的图像组装器。更复杂的图像组装器构成了视频处理器的核心，它能产生许多特殊效果，就像我们在电视上看到的那样。

17.7 图像存储机制

当我们存储一幅图像的时候，事实上我们存储的是一个二维数组，二维数组中的元素是赋予图像中一个像素的*数值*。对一个位图来说，这个数值是一个二进制位。对一个彩色图像来说，
843 这个数值也许是三个数的组合，代表一个像素的红、绿、蓝颜色分量，或者是红、绿、蓝三个查色表的索引值；这个数值也可能就是一个颜色三元组组成的查色表的索引值，或者是任何一个其他数据结构的索引值，只要这个数据结构能够表示颜色就行，包括CIE或XYZ颜色系统，它甚至可能是描述颜色的一组4个或5个光谱样本。

另外，每个像素可能还有其他信息，如该像素的z缓冲器值，在该像素处显示的表面的法向量，或者α通道信息。因此，我们可以将图像看成是由一组*通道*组成的，每一个通道给出了图像像素一种信息。从而，我们经常会提到一幅图像的*红色通道*、*绿色通道*、*蓝色通道*。

虽然这种做法看起来与好的编程实践相反，因为在好的编程实践中我们学习把与某个对象的信息组织起来形成单个的数据结构，但是将不同的通道分开通常有利于存储。然而，一些图像压缩方法的确把图像当成二维数组对待，如四叉树和后面要介绍的分形编码方法，在这些情况下，将图像分割成通道是不合适的。

在讨论将图像作为通道或数组进行存储的算法之前，我们介绍存储图像的两个重要方法：*利用元文件*和*利用依赖于应用的数据*。严格说来，这两种数据都不是图像格式，但它们都是传递图像中表示信息的一种机制。

如果一幅图像是通过对一组例程的一系列调用生成的，那么就用一个元文件来存储这一系列调用，而不是保存所产生的图像。这个调用序列可能比图像本身要简洁得多（日本国旗的图像可以这样来生成：先调用一个画矩形的例程，在调用一个画圆的例程，而这幅图像若要用RGB三元组存储则要占几兆子节。）如果例程足够简单，或者是用硬件来实现的，重现显示元文件可能会比重新显示一幅像素图更快。术语*元文件*也用来指一个与设备无关的标准数据结构的描述，如第7章中介绍的PHIGS数据结构。为了将一幅图像存储在元文件中，我们遍历当前的数据结构，将它们以某种与设备无关的格式记录下来，以便于以后的重新显示。这个描述可能不是一系列函数调用，而是一些分层结构的文本描述。

第二种存储方法必须采用依赖于应用的数据。如果一个应用显示一类特殊的图像，按照这些图像的创建方式去记录数据或者记录数据和一些标准数据集之间的差就可能非常方便。如果图像都是以多边形表示的人脸，仅仅存储那些与一些标准脸部图像多边形位置不同的多边形（当然，还有它们新的位置）可能更简单。这类信息压缩方法的一个更极端例子是MIT媒体实验室Talking Heads项目所用的方法，其中，仅仅存储眼球、嘴唇的位置等高层特征[BOLT84]。在这一点上，图像描述更像是一个场景描述，属于建模领域，而不属于图像存储的范畴。

17.7.1 存储图像数据

现在，让我们来考虑如何存储包含几个通道数据的图像。如果我们的显示器期望接收的图像信息是RGB三元组的形式，那么将图像保存成RGB三元组可能最为方便。但是，如果像通常那样存储空间是首要因素，就需要通过某种方法对各个通道进行压缩了。衡量压缩方法必须考虑到解压缩的代价：压缩技术越复杂，解压缩的代价可能就越昂贵。虽然所有这些技术对各个通道的信息都是同等对待的，我们将以颜色通道为例来进行讨论，因为它在图像中是最经常出现的（z缓冲器、法向量和其他信息是可选的）。 844

如果一幅图像包含很少的几种颜色，每种颜色出现很多次（就像报纸中的图像一样，其中也许只有黑色、深灰色、浅灰色和白色），那么也许值得为所出现的颜色构造一张表（这里，表中仅有4项），然后将像素颜色转换成单个通道，即这个查色表的索引。在我们报纸的例子中，这个通道将只需要2位来表示一个像素，而不是每个像素每种颜色分量8位，结果图像被压缩了12倍。对包含更多颜色的图像来说，节约的没那么大。在一个极端的例子中，每个像素的颜色都不相同，那么，查色表就与以RGB三元组存储的原图像一样大，而图像中包含的查色表索引值占据了额外的更多空间。大致上说，如果颜色个数小于像素个数的一半，采用查色表方式是值得的。（当然，如果硬件也通过查色表方式显示图像，那么用这种方式存储图像就比用RGB三元组方式更容易、更快。通常，这种硬件为查色表提供了大小适中的空间，大约每像素8到12位。）

采用这种单通道方法，每个像素仍然至少需要一个单位信息。如果图像有很多重复的部分，可以对一个通道做*行程编码*进一步压缩图像。行程编码包含一个计数和一个值，计数指示这个值重复的次数。Utah Raster Toolkit的设计包含了这个基本思想的很多改善方法[PETE86]。例如，计数n是8位有符号整数（取值在-128到127之间）：一个负的计数表示接下来的n个像素没有被编码，一个非负的计数表示后面一个单位的数值就是接下来的$n+1$个像素的值。进一步的改善可能包括将某些负值赋予特殊的含义：-128也许表示接下来的几个字节的信息给出了一个扫描行和需要跳过的位置（为了跳过大片的背景颜色区域），-127也许用来指示跳到某个特定的扫描行。这个简单的行程编码策略在最坏的情况下也只会每126个值增加一个额外的字节（-126表示接下来的126个像素没有被编码），其代价是0.25%。对一幅红、绿、蓝在最好的情况下，图像中所有的像素颜色相同，压缩比将是100倍：128个像素的存储量被压缩成颜色分量各占8位的图像来说，这一个像素的存储量在这个例子中是24位，而计数字节还需要加8位。

还有其他用于压缩通道的好格式。例如，我们可能将位图中的像素值存放在一个整数中（0或者1），但是，这个整数中的大多数位都浪费了。我们改变一下，将每个像素的值存放在1个二进制位中（这是术语*位图*的本意）。如果图像中包含了这样一些区域，它们是用宽度为8的倍数的模式填充的，那么，我们可以进行类似的行程编码，其中的第一个字节给出了计数n，接下来的字节给出了位图中重复$8n$次的像素的值。对彩色图像来说，这个方法不大可能比普通的行程编码效果更好，因为8个值组成的块必须有重复，才能产生压缩效果。 845

行程编码以及其他一些标准的信息理论方法（如霍夫曼编码）将图像看成是由通道组成的，每个通道又可以被想像成关于某个值的线性表（虽然多个通道可以被看成一个大的通道，从而我们也能对RGB三元组的集合做行程编码）。其他方法将图像看成关于某个值的二维数组，从而能够利用行间的连贯性。这些方法中的一类就是基于四叉树的使用。

基于四叉树的图像描述方法的基本思想是，一幅图像的某个区域可能相当的均匀，因此，可以认为该区域中的所有像素具有同样的值。确定这些近似恒等的区域是算法的关键。这个算法可用于图像的单个分量，如红色分量组成的二维数组，也可用于像素的总体值。为了简单起

见，我们以单个分量来描述算法。这个算法需要一个机制，用来确定图像中一个区域的平均值，以及该区域的偏移范围（像素值偏移平均值的范围）。

开始时，图像被作为一个整体来考虑。如果图像的偏移值足够小（小于或等于某个非负的误差范围），那么，就报告整个图像仅有一个值——图像的平均值。（如果误差范围被设成了0，那么图像的值必须真的是恒等的，这种情况才会发生。）如果从平均值的偏移并不小于误差范围，那么，记录下图像的平均值，将图像分割成四块区域，同样的算法再对其中的每一块做一遍。如果必要，分割过程一直进行下去，直到每块区域仅包含一个像素，此时算法结束。对单个像素组成的区域，从平均值的偏移一定是0，因此小于或等于任何误差范围。

我们可以通过以下方法改善上面的算法：当图像被分割时，不是记录图像的平均值，而是记录四块子区域的平均值，当图像没有被分割时，记录它的平均值。这样做的好处在于，当图像被重新显示的时候，如果按广度优先方式遍历四叉树，显示过程将是由粗到精的，图像的细节逐步增加。开始时的图像是四个彩色矩形，接下来，每个矩形被分割，它的颜色变得更丰富，如此下去直到结束。对一个目标是扫描大量图像的系统来说，这种方法尤其方便：仅仅在几个字节的信息被传送之后，图像的大致样子就开始出现了，用户可以选择拒绝这幅图像转而看下一幅图像。当图像是在一个低带宽的传输通道上传输时，这种对图像大致内容的判断特别有用。四叉树图像压缩方法由Knowlton、Sloan和Tanimoto开发[KNOW80；SLOA79]，Hill[HILL83]进一步改善了这个算法。习题17.7讨论了其他建立图像四叉树的方法，某些方法可能比这里描述的效率更高。

17.7.2 用于图像压缩的迭代函数系统

第二种图像压缩算法是基于迭代函数系统(IFS)的。采用这种压缩方法，压缩比可以非常高，
846 但压缩图像所付出的代价也同样非常大。对每一幅图像，该算法需要用户交互地求解一个几何问题，我们将在下文中描述它[BARN88a]。并且，像非破坏性的压缩方法一样，鸽笼原理[⊖]指出，如果一些图像被压缩了，另一些图像必然以适当的程度膨胀（因为从$n \times k$的数组到$p \times q$的数组不存在1对1映射，这里pq是小于nk的）。IFS技术的优点在于，包含大量几何规则图形的图像会被压缩，而包含类似噪声的图像有可能膨胀。

一个IFS代码是一组仿射映射$\{w_1, \cdots, w_r\}$的集合，每一个变换w_i都将平面映射到它自身，并且还有一个伴随概率p_i。这些映射必须是压缩的，也就是说，在总体上，经过这些映射的作用之后，两点之间距离应该减小（更精确的要求在[BARN88a]中描述）。一个平面仿射映射的公式定义如下

$$w\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{17-1}$$

它由6个数完全决定：a, b, c, d, e和f。注意，这些仿射映射仅仅是平面旋转、平移和缩放变换的组合。压缩条件表明它的缩放因子必须是小于1的。

下面的几页简略地介绍了如何用IFS代码生成一幅灰度图像，这个方法很容易扩展为从3个IFS代码生成3幅灰度图像，其中的每一个作为彩色图像RGB分量的一个。（[BARN88a]采用了不同的颜色编码方法。）这种从IFS代码生成的图像的过程本质上是我们所要介绍的IFS算法的解码过程，我们在这里先介绍它。

考虑一个矩形V，它在平面上定义了一幅图像，想像一下将这个矩形划分成矩形网格，其中的每一个小矩形记为V_{ij}，$i = 1, \cdots, n$；$j = 1, \cdots, k$。选取一个点(x_0, y_0)，它在某一个映射（不失

⊖ 鸽笼原理是这样一个观察结果，即如果有多于m个物体放在m个盒子里，那么必然有些盒子中有多于1个物体。

一般性，假定为w_1）下保持不变。现在，我们以随机的顺序（由概率p_i确定）将映射$w_1,\cdots, w_r$作用于(x_0, y_0)，看看它将被映射到哪里，我们把它落在V_{ij}中的次数作为像素$[i, j]$的最终亮度。这个过程的伪代码在图17-23中显示。

在进行这个算法的最后一步之前，每个image$[i,j]$项表示初始点经过随机的映射之后，落在图像中的第$[i,j]$块的次数。为了使这个数准确地反映在无限次的迭代变换过程点落在小方块中的概率，算法的迭代次数必须非常大：K应该比图像中的像素个数大若干倍。

这个算法本质上是要创建IFS的吸引子（attractor）。吸引子是一个集合，记为A，它具有这样的性质，即如果所有的仿射映射都作用于A，再将结果取并集，结果仍然是A：

$$A = \bigcup_{i=1}^{r} w_i(A)$$

847

集合A由初始点(x_0, y_0)在迭代变换的过程中经历的点组成。这些点落在某些位置的频度要高于其他位置，它们落在一个区域内的可能性定义了该区域的一个概率测度。如果在无限次的迭代变换过程中，一个点落在区域Q内的时间占全部时间的p部分，那么这个小区域Q的测度即为p。我们正是利用这个概率测度来计算像素值的。

```
void IFS (double image[MAX][MAX])
/* 给定一组仿射变换w_i，和伴随概率P_i（它们是全局变量），  */
/* 生成一幅灰度图像 */
{
    int x, y;                          /* 平面上的一个位置 */
    int i, j;                          /* 循环计数器 */
    int m;

    Initialize x and y to be a fixed point of w0;
    Initialize image[i][j] to 0 for all i and j;
    for (i = 0; i < K; i++) {
        double r = Random (0, 1);      /* 随机数0<=r<=1 */
        double total = p[0];           /* 概率计数器 */
        int k = 0;
        while (total < r) {
            k++;
            total += p[k];
        }
        apply (k, x, y);               /* 将w_k作用于点（x,y） */
        for (each i, j pair)
            if (LiesIn (x, y, i, j))   /* 如果(x,y)在V_ij中，返回真 */
                image[i][j]++;
    }

    m = maximum of all image[i][j] entries;
    for (each (i, j) pair)
        image[i][j] /= m;
}  /* IFS */
```

图17-23 迭代函数系统绘制算法

因为K必须非常大，所以从一个IFS代码重建一幅图像所需的时间就很长（虽然这个过程具有非常好的并行性）。如果从一幅图像创建一个IFS代码怎么样呢？为了做到这一点，我们必须

848 找到一组平面仿射映射，它们具有这样的性质，即将它们作用于原始图像后，所得结果的并集与原始图像很“相似”。图17-24显示了如何通过先创建四个小的树叶（有一点重叠），再将它们拼贴起来得到目标树叶。



图17-24 通过拼贴生成树叶。（©Michael Barnsley, *Fractals Everywhere*, Academic Press。）

拼贴定理[BARN88a]保证：由这些仿射映射组成的任何IFS都有一个吸引子，它与原始图像很相近，调整w_j的伴随概率会改变w_j控制的那部分图像的辉度。为了将一幅图像压缩成一个IFS代码，用户必须找到一种方法，将原始图像表示成若干个子图像的并集，每个子图像又是原始图像的仿射映射结果。这就是前面提到的要由用户解决的几何问题。Barnsley宣称这个过程可以自动完成[BARN88b]，只有这种机制被公诸与众，这一技术才可能用于压缩大量的图像。同时，这个方法也可用于一些有趣物体的建模（参见第20章）。彩图IV-1显示了一个采用IFS建模的整个森林。

17.7.3 图像属性

当我们按照传统的方法将一幅图像存储为几个通道时，我们必须存储每个像素的信息——每个通道中每个像素的值。与图像相关的还有其他一些整体信息，如宽度和高度，任何图像描述格式必须包含这些信息。仅仅在图像数据的开始为宽度和高度分配几个字节是不够的，经验显示，其他图像属性也会出现。典型的例子是查色表所需的空间、图像的深度（它所占的位面的个数）、通道的个数和该图像创建者的名字。为了准确地在其他设备上显示图像的颜色，我们可能还需要记录用于该图像的纯红色、纯绿色和纯蓝色的光谱，以及所用的γ校正的一些信息（如果有）。

存储这些属性的需求导致了各种灵活的图像格式出现，如RIFF[SELF79]和BRIM（从RIFF修改得到）[MEIE83]，它们是普通的属性值数据库系统。例如，在BRIM中，图像总是包含宽度、高度和创建者名称，同时还有一个“历史”域，描述了图像的创建和修改情况。采用BRIM的程序可以向历史域中添加他们自己的签名和时间戳，使得这项信息自动得到更新。图
849 17-25显示了一个文本列表，它是一幅图像典型的BRIM头。

```
TYPE (string, 5): BRIM
FORMAT (string, 11): FORMAT_SEQ
TITLE (string, 31): Molecular Modeling Intro Frame
NAME (string, 27): Charles Winston  447 C.I.T.
DATE (string, 25): Sun Oct  9 12:42:16 1988
HISTORY (string, 82): Sun Oct  9 12:42:16 1988 crw RAY/n/
00033: Sun Oct  9 13:13:18 1988 crw brim_convert
DESC (string, 21): A ray-traced picture
IMAGE_WIDTH (int, 1): 640
IMAGE_HEIGHT (int, 1): 512
BRIM_VERSION (int, 1): 1
CHANNEL_DESC (string, 21): RED GREEN BLUE ALPHA
CHANNEL_WIDTH (short, 4): 8 8 8 8
ENCODING (string, 4): RLE
```

图17-25 一幅图像的BRIM头文件

人们开发了许多图像处理软件包，其中用得最广泛的是Utah Raster Toolkit[PETE86]，它是用移植性相当好的C语言编写的，使得它可以在具有不同体系结构的系统上运行。在设计这样

的软件包时，最令人头痛的是每个字包含的字节数和一个字内字节的排列顺序。

17.8 图像的特殊效果

17.3节中介绍的图像处理技术可以产生有趣的特殊效果。如果一个图像经过高通滤波，图像中就会仅剩下很少的细节，变化缓慢的部分被去掉了。通过微分滤波器，我们可以加亮图像中所有发生突变的点。将所有亮度值低于某个水平的像素值降为0，而将其余的像素值升为1的滤波器可被用来生成高对比度的图像，如此等等。

有很多视频技术可以用来模糊图像，使图像淡化，使它们实时地滑出屏幕等等。许多这样的效果是显示时用电子硬件设备实现的，它们可以产生和修改视频信号。所有这些技术都属于电子工程领域，而不是计算机图形学领域，虽然两个领域的融合可能是富有成效的。

最后，我们介绍一种模拟氖管制作的数字技术作为本章的总结。如果我们用一个颜色均匀的等宽的（经过反走样）画刷将一个氖管的形状画在黑色的背景上，得到的结果图像不会特别地激动人心。但是，假设对图像进行平滑滤波——将每个像素的值设置为它相邻像素的平均值，连续做几次之后，我们所画的条状图形的边界就变得模糊了。如果我们现在再对图像进行亮度-映射滤波，即加亮亮度值大于某个阈值的像素，而使亮度值小于这个阈值的像素变暗，结果看起来就非常像氖管了。（还有一种方法能产生同样的效果，请参见19.3.4节中关于反走样画刷的讨论。）采用大于1的ω和小于1的α值合成图像可以使“氖”照亮它所放置的任何地方，同时保持氖管是部分透明的。 850

17.9 小结

我们已经讨论了一些存储图像的方法，包括针对编程实现方面的考虑（多个通道、头文件）和以压缩或易于传输为目标的技术（四叉树、IFS编码）。事实上，还有很多其他的图像存储格式，其中一些是商业格式，它们相互竞争，希望成为“标准”。在应该存储多少颜色信息（应该仅仅是RGB本身，还是应该包含几个光谱样本？）和图像中应该包含哪些信息（应该存储z缓存值和α值吗？）等方面，仍然存在不同意见，因此在短时间，不太可能出现统一的图像格式。

我们也讨论了图像的几何变换，包括带滤波的多重算法，以及在变换时进行滤波的必要性。图像变换所能产生的有趣效果非常多。因为使用了滤波，反复地变换会模糊图像。对位图变换涉及其他一些困难，因为没有灰度级可用来平滑变换时引起的走样。但是，因为它的数据简单，使得开发快速算法成为可能。

我们还讨论了图像的合成，它已经成为产生复杂图像的一个极流行的工具。当采用α通道时，除非构件图像有很高的几何相关性，否则在构件图像重叠的地方合成图像不会出现缝隙。如果绘制速度提高到了这样的水平，即重新产生图像不太费时，图像合成可能就不会像现在这样是一个非常重要的工具了。但是，如果计算机图形学像目前这样发展，每一代新的硬件支持更复杂的绘制技术，那么我们将期望图像的质量更好，而且每幅图像的生成时间将大致保持不变。因此，图像合成技术仍会存在一段时间。

习题

17.1 证明下面的矩阵乘积

$$\begin{bmatrix}1 & \tan(t)\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ -\sin(t)\cos(t) & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & \cos(t)\end{bmatrix}\begin{bmatrix}1/\cos(t) & 0\\ 0 & 1\end{bmatrix}$$

等于

$$\begin{bmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{bmatrix}$$

利用这个结果，说明如何用错切变换和缩放变换构造一个旋转变换。参考Weiman算法（见17.4.1节）并利用这个方法给出一个像素图的旋转算法。

851 17.2 如何构造一个Weiman式的平移算法？假定一个像素图中，黑色像素与白色像素一列一列相间排列，那么，将这个图像平移1/2像素结果如何？如果用Weiman的缩放算法作用于这幅图像，使其放大2倍，结果如何？你认为这些结果怎么样？

17.3 当缩放位图时，你不可能像在Weiman算法中那样做平均，那么，什么是选择目标像素值的好的标准？多数原则是最好的吗？如果你希望保持图像的特征，使原始图像白色页面中间的黑线仍然在结果图像中出现，该怎么办？Rothstein代码中的循环排列仍然是必要的吗？

17.4 说明在Feibush-Levoy-Cook滤波方法的第三步，边滤波过程中（见17.4.2节），定义于三角形上的体的正负号是正确的。（提示：每个三角形必须有一个正负号，并且这个正负号是关于三角形形状的连续函数——看起来相近的两个三角形正负号相同。只有当逐渐改变三角形，使它通过奇异情况——三个顶点共线时，它的正负号才会改变。因此，为了完成本题，你只需要说明两个三角形的正负号是正确的就可以了，正负号代表了它们的朝向。）

17.5 这个问题弥补了Feibush-Levoy-Cook图像变换算法（见17.4.2节）中边滤波方法的细节，在矩形区域内给定一个三角形（它的一个顶点在矩形的中央C）和一个定义在矩形区域上的函数（作为高度画出来），三角形上方的体积可以通过以下方法计算出来：

1) 假设三角形为ABC，从C向AB画一条垂直线，交AB于D点。将原三角形表示为三角形ACD和BCD之和或之差。

2) 找出三角形ACD和BCD上方的体积，再用这两个值计算三角形ABC上方的体积。

3) 通过观察知道，如果滤波函数是中心对称的，那么，第2步中需计算的三角形ACD上方的体积仅仅是关于三角形的底边AD和高CD的长度的函数（三角形BCD也类似）。

a. 画出步骤1中所描述的图形，分为两种情况：角ACB小于90°和大于90°。

b. 寻找这样一个条件，该条件决定了ABD是否等于ACD与BCD之和。

c. 给出一个计算任意三角形上方体积（如第3步中所描述的那样）的方法。因为给定的函数在一般意义下可能不可积，请考虑用蒙特卡洛方法。

d. 描述你将如何构造一个关于宽度和高度的查找表，在其中存放第3步计算出来的体积。

17.6 考虑下面描述的一幅1×3的图像，这个图像的像素分别落在(-2, 0), (0, 0)和(2, 0)点，这些点的值分别为-1，0和1。这个图像的Nyquist频率是1，它可被看成是频率等于1（或更小）的正弦函数和余弦函数线性组合的一个样本。所有的余弦项都是0（因为该函数是一个奇函数，即对所有的x，$f(-x) = -f(x)$）。

a. 计算$\sin(kx)$（对$0 \leqslant k \leqslant 1$）的系数$a_k$，使得$\sum a_k \sin(kx)$的一个样本是这幅图像。提示：只有一个$k$使得$a_k \neq 0$。

852 b. 这个图像也许是一个很简单的灰度斜坡（如果我们把 - 1想像成黑色，1为白色）。如果我们对(a)中计算出的信号在(-1, 0)和(1, 0)点采样，它们的值会像期望的那样插值在灰度斜坡上吗？

这一题说明了在不用滤波进行精确重建时，我们所遇到的困难。

17.7 假定你有一幅$2^k \times 2^k$的灰度图像，亮度范围是$0 \cdots 2^n - 1$。你可以通过将2×2的区域压缩为1个像素来得到一幅$2^{k-1} \times 2^{k-1}$的图像。在17.7.1节中，我们提出用计算区域的平均值来压缩。当然，还存在其他一些方法。通过分析压缩时所用的选择方法，我们知道，选择意味着在选择区域的某一点作为代表值。

a. 假定你希望传输一幅$2^k \times 2^k$的灰度图像，它已经用选择方法完全压缩了，那么，你需要发送多少个像素值？（假定接收方的硬件知道如何解码输入的数据，能够在显示器上显示填充的矩形。）

b. 对哪类图像，这种选择方法在图像被完全显示之前存在严重的人工痕迹？对这些图像，存在其他更好的压缩准则吗？

c. 什么样的压缩准则对发送黑白文档（例如书上的典型页面，其中可能包含插图）最适合？解释你的结论。

17.8 （注意：为了完成这个编程题，你必须用支持位图的快速bitBlt操作的硬件设备。）编写一个程序，采用深度为1位的α缓存合成图像，就像用α缓存（它决定了哪里显示，哪里不显示）在位图上画画一样。这个问题在[SALE85]有深入的讨论。

17.9 假定现在有一幅不完整的行程编码图像，其中的几个字节丢失了。你能够重建出大部分图像吗？为什么能或为什么不能？试提出一个更强的行程编码方法，使其能更容易地从一幅遭破坏的图像中恢复部分内容。

17.10 我们在17.5节中看到，很多变换都可以用多重变换实现。

a. 如果一个纹理被线性地映射到一个多边形上，然后被投影到观察平面上，证明：从纹理坐标(x, y)到图像坐标(u, v)的复合映射具有如下形式：

$$(x, y) \rightarrow (u, v),$$

其中

$$u = (Ax + By + C)/(Dx + Ey + F)$$
$$v = (Px + Qy + R)/(Sx + Ty + U)$$

b. 说明如何将这个映射分解成两重变换，就像对下面的映射所做的那样

$$u = x/(y + 1),\ v = y/(y + 1)$$

Catmull和Smith在[CATM80]中提出了用两重变换做纹理映射的思想。之所以这样选择坐标变量的名称是为了与17.6节的风格保持一致，它与其他地方描述纹理映射时采用的坐标名称不一定一样。

第18章　高级光栅图形体系结构

本章将更详细讨论第4章介绍的光栅图形系统体系结构的有关问题。我们研究光栅系统执行的主要计算和可用于它们的加速技术。尽管图形体系结构和算法的范围广泛，我们在此集中在显示3D多边形模型的体系结构上，因为这种体系结构是高端系统当前的焦点，也是大部分支持复杂图元和绘制方法系统的基础。

图形系统体系结构是计算机体系结构的特殊分支。因此，它也受到过去几十年推动通用计算机体系结构发展的半导体技术的发展的推动。可以应用许多相同的加速技术，包括流水线、并行机制和以空间换时间的方法等。然而图形应用有它特殊的要求，同时也提供了新的机遇。例如，因为图像显示通常包括大量的重复计算，它就比通用计算更容易利用大规模并行方法。在高性能图形系统中，计算量通常超过了一个单个CPU系统的处理能力，因此并行系统在近几年成为标准。这些并行系统的组织是图形体系结构的主要焦点，也是本章讨论的重点。

我们首先来回顾一下第4章介绍的简单光栅显示体系结构，然后描述一系列的技术来提高系统的性能，并讨论在每一个性能水平上出现的瓶颈问题和能够用来克服它们的技术。我们将看到，三种主要的性能瓶颈一直阻碍我们试图提高绘制速度的努力：执行几何运算的浮点运算数，计算像素值的整型运算数，为存储图像和确定可见面的帧缓存的内存存取操作数。这些需求对图形系统有着普遍和深入的影响，也使得今天所看到的多处理机图形体系结构多种多样。在本章的最后，我们还将简单地讨论几种不常用的体系结构，例如，用于光线跟踪、真3D显示和商用飞行模拟器的体系结构。

18.1　简单光栅显示系统

第4章已经指出，一个简单的光栅显示系统包括一个CPU、系统总线、主存、帧缓存、视频控制器和CRT显示器（见图4-18）。在这样的系统中，CPU要执行所有的建模、变换和显示计算，并将最终的图像写到帧缓存。视频控制器按光栅扫描的顺序从帧缓存读取像素数据，将数字像素数据转换成模拟信号，驱动显示器进行显示。

重要的是要记住，如果这样的系统有足够的帧缓冲内存，有一个合适的CRT显示器，并且有足够的时间，它能够生成并显示实际上无限复杂和真实的场景。这里并不讨论用于提高这一基本能力的体系结构和体系结构技术（除了18.11节中论述的几种特殊的3D显示设备）。相反，图形体系结构中的大部分工作都涉及到提高绘制速度的努力。

在第4章，我们讨论了限制这个简单系统性能的两个问题：视频扫描输出所需要的大量帧缓冲内存访问周期和图像生成给主CPU带来的负担。现在，我们将更详细地讨论每一个问题。

18.1.1　帧缓冲内存访问问题

在第4章，我们计算了一个低分辨率单色显示器被刷新时连续内存访问的间隔时间。对一个有16位字长的系统，访问速率是重要的，每864ns访问一次内存。具有高分辨率彩色显示器的系统需要更高的内存访问速度。例如，以60 Hz刷新一个有32位（一个字长）颜色1280 × 1024的屏幕需要每1/(1280 × 1024 × 60)s = 12.7ns访问一次内存。甚至这只是一个平均的内存访问速率，不是峰值速率，因为在水平或垂直回扫期间内是不能扫描输出像素的[WHIT84]。另

一方面，一个简单的动态随机访问存储器(DRAM)系统的一个周期大约是200ns，比要求的速度慢16倍。显然，必须想办法来提高帧缓存内存的带宽。

下面的几节将讨论已经被许多系统设计者所采用的解决方法。其中有些方法仅仅能有限地提高系统的性能，但对于低分辨率系统却是足够的。另外一些方法能极大地提高内存的带宽，
856 但使得系统非常复杂。我们首先简要回顾一下DRAM存储器的基本知识，因为DRAM的特点对可用的解决方法影响很大。

18.1.2 动态存储器

大部分计算机系统都使用DRAM作为内存。能够一直保持存储数据的静态随机访问存储器(SRAM)可以制作得比每几毫秒就必须刷新来保持数据的DRAM快。但是DRAM每个存储位具有高得多的密度和便宜得多的价格。(DRAM的读写时序逻辑也比较复杂，但是这些问题可以很容易地用支持电路来解决。)

图18-1是一个典型的1M位DRAM芯片的结构。像在大多数DRAM中一样，单个存储位的存储单元被排列成一个或几个方形矩阵(在本例中，有四个矩阵，每个矩阵维数为512 x 512)。垂直的存储位数据线将数据传入和传出存储矩阵，一条存储位数据线对应每一个矩阵中的一列。在读写操作期间，每一列中的一个内存单元同与其对应的一条位数据线相连接。在读操作期间，同每一条位数据线相连接的读出放大器放大并恢复位数据线上的微弱信号。

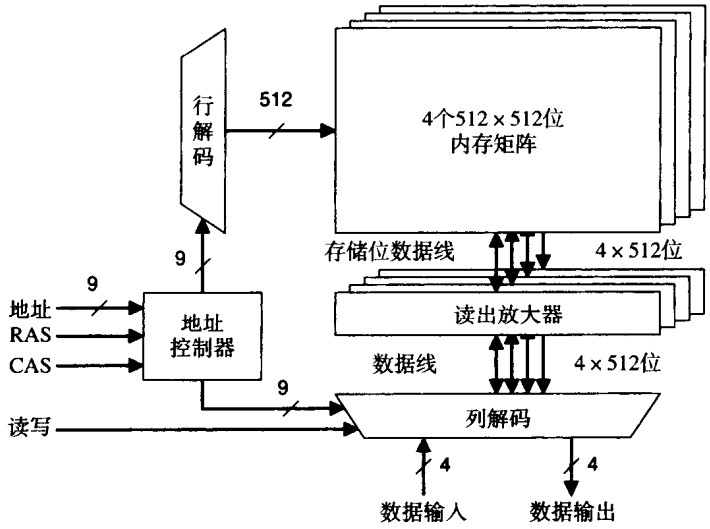


图18-1 一个1M位(256K x 4)DRAM芯片。一行数据是从每一个存储矩阵中同时读写的。列解码器控制选择行中的哪一个单元(列)被访问

在一个DRAM芯片中，读写操作都需要分两步进行。第一步是选择一行。当相应的行地址
857 已经在地址输入上时，将行地址开关(RAS)置位。行解码器输出一个512位的向量，这个向量只有在选中行位置上是1，其他位置都是0。这个位向量决定了哪一行的存储单元同存储位数据线和读出放大器相连接。

第二步是选择一列，当相应的列地址已经在地址输入上时，将列地址开关(CAS)和读写信号置位。列地址从每一个存储矩阵的活动行中选择出一位，选出的位或者缓存起来用于输出(在读操作中)，或者设为输入的数据值(在写操作中)。有些DRAM还提供了称为页模式的快速访问模式，以支持对同一行数据的连续读写操作。在页模式中，行仅需要选择一次，连续的列则由列地址位和CAS信号选择。页模式中连续的内存访问仅仅需要一个地址周期，而不是两个。因此，如

果相邻的像素存储在内存的同一行，页模式几乎可以使帧缓存的刷新和显示带宽增加一倍。

要得到字长多于单个内存芯片数据针脚的内存，可以将多个内存芯片并联起来。在这种结构中，整个字的数据可以在一个内存周期内从内存系统中读出或写入。例如，8个4位宽的DRAM能够组成一个32位的内存系统。

18.1.3 提高帧缓冲内存带宽

像我们看到的一样，要得到足够的帧缓冲内存带宽，即能满足视频扫描输出又能满足CPU刷新，是一件十分麻烦的事。目前已经提出了许多方法，并应用到了许多系统中。

一个适合于低分辨率显示系统的部分解决方案是将帧缓存放在独立的总线上。这使得视频扫描输出能和不影响帧缓存的CPU运算同时进行。仅当CPU需要读写帧缓存时，它才需要同视频扫描输出竞争总线周期。

第二个部分解决方案是利用DRAM页模式的优点。因为视频扫描输出是按顺序访问内存的，DRAM的页缺失很少发生，所以帧缓存的运行几乎是正常速度的两倍。但是注意，CPU访问帧缓存却很少有规律，也就不能利用页模式的优点。我们可以提供一个数据高速缓存来减少CPU引起的页缺失。就像传统的计算中，高速缓存可以减少CPU和内存之间的数据交换，因为一个内存周期中访问的内存地址，很可能在不久的将来重新被访问。而且，高速缓存单元通常包括几个字，在高速缓存和主存之间交换数据可以利用页模式。尽管高分辨率的显示系统还需要18.1.5节讨论的进一步解决方法，将页模式和数据高速缓存结合起来还是可以充分地减少内存访问的问题。

第三个独立的方法是使用两个帧缓存内存，建立一个双缓冲系统，在显示一个缓冲区的图像的同时，计算另一个缓冲区的图像。图18-2给出了一个可能的实现，其中多路复用器将每一个缓冲区同系统总线和视频控制器连接起来。双缓冲技术可以使CPU不间断地访问一个缓冲区，同时视频控制器也能不间断地访问另一个缓冲区。但这种方式的双缓冲技术同单缓冲显示系统相比，使用了两倍的内存，价格比较昂贵。而且，为DRAM提供双路连接的多路复用器需要许多芯片，这也增加了系统的规模。（实际上，也可以使用具有两个可完全独立读写端口的双端口内存，但密度低和价格高的缺点使得它们几乎对所有的应用都是不实际的。）

858

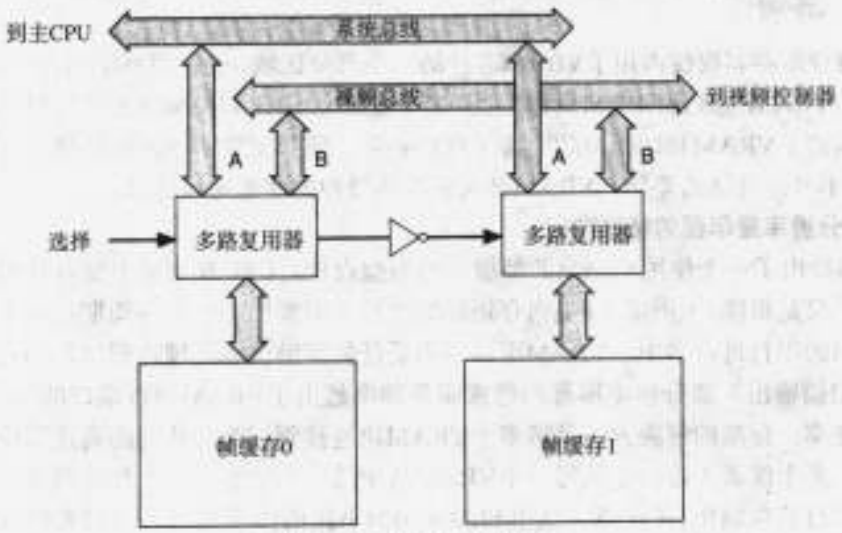


图18-2 双缓冲帧缓存

18.1.4 视频RAM

在1983年，德克萨斯仪器公司发布了一种新的DRAM，即视频随机访问存储器(VRAM)，

特别设计用于使视频扫描输出独立于其他的帧缓存操作[PINK83]。如图18-3所示，VRAM芯片同普通的DRAM芯片基本相似，但它包含一个同第二个数据端口相连接的并行输入/串行输出数据寄存器。串行寄存器的宽度同内存矩阵的宽度一样，置位数据交换信号可以在一行内存数据读出的同时加载寄存器。串行寄存器有它自己的数据时钟，使它能够高速传送数据。串行寄存器和端口能够有效地为内存矩阵提供第二个串行访问端口。如果这个端口被用于视频扫描输出，扫描输出就能同正常的芯片读写操作并行进行。从而事实上消除了视频扫描输出问题。

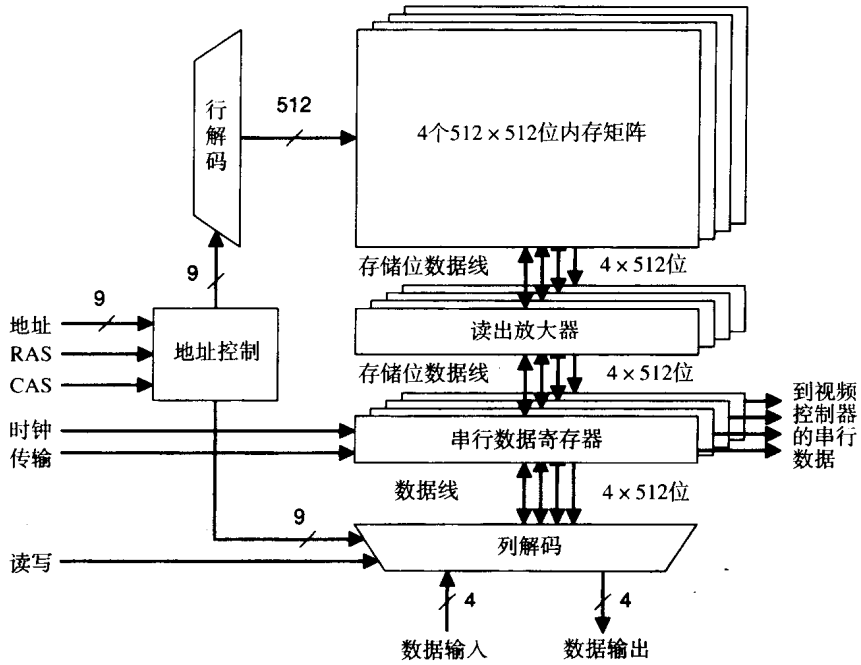


图18-3 1M位(256K×4)VRAM芯片结构图：串行数据寄存器为内存矩阵提供了第二个（串行）访问端口

因为移位寄存器仅仅占用了VRAM芯片的一小部分区域，也仅需要增加几个管脚来控制它，所以理论上VRAM应该比DRAM只稍微贵一点。但是，因为VRAM的生产量很小，规模效益的因素大大提高了VRAM和DRAM的比价（在1989年，同等密度VRAM的价格大约是DRAM的两倍）。如果不考虑价格的差异，VRAM是大多数帧缓冲内存的绝好选择。

18.1.5 高分辨率显示器的帧缓存

图18-4给出了一个使用VRAM的帧缓存的典型设计。CPU在本质上没有任何帧缓冲内存的访问限制。仅是每隔n个周期（n是内存矩阵的维数）需要用一个内存周期把一个新的数据行加载到VRAM的串行寄存器中。VRAM可以不需要任何辅助就能处理达到512×512像素分辨率屏幕的视频扫描输出。高分辨率屏幕的像素刷新频率超出了VRAM串行端口的30 MHz或更少的数据传输速率。标准的解决方法是将多个VRAM块连接到一个芯片外的高速移位寄存器上，如图中所示。多个像素（每一个从每一个VRAM块中读出）能够并行地加载到寄存器中，然后以视频速率串行移位输出。（注意，这里1280×1024系统的像素是512×512系统的5倍，使用了5路多路复用。）这个移位寄存器通常同查找表和数模转换器集成到一个芯片中，例如Brooktree的Bt458 RAMDAC [BROO88]。在如图18-4所示的每像素24位的系统中，通常使用3个RAMDAC芯片，每8位颜色通道一个。

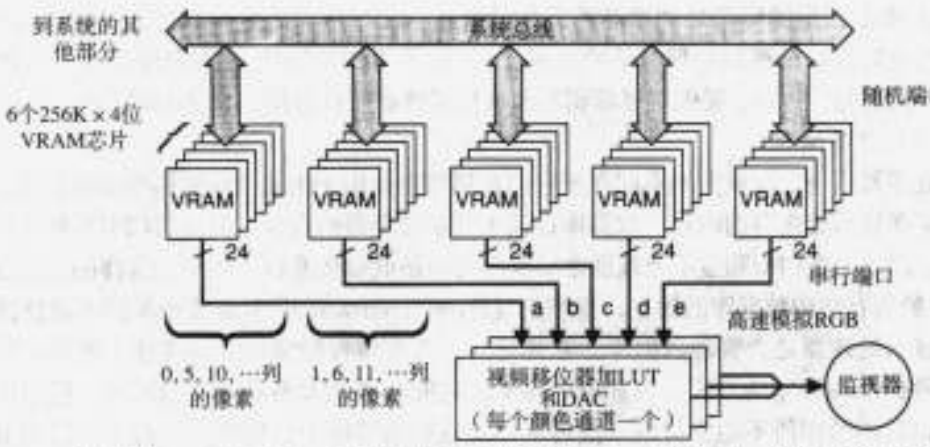


图18-4 一个使用VRAM的1280×1024分辨率显示系统结构图：系统使用一个并行输入/串行输出的移位寄存器来实现高速率的像素输出，以驱动显示器

VRAM给帧缓存内存访问提供了一流的解决方案。这样的系统中仅存在CPU处理速度瓶颈。为生成一个（二维或三维的）光栅图像，处理器必须计算每一个图元的每一个像素。因为一个典型图元往往覆盖许多像素，一幅图像中又往往包含几千个图元，所以生成一幅高分辨率的图像需要几百万个CPU操作。因此，即便是最快的CPU也不能胜任实时交互绘制图像的任务。

18.2 显示处理器系统

要构造一个比上节所述的单CPU系统性能更高的系统，我们必须解除一些CPU的图形计算负担，或者提高CPU的性能来完成这些工作。这里我们将考虑那些不过分增加系统规模和成本的策略（我们将在18.6节到18.10节考虑大规模、多处理器解决方案）。这些策略一般分为三类：(1)同主CPU共享系统总线的协处理器，(2)有独立的总线和内存系统的显示处理器，(3)包括对图形操作的内部硬件支持的集成处理器。

协处理器系统已经在第4章广泛讨论（单地址空间(SAS)结构）。这些系统已经应用于2D系统中，但并没有在3D系统中得到广泛应用。主要原因是3D系统比2D系统需要更多的操作，并且SAS体系结构中的总线竞争严重限制了系统的性能。后两种方法已经成功地应用于3D图形系统，我们现在就深入讨论它们。

18.2.1 外部显示处理器

图4-22给出了一个典型的外部显示处理器系统组织。显示处理器通常有自己的内存系统（包括帧缓冲）和与主CPU的高速通信连接。主CPU向显示处理器发送显示命令，显示处理器利用驻留软件、固件例程或特殊硬件来执行它们。

859 ↓ 861

显示处理器可以是与主CPU同类型的处理器，也可以是专用处理器。使用同类型的处理器（例如Masscomp MC-500[MASS85]）可以使得软、硬件的设计更简单、灵活。可是这种系统并没有利用大多数图形操作的规律和特点，由于系统的整体处理能力提高了一倍，因此这种系统的性能最多是原来的两倍。目前大多数显示处理系统都使用为图形操作特殊设计的外部处理器。

1. 单芯片图形处理器

单芯片图形处理器是一种非常成功且便宜的外部显示处理器类型，它是作为单芯片视频控制器功能扩展发展起来的。因此，许多现有的图形处理器芯片都结合了视频控制器和显示处理器的功能（18.2.2节给出了一种非常流行的图形处理器芯片TMS34020）。

特别针对3D图形的图形处理器芯片只是最近才见到。这类芯片中的先锋是Intel的i860，它既可以作为支持图形处理的处理器单独使用，也可以作为3D图形处理器使用。与大部分2D图形处理芯片不同的是，i860不提供对屏幕刷新和视频时钟的硬件支持。18.2.4节将深入讨论i860。

2. 显示处理器编程

显示处理器系统之间最大的不同在于它们在哪里存储用于生成图像的模型或数据库：在CPU的内存还是在显示处理器的内存。数据库存储于显示处理器的内存，可以将CPU从数据库检索中解放出来。因此，在CPU和显示处理器之间仅需要一条低带宽通道。然而，这样的系统也继承了第7章所述的所有结构数据库的缺点，特别是规范的“hard-wired”数据库模型的不灵活性。

不管显示处理器是否管理数据库，它都是一个需要编程的额外的处理器。通常，它的编程都采用一种标准图形库的形式，例如PHIGS+，它能够满足大多数应用的需求。但是应用中总是不断地出现图形库所不能支持的一些需求。在这样的情况下，用户就不得不自己对显示处理器进行编程（这甚至是不可能的），或者利用现有的图形处理器的特性来设计一些麻烦的方法来实现。

18.2.2 德克萨斯仪器公司的TMS34020——单芯片外部显示处理器

德克萨斯仪器公司的TMS34020[TEXA89]是一个单芯片图形处理器的实例[⊖]，它是为PC和工作站的2D显示加速而设计的。它可以同TMS34082浮点协处理器一起使用，TMS34082可以快速执行3D图形计算的3D几何变换和裁剪操作。与其他仅执行特殊图形操作的图形处理器芯片不
862 同的是，34020是一个完全可编程的32位处理器，如果需要，它可以作为单独的处理器使用。

图18-5给出了一个使用34020（和可选的34082）的典型系统配置。如图所示，图形处理器的局部内存既可以使用VRAM作为帧缓存，DRAM作为程序和数据的存储，也可以全部使用VRAM。34020芯片本身包含了时钟和寄存器来控制视频扫描输出和DRAM刷新。

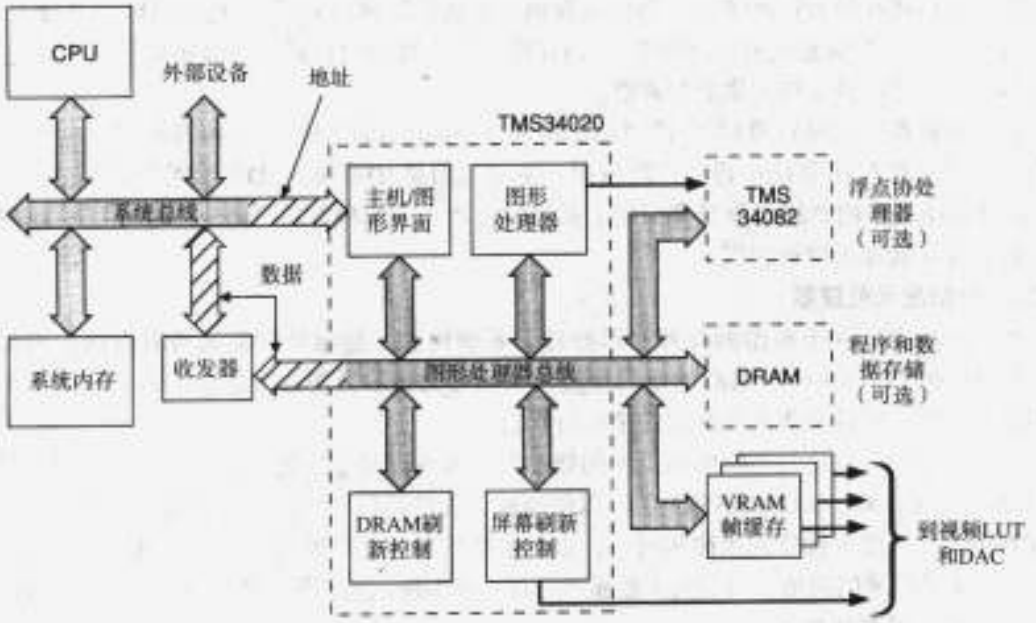


图18-5 作为外部显示处理器的TMS34020图形处理器芯片的系统结构图

⊖ 这个例子的资料由德克萨斯仪器公司的Jerry R.Van Aken提供。

34020支持像素和像素图的数据结构，既可以用(x,y)坐标来访问像素，也可以用它们在内存的地址来访问。下面的几条指令实现了基本的2D图形操作：

- PIXBLT: PIXBLT(pixel-block transfer)指令实现了一个通用的bitBlt操作。它首先从源矩阵中读取出像素行，自动地对齐、屏蔽和裁剪这些像素，并复制1到8位的像素，然后将它们写到目的矩阵中。它使用了许多在第19章中讨论的优化策略，而且并行地处理几个像素，因此它的传输速率利用了系统的全部内存带宽——每秒18 000 000个8位像素。另一个PIXBLT的特殊版本通过将压缩的1位位图扩展为8位像素映射来提供对文本的支持。
- FILL: FILL指令是用一种一致的颜色填充一个矩形区域的所有像素。大区域的填充速率可以达到内存总线的最高速度。德克萨斯仪器公司的1M位VRAM支持一种特殊的块写模式，可以将一个集成在芯片上的颜色寄存器的数据一次性写入到四个内存地址。使用这种块写模式，34020的填充速率可以达到每秒160 000 000个8位像素。 863
- FLINE: FLINE指令使用中点扫描转换算法（见3.2.2节）画一条1个像素宽的直线。FLINE指令每秒可以画5 000 000个像素。
- DRAV: DRAV(draw-and-advance)指令在(x,y)坐标对指示的地方画一个像素，并同时将x,y坐标加1。它用于画圆和椭圆的增量算法的内部循环中；例如，中点圆程序的内部循环（见3.3.2节）每秒画2 000 000个像素。

34020的软件一般分为两部分：用汇编语言写的时间限定的图形子程序和用高级语言写的其他软件。运行在34020上的软件与运行在主CPU上的应用程序进行通信。

34020包含了一个集成在芯片上的512字节的指令高速缓存和30个通用寄存器，这些内存足够用来存储许多图形子程序内部循环需要的指令和数据。一旦循环所用的指令和数据从外部内存中读入，内存总线就可以专门用于向帧缓存传送和读出像素数据。

TMS34082浮点协处理器芯片可以直接同34020相连接，它提供的浮点指令和寄存器可以提高系统的3D性能和其他需要大量浮点计算的应用性能。34082监控着34020的内存总线，并接受由34020传送到总线上的浮点指令和数据。34082能够每2.9ms转换一个3D齐次坐标点（包括齐次坐标除法）。

18.2.3 集成的图形处理器

提高系统性能的另一种方法是将图形的支持直接集成到主CPU中。这种方法只有在最近才是可行的，因为高密度VLSI技术的发展已经能够使得整个系统集成到一个芯片上。虽然完全可编程的2D图形处理器（例如，德克萨斯仪器公司的34020）更多考虑了图形功能，而相对较少考虑通用处理功能，我们还是把它看成是这种方法的早期的例子。

Intel的i860是最早集成对3D图形支持的微处理器芯片。支持这种方法的人声称，用于支持合理的3D图形性能的芯片面积很小，但它的收益却很大。因为3D图形已经成为许多计算环境中所必须的一部分。随着VLSI密度的不断提高，更精密的系统和更多的系统部件可以集成到一个芯片上，使用这种方法就可能构造出高性能、低成本的3D系统。

18.2.4 Intel的i860——一个具有集成3D图形支持的单芯片微处理器

VLSI技术的最新进展已经能够将大的晶体管集成到一个芯片上，1989年就可以超过1 000 000。这使得高性能的CPU芯片可以集成其他性能，例如高速缓存、I/O控制器和对特殊指令的支持。 864
Intel的i860，也叫作80860[GRIM89]，就是这样一个集成了对3D图形支持的芯片。图18-6给出了i860的主要数据通路结构图。

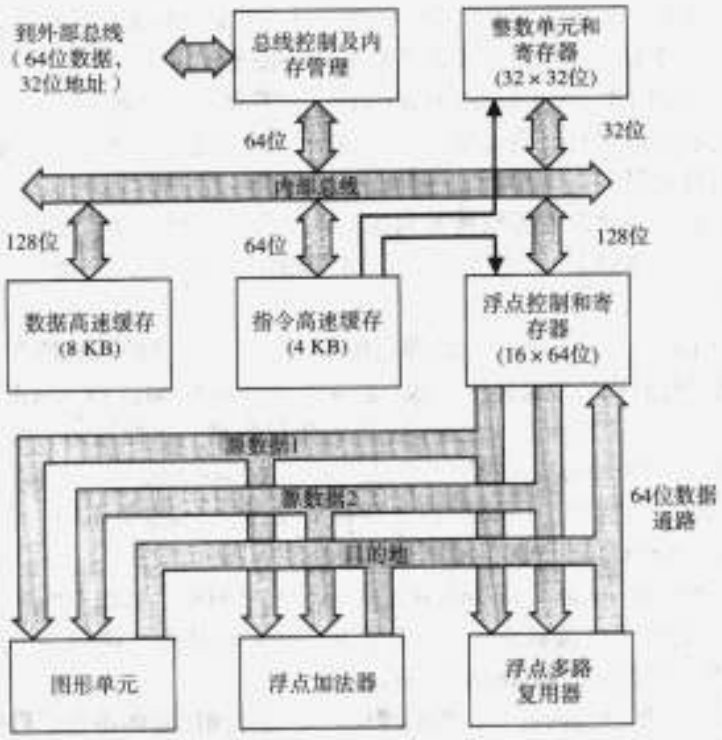


图18-6 Intel i860微处理器数据通路结构图。（摘自[GRIM89]。）

据Intel称，i860的性能达到了如下的基准：每秒33 000 000 VAX相当指令（在编译过的C程序）、13个双精度MFLOP（使用Linpack基准）和500 000个齐次向量变换[INTE89]。

i860的图形指令通过将尽可能多的像素打包在一个64位的数据字中来实现并行操作。对于使用8位像素的应用，可同时执行8个操作。对于3D立体图形，通常需要32位像素，可同时执行两个操作。并行的图形指令包括：

- 并行多线性插值。
- 并行多z缓存的比较。
- 有条件地并行多像素刷新。

对使用8位像素的系统（8个操作可以并行执行），i860每秒能对50 000个100像素大小的三角面片进行扫描转换和Gouraud光照计算。

865 i860既可以作为外部显示处理器使用（例如，用80486微处理器作为主CPU），也可以作为单独的处理器使用。i860将前端和后端的功能集成到一个处理器中，从而能够使用很少的部件——最基本的处理器、内存和视频系统——构造出功能强大的3D图形系统。

随着VLSI密度的进一步提高，我们可以预见更高性能和更复杂的图形处理器不断被设计出来。那些通常是2D图形处理器的功能，例如扫描输出和视频计时电路，也可能被集成到这样的处理器中。

18.2.5 三个性能障碍

使用目前为止所述的几种技术可以设计出不同性能需求的系统。使用通用微处理器作为应用处理器和2D图形处理器的低端系统非常适合于PC机。使用高性能应用处理器和包含显示列表内存及变换硬件的高速3D图形处理器的高端系统适用于工程和科学工作站。

但是这样的设计到底能达到什么样的性能水平呢？让我们来想像这样一个显示系统，它使

用所能得到的最快的CPU和显示处理器，帧缓存使用所能得到的最快的VRAM。这样的系统所能达到的性能给人留下深刻的印象，但仍然是有限的。实际上，这样的单数据流结构不能够达到以交互刷新速率显示大的3D数据库的要求。以更高的刷新速率显示更多图元的体系结构必须要克服下面三个障碍：

- 浮点几何处理：对图元的变换和裁剪进行加速，以超过单浮点处理器所能达到的速度。
- 整数像素处理：对扫描转换和像素处理进行加速，以超过单显示处理器和内存系统所能达到的速度。
- 帧缓冲内存带宽：提供比常规内存系统更高的帧缓存访问带宽（快速读写）。

在常规计算机设计中也有类似的性能障碍：常见的处理和内存带宽瓶颈。在常规计算机系统中，这些瓶颈的存在已经导致了并行处理的发展。同样，在过去十年中，图形体系结构的研究也主要集中在如何使用并行技术来克服这些性能障碍。

我们在讨论并行图形体系结构之前，先建立一些有关图形应用的需求和多处理技术的基础知识。然后，我们将讨论建立并行系统来克服这些性能瓶颈的各种方法。

18.3 标准图形流水线

这里，我们从硬件角度回顾一下第16章讨论的标准图形流水线。以前已经指出，绘制流水线是光栅显示系统所需计算的逻辑模型，而不必是物理模型，因为流水线的每一级既可以用软件来实现，也可以用硬件来实现。 866

图18-7给出了一个使用基本图元（线和多边形）和常规明暗处理技术（恒定、Grouraud和Phong）的绘制流水线。我们将依次讨论流水线的每一级，并特别注意那些已经成功应用于硬件系统的算法和在将来可能有用的算法。在本节的最后，我们会估计一下在处理一个10 000个多边形的数据库时图形流水线每一级的计算量。



图18-7 使用Gouraud或Phong多边形明暗处理的标准图形流水线

18.3.1 显示遍历

流水线的第一级是遍历显示模型或数据库。因为连续帧之间的图像会发生任意数据的变化，所以这一步是必须的。数据库中的所有图元，包括上下文相关信息，例如颜色和当前变换矩阵，都要进入随后的流水线。第7章已经给出了两种类型的遍历：瞬时模式和保留模式。两种方法各有优缺点。他们之间的选择依赖于应用和所用的特殊硬件体系结构的特点。

瞬时模式的优点是较灵活。显示模型不需要遵守任何特定显示列表结构，应用程序也可以创建每帧任意变化的场景。然而，主CPU必须执行瞬时模式遍历，花费一些本来可以做其他工作的时钟周期。另一方面，如果结构数据库是存储在显示处理器的局部内存，保留模式则允许显示处理器自己来完成这些工作。通过优化数据库存储和访问程序，或使用特别设计的硬件遍历器，可以加速保留模式数据结构的遍历。而且，由于主CPU仅仅为每一帧编辑一下数据库，而不是从脚本中重建它们，在主CPU和显示处理器之间用一个低带宽的通道就足够了。当然，

这仅允许在各帧之间存在相对较少的变化，或者说这限制了系统的性能。

系统设计者们一直在争论如何选择遍历模式[AKEL89]。许多人争论说保留模式有较高效率和较高的性能。另一些人则相信瞬时模式支持更广泛的应用，而且如果系统有一个性能足够好的CPU，瞬时模式也不一定导致性能的下降。

不幸的是，显示遍历对处理能力的需求与使用的遍历方法和特定显示模型的特点有关，要
867 估计它是困难的。为显示一个字长的数据，最少需要一个读操作和一个写操作。如果结构的层次比较深，或者它包含许多模型变换，对处理能力的需求可能会大得多（因为指针访问、状态保存、变换矩阵级联等都会引起额外负担）。

18.3.2 模型变换

在流水线的这一级，图元都从物体坐标空间转换到世界坐标空间。每一个多边形的顶点都要乘以一个变换矩阵来完成转换。这个变化矩阵是由每一个模型变换矩阵级联得到的。除此之外，一个或多个表面法向量也可能需要转换，这取决于采用的明暗处理方法。

常数明暗处理需要每一个多边形的世界空间坐标的表面法向量，我们可以将物体空间的表面法向量乘以模型变换矩阵的转置逆矩阵来计算得到。Gouraud和Phong明暗处理需要每个顶点而不是每一个多边形的世界空间的法向量，所以每个顶点的法向量都需要乘以变换矩阵的转置逆矩阵。

让我们来计算一下使用Gouraud明暗处理时转换一个顶点所需要的浮点计算次数。一个齐次坐标的顶点乘以4×4的矩阵需要16个乘法和12个加法。一个顶点法向量乘以逆变换矩阵需要9个乘法和6个加法（仅需要矩阵的左上角3×3部分，见习题18.1）。因此，变换一个带表面法向量的顶点需要$16 + 9 = 25$个乘法和$12 + 6 = 18$个加法。

18.3.3 简单接受/简单拒绝的区分

在简单接受/简单拒绝的判断阶段，通过测试来判断图元（已经在世界坐标系空间）是否完全在视见体里面，或完全在视见体外面。在流水线的早期阶段判别出是在视见体外面的图元，可以减少后续阶段的处理量。我们还将在裁剪阶段进一步裁剪那些不能简单接受/简单拒绝的图元。

为了简单接受/简单拒绝一个图元，我们需要把每一个变换的顶点与视见体的6个包围平面进行测试。包围平面一般不与坐标轴平行，一个顶点与一个包围平面的测试需要4个乘法和3个加法（一个齐次坐标点与一个3D平面方程的点积）。每一个顶点一共需要$6 \times 4 = 24$个乘法和$6 \times 3 = 18$个加法。

18.3.4 光照处理

光照模型的计算量与采用的明暗处理算法（恒定、Gouraud或Phong）有关，所以必须分别估算：对恒定明暗处理每一个多边形计算一次，对Gouraud明暗处理每个顶点计算一次，对Phong明
868 暗处理每一个像素都要计算。在高性能系统中，通常使用环境、漫反射和镜面光照模型。

在恒定明暗处理中，对整个多边形，仅需要根据光源的位置、多边形的表面法向量和漫反射颜色分量计算一个颜色值（见式(16-9)）。第一步是计算表面法向量和光线方向的点积（对有方向光源，需要3个乘法和2个加法）。如果采用了基于光源距离的衰减因子，我们还必须计算它，并同这里的点积的结果相乘。然后，对每一个红、绿、蓝颜色分量，我们要将该点积同光源亮度、漫反射系数相乘（2个乘法），将环境光亮度同环境反射系数相乘（1个乘法），并将结果相加（1个加法）。如果假设只有一个单方向光源，计算一个RGB颜色需要$3 + 3 \times (2 + 1) = 12$个乘法和$2 + 3 \times 1 = 5$个加法。用Gouraud明暗处理计算一个三角面片需要计算3个RGB颜色——每个顶点一个。

Phong明暗处理要求在每一像素都计算光照强度，而不是每一个多边形或每一个多边形顶点。因此，它的计算量比恒定明暗处理或Gouraud明暗处理大得多。然而，这些计算是在流水线的光栅化阶段进行的。

18.3.5 观察变换

在这一阶段，在世界坐标系的图元被转换到规格化投影坐标(NPC)系。变换中，世界坐标系的每一个点都需要乘以一个4×4的矩阵，这个矩阵结合了需要将世界坐标系转换到NPC坐标系的透视变换（如果使用的话）和任何扭曲或非均匀比例变换。每个顶点需要16个乘法和12个加法。然而，观察变换矩阵有一些项总是0。如果我们利用这一点，我们可以将这一阶段的计算量减少大约25%。我们先假定在观察变换阶段每个顶点需要12个乘法和9个加法。

注意，如果使用简单的光照模型（既不需要计算光源到图元顶点的距离），模型变换和观察变换矩阵可以结合成一个矩阵。这种情况下，显示流水线中只有一个变换阶段，这是一个非常显著的节约。

18.3.6 裁剪

在裁剪阶段，要用视见体来裁剪那些不能被简单接受或拒绝的已计算光照的图元。如第6章所述，裁剪有两个目的：防止一个屏幕窗口的操作影响到其他窗口，和防止在眼睛后面或很远距离的图元引起数学上溢和下溢。

严格的裁剪算法仅对简单图元（如线和多边形）在计算上是有效的。这些图元可以用6.5节给出的任何3D裁剪算法来裁剪。像球和参数化定义的曲面这样的复杂图元是难于裁剪的，因为裁剪会改变图元的几何性质。那些设计上就仅显示三角面片的系统也存在相关的问题，因为一个被裁剪的三角面片可能有三个以上的顶点。 869

严格的裁剪算法以外的另一个选择是3.11节给出的截剪(scissoring)。这样，那些跨越裁剪边界的图元在像素化阶段以前是按正常图元一样处理，在像素化阶段那些在视口窗口以内的像素才被写到显示缓存中。然而，因为一些计算花在了观察窗口以外的像素上，截剪(scissoring)的效率不高。不过，它是能裁剪许多复杂图元类型的惟一可行的选择。

在这里讨论的流水线中，所有的裁剪都是在齐次坐标系中进行的。这实际上仅仅对z方向裁剪是需要的，因为w值需要用于判断顶点是否在眼睛后面。许多系统为了提高效率在齐次坐标除法以后再进行x和y方向的裁剪。这简化了x和y方向的裁剪，但是仍然可以在w信息丢失之前判断图元是否在眼睛后面并完成裁剪。

裁剪需要的计算量与跨越裁剪边界的图元数目有关，每一帧之间都会发生变化。通常假设仅有一小部分图元（10%或更少）需要裁剪。如果这个假设不满足，系统的性能会急剧下降。

18.3.7 除以w并映射到3D视口

通常，已经经过透视变换的齐次坐标点的w值不为1。为了计算真实的x, y, z值，我们必须把每一个齐次坐标点的x, y, z值都除以w。每个点需要3个除法操作。在许多系统中，顶点的x, y坐标必须从裁剪坐标系映射到真正的3D视口。这需要在x, y方向进行简单的缩放和平移操作，每个顶点需要2个乘法和2个加法。

18.3.8 光栅化

光栅化阶段把经过变换的图元转换为像素值，通常将它们存储在一个帧缓存中。如7.12.1节所述，光栅化操作包括三个子任务：*扫描转换*、*可见面判断*和*明暗处理*。原则上，像素化需要计算每个图元对每个像素的贡献，是一个$O(nm)$的运算，其中n是图元的数目，m是像素的数目。

在软件绘制系统中，光栅化可以按照两种顺序进行：一个图元一个图元地进行（物体为序），

或者一个像素一个像素地进行（图像为序）。图18-8的伪代码给出了两种方法的描述，采用了两个主要的具有图像精度的可见性判断算法：z缓存和扫描线算法。

```
for（每个图元P）                    for（每个像素q）
    for（P中的每个像素q）               for（覆盖q的每个图元P）
        根据q点P的可见性更新                计算P to q的贡献，结束
            帧缓存                          时输出q
            a)                              b)
```

图18-8 a)物体为序和b)图像为序的光栅化算法伪代码

870 现在大多数系统采用物体为序的光栅化操作，使用z缓存算法来计算可见性。z缓存算法仅仅是在获得了便宜的DRAM的今天才变得是可行的。我们假设使用z缓存算法进行可见性判断，光栅化阶段的计算量还要依赖于扫描转换和明暗处理方法。扫描转换的计算难于归类，但在一个实际的系统中占有重要的地位。

恒定明暗处理在光栅化阶段不需要额外的计算，因为在光照计算阶段多边形的单一的颜色已经计算好了。Gouraud明暗处理需要在多边形内对红、绿、蓝分量进行线性插值。可以使用增量加法来计算连续像素的RGB值。Phong明暗处理需要更大的计算量。必须利用每个顶点的法向量在多边形内线性插值计算出每一点的表面法向量的x, y, z分量。因为插值出的向量的模一般不是单位向量，还必须进行规格化。每个像素需要两个加法，一个代价昂贵的平方根的倒数运算和3个乘法。因此，必须估计一下Phong明暗处理的计算量，需要一个点积，许多乘法和加法，并且一般需要计算一个指数。如16.2.5节所述，Bishop和Weimer对Phong明暗处理的近似虽然牺牲了表面高光的真实感效果，但可以减少一些计算量。系统可能还需要其他的明暗处理技术，例如透明度和纹理映射。我们将在18.11节讨论这些技术对体系结构的影响。

除了明暗处理的计算，更新每一个像素还需要读z缓存，对新旧z值进行比较，而且如果图元是可见的，将新的颜色值和z值写到帧缓存中。

18.3.9 一个样板应用的性能要求

在18.2节中，我们提到光栅图形系统的三个主要性能障碍：在几何计算中需要的浮点计算量，面向像素的计算量和光栅化中需要的帧缓冲内存存取。为了估计这些问题的影响，我们应该估算一下，在中等刷新速率下显示一个简单数据库时，绘制流水线每一级的计算量。为了进行这样的估算，我们先来定义一个有代表性的数据库和应用。

1. 一个简单数据库

由于三角面片使用广泛，而且它们的计算需求也比较容易估计，我们的简单数据库就使用三角面片作为图元。（然而，一些有共享顶点的复杂的图元，例如三角形带和四边形网格，变得越来越流行。习题18.2探讨了使用这些图元的好处。）我们假设一个中等数据库的大小是10 000个三角面片，现在的工作站可以以交互刷新的速度来绘制这样大小的数据库[AKEL88；APGA88；BORD89；MEGA89]。另外，每一个三角面片平均覆盖100个像素。

跨越裁剪边界的多边形的数目在每帧之间差别很大。为简单起见，我们假设在我们的应用例子中没有图元需要裁剪。这意味着我们低估了裁剪的计算量，但却高估了下面几级的工作量。

871 图像深度复杂度，即映射到一个像素上的平均多边形数目，与数据模型和视图有关。我们任意假设被所有三角面片所覆盖的像素中的一半也被其他的三角面片遮挡。

我们假设使用环境/漫反射光照模型和Gouraud明暗处理来计算每个图元的亮度。（这是目前3D系统采用的最慢的通用标准，但是Phong光照模型与Gouraud明暗处理相结合的技术正变

得越来越流行。）我们假设屏幕的分辨率是1280 × 1024，每秒刷新10帧，这是现在交互式应用的典型标准，但离理想标准差得很远。

总之，我们的应用例子有如下的特点：

- 10 000个三角面片（无被裁剪的）。
- 每个三角面片平均覆盖100个像素，其中一半像素被其他三角面片遮挡。
- 采用环境和漫反射光照模型（而不是Phong模型）。
- Gouraud明暗处理。
- 1280 × 1024分辨率屏幕，每秒刷新10帧。

由于图形处理的许多步难于归类，我们无法计算我们的应用例子对计算量和内存带宽的总的需求。相反，我们集中讨论三个性能障碍：用于几何计算的浮点操作计算量，计算像素值的整型操作计算量和光栅化操作中的帧缓冲内存访问量。

2. 几何计算

对每一帧，我们必须处理10 000 × 3 = 30 000个顶点和顶点法向量。在模型变换阶段，变换一个顶点（包括变换法向量）需要25个乘法和18个加法，因此这一阶段共需要30 000 × 25 = 750 000个乘法和30 000 × 18 = 540 000个加法。

简单接受拒绝/简单判断需要把每个图元的每个顶点与视见体的6个边界面进行测试，每个顶点共需要24个乘法和18个加法。因此不管有多少图元是可见的或不可见的，这一阶段共需要30 000 × 24 = 720 000个乘法和30 000 × 18 = 540 000个加法。

每个顶点的光照计算需要12个乘法和5个加法，一共30 000 × 12 = 360 000个乘法和30 000 × 5 = 150 000个加法。

对每个顶点的观察变换需要8个乘法和6个加法，一共30 000 × 8 = 240 000个乘法和30 000 × 6 = 180 000个加法。

裁剪的计算量变化较大，准确的计算量与不能通过简单接受或拒绝判定的图元数目有关，这又依赖于场景和观察角。我们已经假设我们的数据库是最简单的情况，既所有的图元都完全在视见体以内。如果大部分图元需要裁剪，计算量是非常大的（甚至比几何变换阶段的计算量还要大）。

每个顶点的规格化操作需要3个除法，一共30 000 × 3 = 90 000个除法。将每个顶点映射到3D视口需要2个乘法和2个加法，一共30 000 × 2 = 60 000个乘法和30 000 × 2 = 60 000个加法。 872

整个几何子系统阶段每帧的浮点计算量是2 220 000个乘法/除法和1 470 000个加法/减法。因为每1/10秒要计算一帧，一共需要每秒计算22 200 000个乘法/除法和14 700 000个加法/减法（总共36 900 000个浮点操作），这是一个非常大的计算量。

3. 光栅化计算和帧缓存存取

现在让我们来估计一下绘制每帧需要的像素计算量和帧缓冲内存访问量。我们假设z值和RGB值能够打包在帧缓冲内存的一个字（32位）中（现在大部分高性能系统都支持这一功能）。对每个初始可见的像素（即需要更新帧缓存的结果），需要计算z, R, G, B值（如果使用前向差分每像素需要4个加法），从显示缓冲内存中读取z值（一个显示缓冲内存周期），对z值进行比较（一个减法），并写入新的z值和颜色值（两个显示缓冲内存周期）。对每一个初始不可见的像素，仅需要计算z值（一个加法），从显示缓冲内存中读取z值（一个显示缓冲内存周期），比较两个z值（一个减法）。注意，初始可见的像素可能被覆盖，但初始不可见的像素却永远不会暴露出来。

由于我们假设每个三角面片的像素在最终的场景中有一半是可见的，3/4的像素是初始可见

的和1/4的像素是初始不可见的就是一个合理的估计。每一个三角面片覆盖了100个像素，那么3/4 × 100 × 1000 = 750 000个像素是初始可见的，1/4 × 100 × 1000 = 250 000个像素是初始不可见的。因此，要显示整帧场景，共需要(750 000 × 5) + (250 000 × 2) = 4 250 000个加法和(750 000 × 3) + (250 000 × 1) = 2 500 000个帧缓冲存取周期。为初始化每一帧，需要清除颜色和z缓存区，另外需要1280 × 1024 × 2 = 2 600 000个帧缓冲存取。因此，每帧共需要2 500 000 + 2 600 000 = 5 100 000个帧缓存存取。如果每秒绘制10帧，每秒需要执行42 500 000个加法和5 100 000个帧缓存存取。

在1989年，最快的浮点处理器大约每秒可计算20 000 000个浮点运算，最快的整型处理器每秒大约可计算40 000 000个整型运算。DRAM内存系统的一个周期大约是100ns。因此，我们的应用例子对浮点和整型计算的需求正好在单个CPU所能达到的计算性能的极限，但对帧缓冲内存的存取却远远超过了一个常规内存系统的能力。我们前面已经指出，对1989年的系统来说，这样的数据库的规模仅仅是中等大小。在下面的几节，我们将说明如何利用多处理技术来获得能够显示这样规模或更大规模的数据库的能力。

18.4 多处理简介

以高刷新速率显示大数据库需要很高的系统性能，包括计算能力和内存访问带宽。我们已
873 经看到，图形系统几何处理部分对计算能力的需求超过了单CPU的计算能力。同样，光栅化对内存带宽的需求也超过了单内存系统的性能。要达到这样性能水平的惟一途径是同时执行多个运算和同时对内存进行读写——我们需要并行处理。

并行处理，即多处理，是几乎所有高性能图形体系结构的基础。多处理有两个基本形式：流水线（pipeline）和并行（parallelism）（我们一般为多处理保留术语concurrency）。一个流水线处理器包含了一系列处理单元（PE），这些PE按流水线方式排列，一个PE的输出是下一个PE的输入（见图18-9a）。并行处理器的PE并排排列，并同时对数据的不同部分进行运算（见图18-9b）。

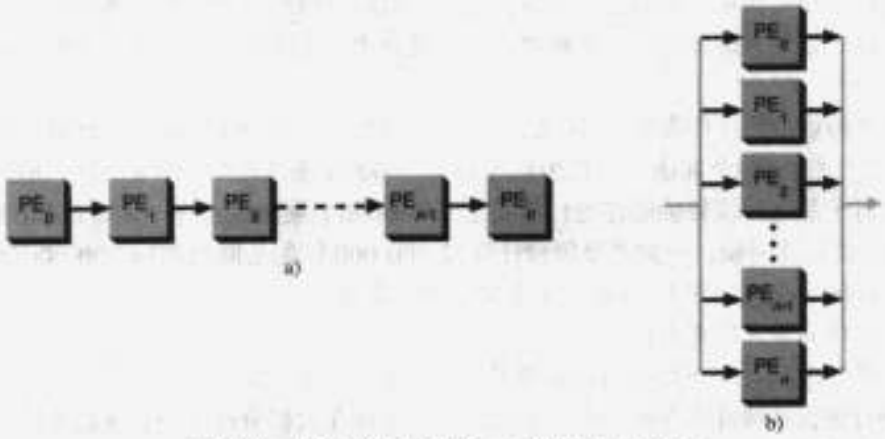


图18-9 多处理的基本形式：a)流水线，b)并行

18.4.1 流水线

要以流水方式完成一个计算，我们要将它划分成能分别在PE中顺序执行的几个阶段。显然，流水线的速度是由它的最慢一级决定的，因此处理负载必须平均分配到PE上。如果不能做到这一点，可以按照PE必须承担的任务使用不同性能的PE。

流水线系统中最重要的问题是吞吐量和延迟。吞吐量是数据处理的总速率；延迟是指一个

数据单元从流水线开始传送到最后所需要的时间。有些计算可以用许多级的流水线以获得高吞吐量。然而，流水线的延迟随着其长度增长而增长，并且有些计算对时间延迟有限制。例如，像飞行模拟器这样的实时系统必须尽快地响应飞行控制器的变化。不考虑帧刷新速率，如果在流水线中同时绘制超过1到2帧的数据，就可能削弱了系统的可交互性。 874

18.4.2 并行性

要并行化一个计算，我们要将数据分为能在不同的PE中独立处理的几个部分。通常，PE执行相同的程序。*同构*的并行处理器包含相同类型的PE；*异构*的并行处理器包含不同类型的处理器。在任何并行系统中，整体计算速度是由最慢的PE完成任务的时间决定的。因此，PE之间的负载平衡是十分重要的。

同构处理器之间的另一个差别是：处理器是锁步运行还是独立运行。锁步运行的处理器一般共享一个指令存储，被称为*单指令多数据流*（SIMD）处理器。独立执行的处理器每个PE都有独立的指令存储，被称为*多指令多数据流*（MIMD）处理器。

1. SIMD处理器

因为SIMD中的所有PE都共享一个指令存储，SIMD处理器一般比MIMD处理器便宜。然而，它们并不能很好地执行具有条件转移分支和指针或间接访问数据的算法。因为条件分支的执行路径与PE上的特定数据有关，不同的PE可能执行不同的分支路径。由于SIMD的所有PE必须锁步执行，它们必须执行所有可能的分支。为了兼容条件分支，PE一般用一个*开关寄存器*来控制写操作。只有那些开关寄存器置位的PE才写入计算结果。通过适当地置位和清除开关寄存器，PE就能够执行条件分支（见图18-10a）。

```
    语句 1;
if not  条件  then
  enable = FALSE;
    语句 2;
toggle enable;
    语句 3;
    语句 4;
enable = TRUE;
    语句 5;
    语句 6;
```

总操作数
条件求值为TRUE时为5
条件求值为FALSE时为10

a)

```
    语句 1;
if  条件  then
      语句 2;
else
  begin
        语句 3;
        语句 4;
  end;
    语句 5;
    语句 6;
```

总操作数
条件求值为TRUE时为5
条件求值为FALSE时为6

b)

图18-10 相同算法的a)SIMD和b)MIMD表示。在SIMD程序中，条件分支被转换成开关寄存器的操作。当一个特别的PE的开关寄存器状态是FALSE时，PE执行当前指令，但结果将不被保存

只有有少量条件分支的算法能够在SIMD上有效地运行。然而，有许多条件分支的算法的效率却非常低，因为在任何时候可能大多数PE的开关都是关闭的。包含指针的数据结构（例如链表或树）或者索引数组有同样的问题。由于每个PE上的指针或数组下标的值可能是不一样的，必须列举出所有可能的值以保证任何PE都能访问它要求的内存地址。对大的数组或指针，这是对处理资源的极大浪费。有一些SIMD处理器为每个PE提供独立的地址线来避免这个问题，但是这增加了系统的规模和复杂度。 875

2. MIMD处理器

由于每个PE必须有它自己的指令存储和控制器，所以MIMD处理器比SIMD处理器贵的多。

在一个MIMD处理器中的PE经常执行同样的程序，但不像SIMD中的PE，它们不受锁步执行的限制。由于有这样的自由，MIMD处理器在执行条件分支时不会有任何麻烦；每个PE进行独立的控制流判断，跳过不需要执行的指令（见图18-10b）。因此，MIMD处理器在通用类型的计算方面能获得很高的效率。然而，各处理器的启动和终止时间可能是不一致的，它们处理数据的速率也不一样，同步控制和负载平衡就变得很困难，经常需要在PE的输入或输出使用FIFO缓冲区。

18.4.3 多处理器图形系统

流水线和并行处理器是几乎现在所有高性能图形系统的基本模块。这两种技术都可以用来对图形系统的前端子系统和后端子系统进行加速，如图18-11所示。

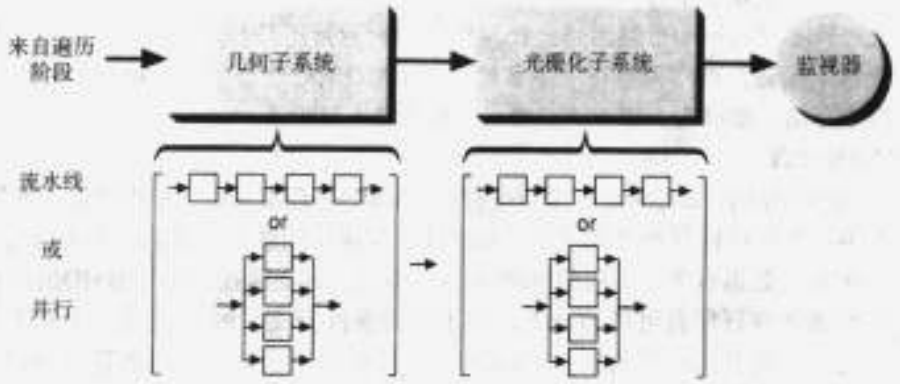


图18-11 用流水线和并行方法来加速一个图形系统的前端和后端

在下面几节中，我们将依次探讨这些策略。18.5节和18.6节讨论流水线和并行的前端体系结构，18.8节和18.9节讨论流水线和并行的后端体系结构，18.10节讨论基于混合并行技术的后端体系结构。

876

18.5 流水线前端体系结构

回忆一下18.3节的内容，图形显示系统的前端子系统有两个主要的任务：遍历显示模型和把图元转换到屏幕空间。就像我们已经看到的那样，要获得现在应用所要求的绘制速率，我们必须使用并行技术来加速这些计算。几十年来，流水线和并行技术都已经用于高性能图形系统的前端子系统。由于前端本质上是具有流水性质的，其各个阶段的任务能够分配给分离的硬件单元。大部分图形数据库中的大规模图元也可以分布到多个处理器上并行处理。在本节，我们讨论流水线前端系统。我们将在18.6节讨论并行前端系统。

在图18-7给出的标准图形流水线中，我们指出它提供了一个非常有用的绘制流程概念模型。由于它的线性性质和处理负载分配的平衡性，它同样能很好地映射到一个处理器的流水线物理上。[MYER68]中指出，从20世纪60年代以来，这就是一种构造高性能图形系统的非常流行的方法。流水线的每级可以通过几种方式来实现：一个单独的通用处理器，一个定制的硬件单元，一个流水线或并行处理器本身。我们现在来讨论前端流水线每一级的实现。

18.5.1 应用程序和显示遍历

一些处理器必须要执行一些驱动整个图形系统的应用程序。这样，处理器除了要驱动图形流水线以外，一般还要处理输入设备、文件I/O和与用户的所有交互。在使用瞬时模式的系统中，显示模型一般存储在CPU的主存中。因此，CPU必须在运行应用程序的同时遍历模型数据。在使用保留模式的系统中，模型一般（但不总是）存储在显示处理器的内存中，显示处理器来

执行模型遍历。在7.2.2节已经讨论过，尽管这样的系统不灵活且有其他的限制，但由于使用了两个处理器来完成任务，这样的系统一般比较快。

当需要更高的性能时，单处理器可能就不能胜任以足够的速度来遍历整个数据库。惟一的办法是把数据库分成几部分，并行遍历。18.6.1节将讨论这种相对较新的技术。

18.5.2 几何变换

几何变换阶段（包括模型变换和视见变换）的计算量非常大。幸运的是，向量和矩阵乘积的计算比较简单，不需要分支和循环，并且能比较容易地在硬件上实现。

这些阶段最普遍的实现是用一个单CPU或功能单元对一系列点进行变换。这种类型的早期处理器是Matrix Multiplier（矩阵乘法器）[SUTH68]，它可以在20ms内完成一个四元素的齐次 877
向量和变换矩阵的乘法运算。从此以后还开发了其他专用的几何处理器，其中最著名的是Clark的Geometry Engine（几何引擎）。它还具有裁剪功能（见18.5.5节）。最新的几何处理器已经利用了商业化可编程浮点芯片的强大能力。

如果流水线不能够提供足够的处理能力，变换计算可以按几种方式实现并行化：

- 顶点的各个分量可以并行计算。使用四个处理器，每个都存储一个当前变换矩阵，并行计算x, y, z和w。
- 多个顶点能够并行变换。如果图元都是统一形式的，如三角形，则每个三角形的三个顶点可以同时变换。
- 整个图元可以并行变换。如果有n个变换引擎，每隔n个图元由一个处理器负责变换。这种技术既有并行前端系统的优点也有其缺点，我们将在18.6节讨论。

18.5.3 简单接受/简单拒绝的区分

简单接受和简单拒绝测试易于实现，因为判断一个顶点在一个裁剪面的哪一侧，最多需要一个点乘积，最少只需要一个浮点数比较（或减法）。由于这些测试只需要很少的计算，它们通常由负责图元变换的处理器来执行。

18.5.4 光照处理

像几何变换一样，光照计算也非常直观，并且需要大量的浮点计算。一个特殊设计的硬件处理器能够根据多边形的颜色和光的颜色计算出顶点的颜色。更一般地，光照计算是用一个可编程浮点处理器来完成的。在低性能系统中，也可能使用同一个处理器来变换顶点和进行光照计算。注意，如果使用Phong明暗处理，光照计算要推迟到光栅化阶段才进行。

18.5.5 裁剪

多边形裁剪从前被认为是一件十分麻烦的事情，因为裁剪过程中可能会改变顶点的数目，而且凹多边形还可能在裁剪中分裂成几个多边形。Sutherland和Hodgman[SUTH74]指出，用一个处理单元多次对多边形的顶点进行处理，可以将任意的凹或凸多边形裁剪到一个凸视见体中。每一次用不同的面对多边形进行裁剪。1980年，Clark将这一循环过程分解成一个使用相同处理器的简单流水线，每一级都能集成在一个VLSI芯片，他将这个芯片命名为Geometry Engine（几何引擎）[CLAR82]。几何引擎比较通用，它除了能对图元进行变换，还可以执行透视除法。 878

使用几何引擎（或简单处理器）进行裁剪的方式，可以是一个处理器分别用所有的面对每个多边形进行裁剪，也可以是用一个裁剪处理器的流水线，每一个负责一个裁剪面。图形系统所采用的技术能影响它在最坏情况下的性能：采用一个裁剪处理器的系统，在绘制大量图元需要裁剪的帧时，可能陷入停顿，而每个裁剪面用一个裁剪处理器的系统却能全速运行。然而，后一种方法的裁剪处理器在大部分数据库和视图下都是空闲的。

最近，通用浮点处理器已经开始替代定制的VLSI变换和裁剪处理器。例如，许多年一直在研究专用前端处理器的Silicon Graphics，在1989年也在它的POWER IRIS系统中使用Weitek 3332浮点芯片做变换和裁剪（将在18.8.2节中详细讨论）。由于性能和价格之间的微妙平衡，现在更倾向于使用批量化处理器。将来，如果进一步提出了不能用通用处理器经济地实现的特殊图形功能需求，这种平衡还可能再次改变。

18.5.6 除w并映射到3D视口

像几何变换和光照处理一样，这一阶段的计算比较直观，但需要大量浮点资源。即使是对大部分浮点处理器，浮点除法也是十分耗时的（许多处理器用迭代的方法来做除法）。同样，可以用定制功能单元或商业化浮点处理器来实现这一级。在高性能系统中，这些计算可以分别在流水线处理器上执行。

18.5.7 前端流水线的限制

尽管流水线是构造高性能前端系统的主要方法，它还是有几个值得考虑的限制。第一，前端系统的每一级使用不同的算法。因此，必须设计各种各样的硬件功能单元；如果使用可编程处理器，就必须编写加载到每个处理器中的不同的程序。不论哪种情况，处理器或功能单元的处理能力必须很好地同它们的任务相匹配，否则就会出现瓶颈问题。

第二，由于绘制算法是为硬件设计的（或至少是固件，因为很少有系统允许用户对流水线处理器重编程），难于给它添加新的功能。即使用户流水线处理器支持编程，系统中的硬件也不足以支持一些新的特征，如复杂图元和图元间碰撞检测等。

流水线前端系统的最后一个缺点是，当显示遍历不能由单个处理器执行时，系统将瘫痪，而且在某些性能水平下这是不可避免的。例如，如果我们假设使用一个20 MHz的处理器和内存系统来执行显示遍历，在数据库中描述一个三角形需要40个字的数据（包括顶点坐标、法向量、颜色等），将每个字发送到流水线需要两个内存/处理器周期（用一个周期将它从内存中读出，用
879 另一个周期把它加载到流水线上），那么不管在流水线中使用功能多么强大的处理器，系统每秒最多可以显示20 000 000/(2 × 40) = 250 000个三角形。现代系统的需求很快就会达到这个限制。

那么我们怎么做才能获得更高的系统性能呢？除前端计算流水线化以外，另一个选择是将它们并行化。下面的一节将讨论这个构造高性能前端系统的第二种方法。

18.6 并行前端体系结构

由于图形数据库是规则的，它一般包含大量的图元，对这些图元的处理几乎一样，另一种实行并行处理的方法把数据分成多数据流，并分别处理。对前端子系统的大多数阶段，这样的划分是很容易实现的；例如，几何变换阶段可以使用18.5.2节中给出的任何并行化技术。然而，那些将数据流分开（显示遍历）或合并（在前端和后端子系统之间）的阶段是有问题的，因为它们必须提供整个数据带宽。

18.6.1 显示遍历

几乎所有的应用程序都假设单个连续的显示模型或数据库。在并行前端系统中，最简单的技术是用单个处理器来遍历数据库（串行遍历），并将图元分布到并行处理器上。不幸的是，这种串行遍历可能成为并行前端系统的瓶颈。可以用几种技术来加速串行遍历：

- 优化遍历程序，或用汇编语言编写其代码。
- 数据库可以存储在快速内存中（例如，SRAM而不是DRAM）。
- 使用一个快速的（或者一个为特定数据格式优化的）遍历处理器。

如果这些优化技术还不够，惟一的选择是并行遍历数据库。数据库可以存储在允许多个处理器并行遍历的单个内存系统中（共享内存模型），也可以分布到有自己内存的多个处理器中（分布式内存模型）。

共享内存方法的优点是，尽管遍历必须分布到多个处理器上，数据库还是存储在一起。一般，每一个处理器遍历数据库的一部分。但是，从层次数据库模型继承下来的特点意味着处理器不得不为访问同样的数据而竞争。例如，每个处理器必须访问当前变换矩阵和其他观察和光照参数。由于共享内存的数据带宽可能不比常规内存高多少，共享内存的方法可能不能提供足够的性能。

在分布式内存方法中，每个处理器在它的局部内存中存储一部分数据库。它只遍历每帧数据库中自己的一部分，也可能执行其他的前端计算。然而，分布式数据库有它自己的问题：除非系统为应用程序员提供一个连续数据库的应用程序接口，否则它就不能支持简单图形库。同 880
样，各个遍历处理器上的负载必须是平衡的，才能保证系统的资源被充分利用。层次数据库进一步恶化了这些问题，因为层次结构中一层的属性影响它下面的图元，并且深层的结构可能被多个高层结构所调用。

下面的两节讨论将层次数据库分布到多个处理机上的两种方法：按结构分布，即分给每个遍历处理器层次结构一个完整分支；或按图元分布，分给每个遍历处理器层次结构中每一块的一部分。

1. *按结构分布*

表面上看，按结构分布很吸引人，因为结构中的状态转换元素明显只需要存储一次。然而这只是一种错觉，因为多个高层的结构可能引用同一个低层的结构。例如，一个数据库包含几辆汽车，用分别的*汽车*结构来描述每个小汽车，这样可以将每个*汽车*结构分布到不同的处理器上。然而，如果每个汽车结构引用了几个*车轮*结构，*车轮*结构就必须复制到每个处理器上。

处理器间的负载平衡也比较困难。因为一个结构中的图元有很大的区域相关性，改变场景中的视点和几何物体可能使得结构的整个部分可见或不可见。保持多处理器之间的平均负载可能需要对数据库的一部分进行动态重分配。

2. *按图元分布*

按图元分布的代价很高，因为数据库的整个层次结构和任何状态转换命令必须复制到每个处理器上。对数据的改变必须发送到每个处理器，所以结构的编辑也是十分昂贵。然而负载平衡是自动的。因为一个层次数据库中的物体一般由大量的简单图元组成（例如，多边形组成的一个平铺的表面），所以这些图元将分散到所有的处理器上，并且每个处理器都有相同的处理负载。

并行显示遍历是一个相对较新的技术。在1989年，高性能体系结构才达到了一定性能水平，使得串行遍历的性能达不到要求。只有少数的系统才实验使用并行遍历[FUCH89]。我们所描述的层次数据库分布技术都不是理想的。同容易划分成并行任务的几何处理相比，显示遍历就比较困难。不过，随着系统性能水平的提高，并行遍历越来越重要。

18.6.2 重组并行数据流

绘制流水线前端和后端部分的过渡同样是比较麻烦的问题。在并行前端系统中，经过图元变换和裁剪的多数据流，必须导入执行光栅化的处理器中。如果使用不同的处理器来分别处理不同的屏幕区域，就需要按图元的空间信息将它们分类。

在并行前端系统中的第二个困难是数据在经过流水线以后，这些数据的顺序可能会改变。例如，一个变换两个小图元的处理器可能在变换一个大图元的处理器之前完成变换。这对于大 881
多数图元和绘制技术都是没有问题的。然而，一些全局的命令，例如刷新其中一个窗口和交换

缓冲区的命令等，需要在命令之前和之后对数据进行同步处理。如果这样的命令大量出现，就需要一些支持同步的硬件。Raster Technologies的一个系统[TORB87]在每个PE中包含了一个特殊的FIFO，它们为每个命令存储了标志码，并允许这些命令被不同的PE处理以后再重新同步。

18.6.3 流水线同并行性的比较

我们已经看到，流水线和并行技术都可以用来构造高性能的前端子系统。在过去的几十年中，尽管流水线是这些系统的主要应用技术，但并行有它的几个优点，包括对不同算法的可配置性（因为一个处理器处理所有的前端操作），和模块化结构（因为并行系统比流水线系统更容易使用相同的PE）。由于流水线系统的性能受其最慢一级的吞吐量的限制，流水线并不像并行系统那样具有扩展性。另一方面，并行系统需要更复杂的同步和负载平衡算法，并且不能够使用像流水线系统中的特殊的处理器。两种设计在将来都是有用的，实际上，高性能的系统更可能结合使用这两种技术。

18.7 多处理器光栅化体系机构

回忆一下，前端子系统的输出一般是一些屏幕坐标中图元的集合。光栅化（后端）子系统将每个图元进行扫描转换，决定每个像素上哪个图元是可见的，计算其相应的亮度值，创建最后的图像。18.2.4节给出了两个基本的原因，说明了为什么简单的显示处理器/帧缓存系统不足以构造高性能的光栅化子系统：

1. 单显示处理器没有处理所有像素计算的能力；

2. 即使显示处理器能够足够快地计算像素，帧缓存也没有足够的内存带宽来处理像素的通信。

过去几十年中，许多对图形体系结构的研究都集中在如何克服这些限制上。已经提出了各种各样的技术，有些技术已经在商业或实验系统中实现。在本节中，我们将讨论一些低成本的、中等性能的、在硬件中实现一般算法的体系结构。在18.8节和18.9节，我们将应用大量的并行技术来加速算法的“内循环”计算，寻找一些提高性能的方法。在18.10节，我们将讨论结合了多种技术的混合体系结构，以提高系统效率，获得更高的性能。图18-12总结了我们将要讨论的并行方法。

882

光栅化算法	体系结构技术		
	串行流水线	高度并行化	混合方法
以物体为序 z缓存，深度排序和BSP树算法	以物体为序流水线 多边形/边/区间处理器流水线	图像并行 分割的图像内存 逻辑增强的内存	虚拟缓冲/ 虚拟处理器 并行虚拟缓冲
以图像为序 扫描线算法	以图像为序流水线 扫描线流水线	物体并行 每个图元流水线一个处理器 树结构化	图像合成

图18-12 并行光栅化方法的分类

18.7.1 以物体为序的流水线体系结构

在光栅化计算中加入并行的最直接的方法是将软件算法的几个步骤映射到硬件流水线上。

这种技术已经被用于构造一些便宜的、中等性能的系统。这种方法中可以使用任何两种主要的光栅化方法：以物体为序（z缓存、深度排序和BSP树算法）和图像为序（扫描线算法）。我们现在来讨论物体序光栅化方法，并在18.7.2节讨论图像序光栅化方法。

物体序光栅化方法包括z缓存、深度排序和BSP树算法（目前为止，z缓存是3D系统中最流行的）。这些算法的外循环是枚举数据库中的图元，内循环是枚举每个图元的像素。对于多边形的绘制，这些算法的核心是光栅化单个的多边形。

图18-13给出了最常用的凸多边形光栅化算法。这个算法是3.6节给出的2D多边形扫描转换算法的扩展。该算法使用定点算术，而不是整数算术。使用Delta增量方法依次按扫描线顺序和像素顺序以累加方式计算x, z, R, G, B。我们将描述该算法的每一步。

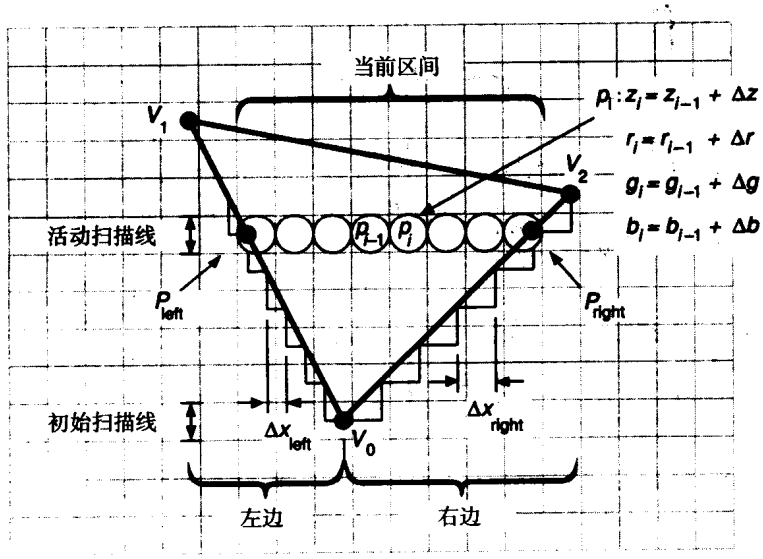


图18-13 光栅化的一个三角形。每个顶点（V_0, V_1和V_2），区间端点（P_{left}和P_{right}）和像素（p_0, p_1等）处的值都有z, R, G, B分量

1. 多边形处理

每个多边形执行一次的计算被归到这一类。第一步是找出同多边形相交的起始扫描线（这是由具有最小y值的顶点决定的）。大多数情况，多边形同扫描线只在一个像素处相交，并有两个在投影面上向上的边，即左边和右边。可以计算出每条边的x, z, R, G, B的Delta增量。这些Delta增量有时被称为斜率。

2. 边处理

每条扫描线执行一次的计算被归到这一类。每个图元内的扫描线被依次处理。前面计算的Delta增量被用来计算左边和右边同当前扫描线（图中的P_{left}和P_{right}点）交点的x, z, R, G, B值。在扫描线上的一系列连续的点，例如P_{left}和P_{right}之间的点，被称为跨度（span）。用于跨度内依次计算各像素z, R, G, B值的Delta增量可以用P_{left}和P_{right}处的值来计算。 883

3. 跨度处理

必须对跨度内每个像素进行的计算在这里进行。对跨度内的每个像素，z, R, G, B的值是将Delta增量依次加到前驱像素的值来计算的。z值同此像素原来的z值进行比较，如果这个z更小，新的像素值就可以替代原来的。

对上述三步的每步都使用一个PE的流水线系统生成图像的速度比通用目的的显示处理器要

快得多。实际上，它生成像素的速度要超过一个标准帧缓冲内存的处理速度。使用这种方法的Hewlett-Packard SRX[SWAN86]能够每秒生成20 000 000个像素，大约是一般VRAM内存系统速度的两倍，光栅化阶段的瓶颈是帧缓冲内存存取。

4. 像素高速缓存

如果帧缓冲内存在某种程度上可以并行访问，像素的读写可以快一些。一种方法是将内存划分为多个（如16个）分区。每个分区存储每隔4个扫描线，一条扫描线上每隔4个像素的那些像素，或者是存储一条扫描线每隔16个像素的那些像素（见图18-14）。按这种方法，16个像素可以被并行读写。这种技术叫作内存交叉存取，它也被用于通用CPU内存设计。

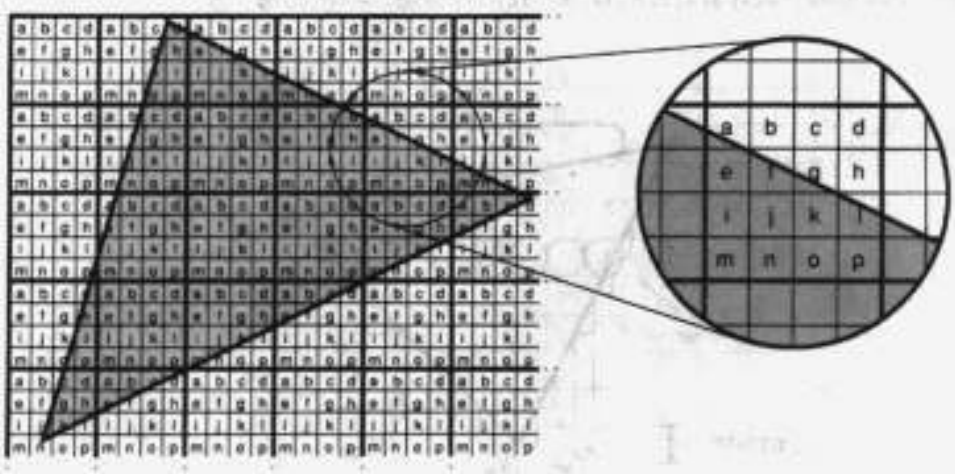


图18-14 4×4的交叉存取内存的组织。每个内存分区（"a"到"p"）存储每个4×4像素块中的一个像素

在光栅化流水线和交叉存取的图像内存之间可以插入一个存储16个像素的像素寄存器或像素高速缓存[GORI87, APGA88]，如图18-15所示。如果像素已经在高速缓存中，高速缓存就可以使光栅化硬件高速访问单个像素。多个像素值可以在高速缓存和帧缓存之间，按照与帧缓存相适应的低速并行移动。



图18-15 像素高速缓存使得与光栅化硬件的高速串行连接和与帧缓存的低速并行连接的带宽相匹配

像任何一个高速缓存内存单元一样，系统的性能与访问的相关性有关，即连续的内存访问很有可能访问内存的同一部分。不确定的访问模式能引起大量的高速缓存丢失，并降低系统的性能。对于多边形绘制，由于在像素访问之前就可以知道多边形在屏幕空间的范围和生成像素的顺序，内存访问的模式能够被精确地预测。利用这一信息，高速缓存控制器能够在上一个像素块被处理的同时，开始从帧缓存中读下一个像素块[APGA88]。

通过并行访问帧缓冲内存的方法来增强光栅化子系统的性能，能增大系统的吞吐量，以至使得光栅化硬件和像素内存之间的单像素通路成为新的瓶颈。符合逻辑的下一步是增强光栅化器的能力以使得它能够并行生成多个像素。我们将在18.8节中考虑这种图像并行体系结构。

18.7.2 以图像为序的流水线体系结构

图像序光栅化方法之外的另一个选择是15.4.4节介绍的图像序（或扫描线）光栅化方法。扫描线算法一个像素一个像素地计算图像，而不是一个图元一个图元地计算。为了剔除那些不为当前扫描线做出贡献的图元，大部分扫描线算法需要将图元转换到屏幕空间，并按照它们出现的第一条扫描线的顺序存储在排序桶中。

884 ~ 885

扫描线算法能以用与物体为序算法相同的方法的硬件实现：将软件算法的每步映射到流水线的一系列硬件单元上。许多硬件扫描线系统的先驱工作是由犹他大学在20世纪60年代后期完成的[WYLI67；ROMN69；WATK70]。

图18-16是一个典型的扫描线光栅化器的结构图。*y*排序器将每个多边形的每条边按照它们第一次出现的扫描线的顺序放到排序桶中。活动段生成器从排序桶中读出边，并维护一个当前扫描线的活动边表。从这个表中，可以构造出一个活动段表（一个段是一个多边形内部的一段扫描线），这些段按它们的左端点的*x*值排序。可见跨度生成器（在犹他大学的系统中也叫深度排序器）遍历活动段表，在需要的时候比较*z*值，并生成当前扫描线的一系列可见跨度。明暗处理器使用Gouraud方法来计算这些跨度的亮度值，生成在视频显示器上显示的像素流。

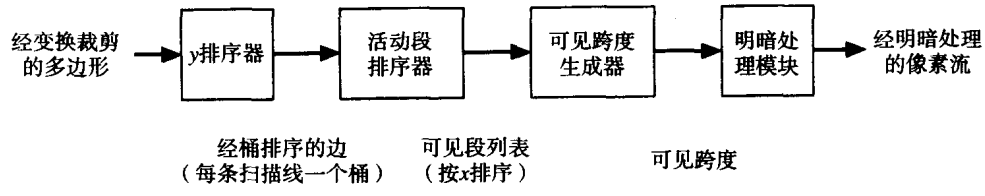


图18-16 一个基于流水线的扫描线像素化器的结构图

注意，只要系统能够以视频刷新的速率生成像素，这样的系统就可以没有帧缓存。最初的Utah扫描线系统在处理中等规模的多边形数据（大约1200个）时，可以实时生成视频信号。然而，像素的生成速率与局部场景复杂度有关，仍然需要一个小的缓冲区（例如足够存储一条扫描线）在一条扫描线中对像素生成速率进行平均。采用双缓冲技术的帧缓存使得像素生成与图像显示完全无关。这种体系结构也是Evan & Sutherland计算机公司在20世纪70年代建立的几代飞行模拟器的基础。

18.7.3 流水线光栅化的限制和对并行性的需求

有两个因素限制了流水线方法加速的可能。第一，大多数光栅化算法仅能够容易地划分成很少的几步。第二，其中有几步的工作量，特别是光栅化算法内循环的几步，要远远超过其他几步。因此，完成这几步任务的处理器就成为系统的瓶颈。

对象序（*z*缓存）系统的内循环在跨度内计算像素，其内循环是一个处理扫描线上活动边的图像序（扫描线）系统。要想使得光栅化系统的加速超过简单流水线系统的可能水平，就必须将这些内循环的计算分布到多个处理器上。在*z*缓存区系统中，这将产生图像并行性；在扫描线系统中，这将产生对象并行性。下面的两节将分别讨论这些方法。实际上，目前几乎所有的高性能图形系统使用了它们的变种。

886

18.8 图像并行光栅化

图像并行一直是设计高速光栅化体系结构的很有吸引力的方法，因为可以按多种方法来并

行生成像素。在这样的体系结构中有两个主要的问题：(1)怎样来划分屏幕？（按行？按列？或按交叉方式？）(2)需要多少个划分？在下面几节，我们将阐述那些经过大量调查的选择，讨论每种方法的优点和缺点。而且，我们将讨论哪种方案正在接近现代体系结构的基本限制。注意，由于图像并行系统是以物体为序进行光栅化，所以需要一个帧缓存来存储中间结果。

18.8.1 内存划分体系结构

有两种明显的划分像素的策略：划分为连续的块（图18-17a)[PARK80]和划分为交叉的棋盘状（图18-17b）[FUCH77a]。无论哪种方法，一个处理器（或PE）都与一个帧缓存的划分相关联。这样的组织结构提供了并行处理，从而提高了图形系统的计算能力，而且为每个PE提供了访问其局部帧缓存的单独通道，从而提高了内存带宽。在光栅化中，多边形从前端系统并行地传送到各个PE，每个PE处理在其帧缓存内的部分。

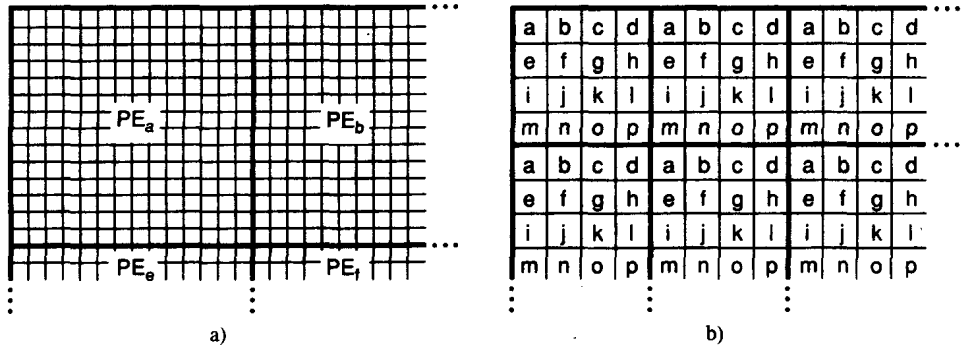


图18-17 帧缓存划分的两个方案。a)每个处理器分配连续的像素块；b)以交叉方式为处理器分配像素

1. 连续划分

在连续区域划分方案中，图元仅需要在它们可能可见的区域中进行处理。这些区域可以
887 通过几何范围来快速确定。如果图元比区域小，每个图元很可能落入一个区域内。大的图元可能落入多个区域。如果区域的大小选择合适，每个处理器处理的图元的数目可能大约是m/p，其中m是数据库中图元的数目，p是处理器的数目。但是注意，如果视点不幸被选在使得所有的图元都落入一个屏幕区域中，一个处理器就必须完成所有图元的光栅化，并且系统的性能极大地降低。在一个连续区域划分系统中，帧刷新速率是由最繁忙区域处理的图元数所决定的。

2. 交叉划分

另一方面，在交叉区域划分中，除了很细小的多边形，几乎所有的多边形都在所有的帧缓存划分中，交叉划分可以获得更好的负载平衡。因为每个处理器都要处理所有的图元（尽管只是它的一小部分像素），这种方案在最好情况下不如连续区域方法的效率高；然而，在最坏情况下，它的性能却有很大提高，因为它的性能与图元的总数有关，而不是与在繁忙区域中的图元数目有关。交叉系统已经成为主要的内存划分体系结构。

18.7.1节描述的多边形扫描转换算法，在进行像素计算以前需要进行初始化计算来获得Delta增量和区间端点。这些计算需要对每个多边形或每个跨度计算一次，而且可以在几个PE之间共享。最初提出的交叉划分内存体系结构[FUCH77a；FUCH79]没有考虑到将这些计算从PE中分离出来（见图18-18）。因为每个PE必须对每个多边形执行整个光栅化算法，所以有大量的冗余计算。

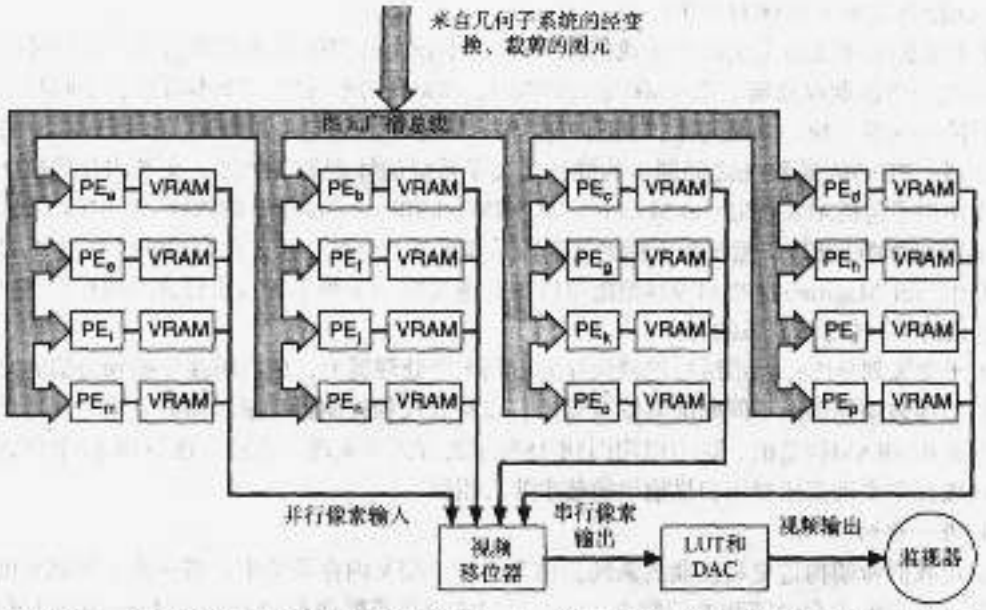


图18-18 一个典型的交叉存储内存系统的框图。每一个PE负责一个4×4像素块中的一个像素

Clark和Hannah[CLAR80]增强了这种体系结构，以利用多个PE之间的相同计算。在他们的方法中，加入了两级额外的处理器来进行多边形和边的处理。一个多边形处理器从前端子系统接收原始的经变换的多边形数据，计算多边形的起始扫描线、边的斜率等。8个边处理器（在8×8矩阵的像素处理器中每列对应一个）计算扫描线区间端点的x, z, R, G, B值。边处理器将区间信息传送到单独的PE（跨度处理器），跨度处理器沿扫描线插值计算像素值。多加的两级处理使得PE能够仅执行像素生成所需要的计算，大大提高了效率。Silicon Graphics的最新高性能系统的光栅化部分就使用这种方法（见18.8.2节）。

3. SIMD与MIMD的比较

这类系统最主要的不同是用SIMD还是用MIMD做PE。我们先考虑一下SIMD处理器。图18-14说明了如何将像素按4×4的交叉划分方式映射到处理器上。用一个SIMD处理器，16个PE对一个4×4的像素块同时进行处理。这种安排有时被称为足迹处理器(footprint processor)，因为就像4×4矩阵的处理器（足迹）走过多边形，同时印出16个像素。注意，如果4×4块中的任何一个像素需要刷新，足迹就必须访问这一块。例如，在图18-14的插入图所示像素块中，在处理器$e, f, i, j, k, l, m, n, o, p$处理它们相应的像素的时候，必须关掉处理器$a, b, c, d, g, h$。

SIMD处理器的一个缺点是它不能充分利用所有的PE。这是因为两个原因。第一，如果正在绘制小的图元，许多PE都映射到当前图元以外的像素上。例如，一个4×4的足迹处理器光栅化100像素大小的多边形时，其中的PE映射到三角形内部像素的概率只有45%[APGA88]。第二，光栅化算法的选择影响到PE的利用率。如18.4.2节所述，只有少量条件分支的算法（如用Gouraud明暗处理来光栅化凸多边形）能够高效率地实现。有大量条件分支或使用复杂数据结构的算法（如绘制曲面图元、纹理映射、阴影处理或反混淆一系列局部遮挡的多边形）却很难取得好的效率。然而，SIMD处理器只使用一个代码存储和控制器就能满足所有的PE，所以它的体积比较紧凑，成本也较低。这在一定程度上弥补了SIMD系统PE利用率低的缺陷。目前已经提出并开发了几个交叉内存的SIMD系统，其中包括Gupta和Sproull的8×8 Display[GUPT81b]

888

和Stellar的GS2000（见18.11.3节）。

如果我们希望支持复杂的算法或消除图18-14中指出的空闲的处理器周期，我们可以给每个PE加上一个控制存储器，把SIMD变成MIMD。在MIMD系统中，PE不需要总是同时处理一个相同的4×4像素块。如果每个PE都有FIFO输入缓冲，PE甚至能处理不同的图元。分离的控制存储器、FIFO队列和PE之间同步的硬件加大了系统的体积和复杂度。交叉内存的MIMD系统的成功例子包括AT&T的Pixel Machine[MCMI87]和Silicon Graphics的POWER IRIS（见18.8.2节）。这样的系统同SIMD相比在支持更复杂图元和更复杂绘制算法方面有很强的竞争力。例如，AT&T的Pixel Machine的PXM 924型能够以交互速度对一个简单场景进行光线跟踪绘制——一个任何SIMD系统都不具备的特点。

由于交叉划分内存体系结构把帧缓存分布到多个处理器上，所以需要一些预防措施以使得像素的扫描转换能够以不间断的数据流输出，以符合视频控制器和显示器的要求。如果分布的帧缓存是用VRAM构造的，这可以用图18-18所示的方式来实现。注意，这与18.1.5节所述的在常规帧缓存中实现高速视频扫描输出的技术非常相似。

4. 进一步划分

假设我们希望构造更高性能的系统。由于在一个交叉内存系统中，进一步增加划分的数目可以提高系统的内存带宽和处理能力，我们可以考虑提高帧缓存划分的数目——如从16提高到64或甚至更高。但是，每个划分所增加的处理器和数据通路使得系统的成本不断上涨。

一个更严重的困难是，如何用同等数目的帧缓冲内存芯片支持更大数目的划分。每个划分有最少数目芯片的需求。例如，一个在PE和内存之间有32位数据通路的划分需要8个4位的芯片或者32个1位的芯片。假设我们我们希望构造一个16个分区、1024×1024像素、每像素128位的帧缓存。使用256K×4的VRAM芯片，每个分区需要8个256K×4的VRAM芯片，即需要16×8 = 128个芯片来支持所有的16个内存分区。这就是需要用来存储像素数据的芯片数量。

然而，假设我们将分区的数目从16增大到64（一个8×8的足迹）。尽管我们仍然只需要128个内存芯片来存储像素数据，但我们需要64×8 = 512个内存芯片来支持PE的内存带宽。额外的384个内存芯片仅是需要用来提供通信带宽，而不是内存。随着我们进一步分割帧缓存，这个额外分区会不断增长。

内存密度的不断增长，从1M位到4M位，进一步恶化了这一问题。例如，如果在上面所提到的例子中使用1M位×4的VRAM内存芯片，尽管每个芯片上有实际需求内存的16倍，仍然需要512个芯片。现在的系统，如使用20个帧缓存分区的Silicon Graphics的POWER IRIS GTX（将在下一节描述），已经达到了带宽极限。改善这一问题的方法是内存制造商在高密度内存芯片上提供更多的数据针脚。有些4M位的DRAM芯片有8个针脚，而不是4个，这稍微有点帮助，但仅仅是将内存带宽问题减少了一半。

18.8.2 Silicon Graphics公司的 POWER IRIS 4D/240GTX ——一个交叉划分帧缓冲内存体系结构[⊖⊖]

889～890

Silicon Graphics的POWER IRIS 4D/240GTX[AKEL88；AKEL89]使用了本章阐述的许多技术。像它的大多数竞争者（包括Ardent Titan[BORD89]、Megatek Sigma 70[MEGA89]和Stellar GS2000（见18.11.3节））一样，SGI POWER IRIS是一个高端图形工作站，它的设计结合了通

⊖ 这个例子的材料取自[AKEL88]和[AKEL89]。

⊖ 在1990年，Silicon Graphics发布了POWERVISION（与GTX类似），它能够在Gouraud明暗处理和每像素268位用于反走样和纹理映射的情况下，每秒绘制1百万个三角形。

用计算和工程与科学应用上的高速3D图形处理。

POWER IRIS有一个功能强大的通用CPU，它由4个紧密耦合、共享内存总线的多处理器组成。它的图形子系统可以每秒绘制超过100 000个全色彩、Gouraud明暗处理、z缓存的四边形[AKEL89]。POWER IRIS继续了Silicon Graphics的瞬时模式遍历、特制VLSI、前端硬件流水线和交叉划分帧缓存体系结构的传统。POWER IRIS的体系结构如图18-19所示，它包含了5个主要的子系统：

1) CPU子系统——运行应用程序和遍历显示模型。

2) 几何子系统——将图形数据变换和裁剪到屏幕坐标。

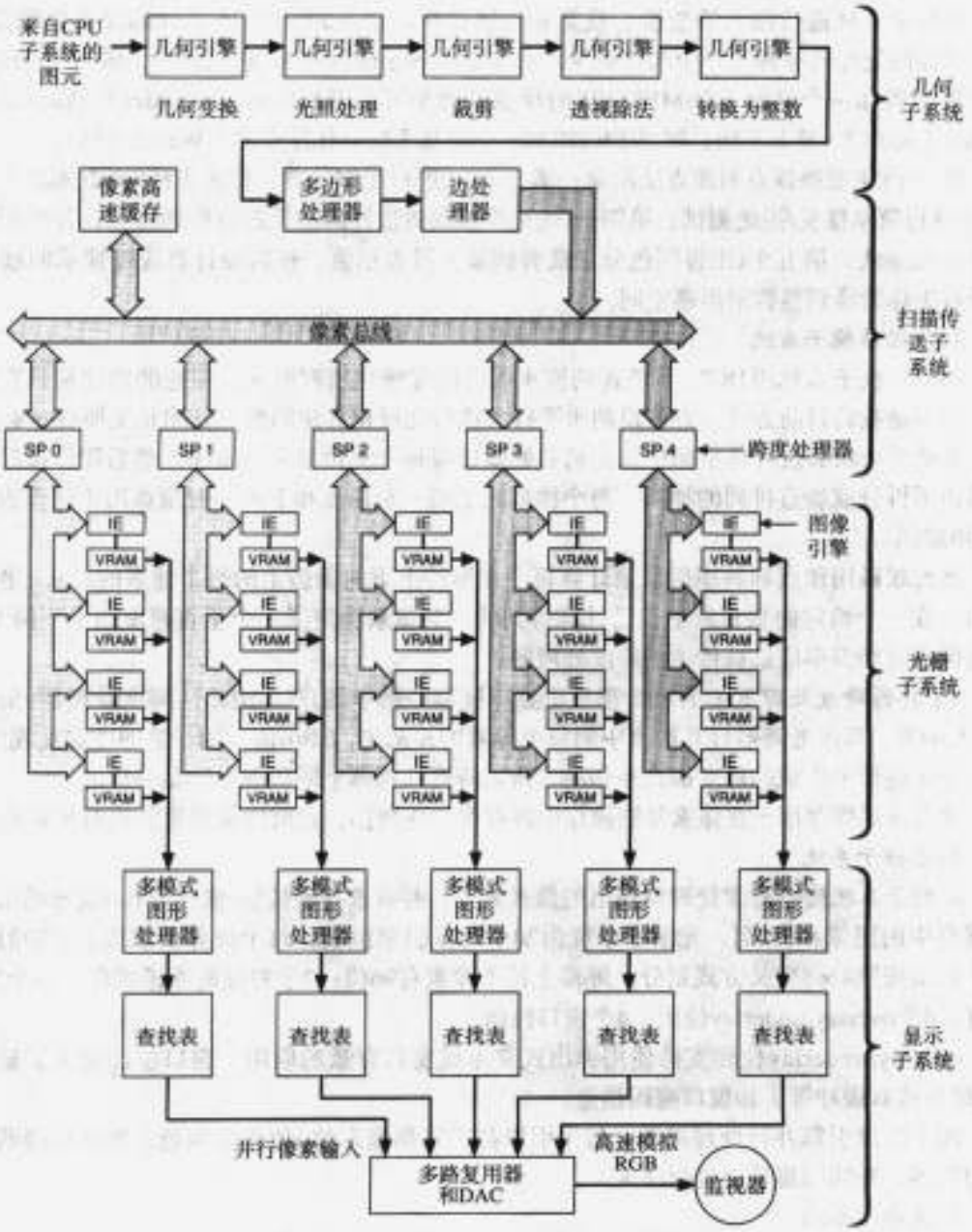


图18-19 Silicon Graphics公司的GTX系统体系结构。（基于[AKEL88]。）

3) 扫描转换子系统——将点、线和多边形转换为像素。

4) 光栅子系统——计算可见性和将像素数据写入帧缓存。

5) 显示子系统——将帧缓存的内容显示到彩色监视器上。

1. CPU子系统

CPU子系统运行应用程序和遍历数据库。它由4个紧密耦合的、对称的、共享内存多处理机组成。由于硬件提供了处理器间的高速同步，所以在一个进程内就可以获得并行性（尽管需要特殊的程序设计结构）。

2. 几何子系统

几何子系统进行图元的变换、裁剪和光照计算。它由5个排成流水线的浮点处理器组成。每个这样的处理器被称为一个几何引擎（geometry engine, GE），它包含一个输入的FIFO队列、一个控制器和一个性能达20MFLOPS的浮点处理单元。与Silicon Graphics早期的Geometry Engine（见18.5.4节）不同，POWER IRIS的GE是基于商业化浮点芯片Weitek 3332。

第一个GE变换顶点和顶点法向量；第二个GE进行光照计算（最多支持8个点光源）；第三个GE进行简单接受/拒绝测试；第四个GE对跨越裁剪边界的图元进行精确裁剪，并对所有图元进行透视除法；第五个GE将颜色分量裁剪到最大可表示值，按需要计算深度提示的颜色，并把所有坐标转换到整数的屏幕空间。

3. 扫描转换子系统

扫描转换子系统用18.7.1节所述的流水线方法光栅化所有图元，但它的跨度是垂直的像素列，而不是我们目前为止一直假设的水平行（这对光栅化算法的惟一影响是交换x和y坐标）。

多边形处理器按屏幕空间中从左到右的顺序将每个多边形顶点排序。然后用已排序的顶点将多边形拆分成垂直排列的梯形。每个梯形的上面一对顶点和下面一对顶点用于计算边处理器使用的斜率。

891 ~ 892

边处理器用顶点和斜率信息来计算每个梯形的上下两条边上的每个像素的x, y, z坐标和颜色值。在一个给定的垂直列中，已计算得到的一对像素决定了一个垂直跨度的上下两个端点。这些像素对和斜率信息被传送到跨度处理器。

5个并行跨度处理器中的每个都负责显示屏幕一列中的1/5。例如，跨度处理器0负责扫描线0,5,10等。跨度处理器计算跨度中的每个像素的z, R, G, B和α值（用于透明度和反混淆）。因为一个多边形中生成的跨度都是相邻的，所以跨度处理器上的负载大致是一样的。

像素高速缓存用于在像素复制操作中缓存像素块数据，这就使得像素总线有足够的带宽。

4. 光栅子系统

光栅子系统接收跨度处理器输出的像素数据，并有选择地按照z值比较和α混合的结果刷新帧缓存中的图像和z位面。光栅子系统由20个图像引擎组成，每个图像引擎负责1/20的屏幕像素，像素按照4 × 5交叉方式划分。屏幕上每个像素有96位：2个32位的图像缓存，一个24位的z缓存，4个overlay/underlay位面，4个窗口位面。

overlay/underlay位面支持使用弹出式菜单或窗口背景的应用。窗口位面定义了显示模式（单缓冲或双缓冲等）和窗口掩码信息。

20个图像引擎并行处理像素，每个引擎都能根据像素的α值混合颜色，能显示透明和半透明的物体，并能用超采样进行反走样。

5. 显示子系统

显示子系统包含5个多模式图形处理器(MGP)，每个负责显示列的1/5。MGP从帧缓存中并

行读出图像数据（和窗口显示模式位），并用相应的显示模式（RGB或伪彩色）进行处理，将它们发送到数模转换器进行显示。

GTX的体系结构是我们目前为止讨论的许多技术的一个很好的应用例子。它以硬件上的合理花费提供高性能的多边形绘制能力。然而，由于GTX的绘制流水线是为图形任务而特别设计的，系统很难应用于我们将要在18.11节中讨论的高级绘制技术，而且它的资源也不能容易地应用于非图形的任务。18.11.3节将讨论与GTX 有着互补优缺点的Stellar的GS2000的体系结构。

18.8.3 逻辑增强的内存

由于商业化内存可能不支持足够的帧缓存分区，一个可能的考虑是制作一个允许大量并行访问芯片上不同分区的特殊内存。由于不同（芯片内）分区必须有自己同（外部）处理器之间的连接，就必须在内存块上添加额外的针脚来支持这些额外的I/O需求。另一个选择是，多处 893
理器集成到内存芯片上。第一种可能（在内存芯片上添加针脚）直接增加了内存带宽，但是却增加了芯片的封装和与它相关的电路板的体积、成本和耗电量。这些封装上的影响随着内存密度的增长变得越来越严重。（在过去20年，典型RAM的位数已经增长了1000倍，而芯片的体积和针脚却几乎没有改变。）

本节中，我们将集中讨论第二个选择，也就是将处理能力加到多分区的内存芯片上。简单的方案仅提供新的地址模式，即并行访问一整块矩形内存像素的能力。另一个极端是把整个微处理器（包括指令存储）提供给每个内部分区。

在我们讨论特定的逻辑增强内存方法以前，让我们来考虑一下任何逻辑增强内存方案的优缺点。第一，将逻辑或处理能力加到内存中可以极大地增加一个系统的处理能力。通过增加内部内存分区数目和在同一个芯片上给每片分区提供处理能力，可以获得极大的处理器/内存带宽。第二，在定制的VLSI芯片中，可以获得板级系统所不能获得的能力，因为VLSI技术对门、线通道和内存等有着完全不同的成本限制。第三，可以降低芯片外带宽的要求，因为片外的通信仅被用来控制处理器和将像素扫描输出。这减少了芯片封装的针脚，也就意味着更小的体积和更少的板上空间。

增强内存方法的主要缺点是降低了内存密度和增加了成本。由于有很高的产量，商业化DRAM生产商能够负担得起开发特殊的、高密度的生产设备和投入大量开发费用来优化他们的设计。然而，开发和生产特殊内存芯片的投入就有很大的限制，因此其密度也就比商业化的RAM的低。每片的价格也较高，因为设计一个特殊VLSI芯片的费用没能被大的产量所抵消。尽管有这些缺点，至少有一种为图形特殊设计的内存已经取得了商业上的成功——VRAM。尽管其他的专为图形设计的内存看起来很有市场吸引力，我们还是一直在市场上看到它，这也再次证明了大规模商业发展的正确性。

1. Pixel-Planes

一个早期的、非常通用的逻辑增强内存设计是Pixel-Planes[FUCH81]。Pixel-Planes把帧缓存划分推到了极限：它为显示器中的每个像素提供了分别的处理器。每个SIMD像素处理器是一个一位的有少量内存的处理器(ALU)。图18-20给出了一个在Pixel-Planes 4中的增强内存芯片的结构图，这是在1986年完成的一个原型系统[EYLE88]。它的设计与图18-3中所示的VRAM芯片相似，这里仅仅是用1位的ALU和与其相连的电路替换了视频移位器。每个增强的内存芯片有128个像素（矩阵中的一列），每个像素有72位的局部内存（在列中的一行）。

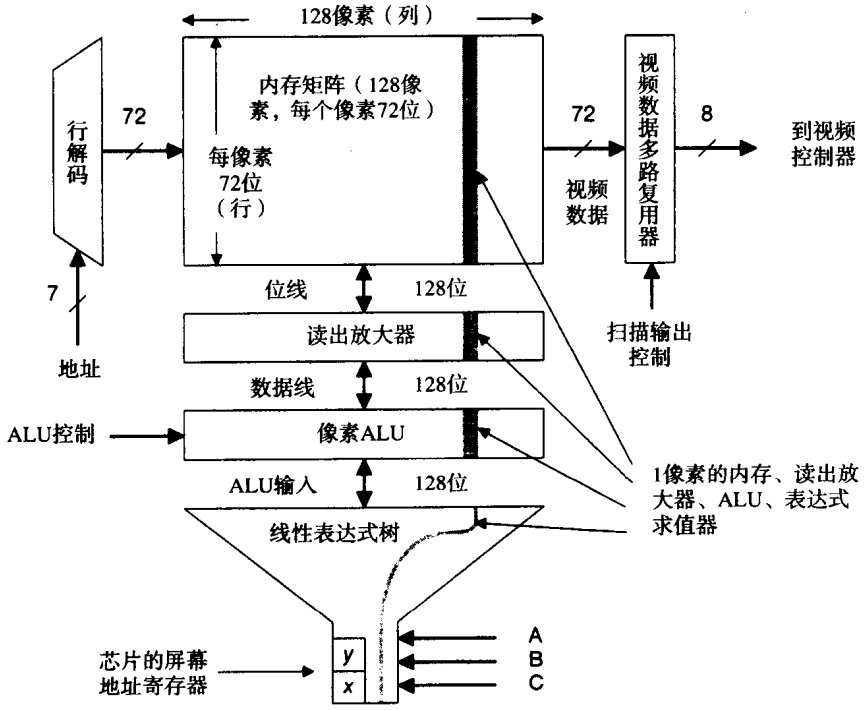


图18-20 Pixel-Planes 4的逻辑增强内存芯片

Pixel-Planes的性能并不是简单地基于大规模并行性。如果是的话，那么每个PE将不得不独立执行所有的扫描转换操作，这使得系统有大量的冗余计算，系统的效率也非常低。然而，Pixel-Planes使用了一个全局计算结构（称为线性表达式树（linear expression tree））来并行计算屏幕上每个像素(x, y)的线性表达式形式$F(x, y) = Ax + By + C$[FUCH85]。A, B, C浮点系数输入到树中，每个像素将自己的F值接收到局部内存中，每时钟周期1位（计算每个线性表达式大约需要20~30个时钟周期）。线性表达式树对加速光栅化计算特别有效，因为这些计算的大部分表示为线性表达式。例如，

- 凸多边形的每条边都可以用线性表达式表示。在边的一侧的所有点都有$F(x, y) \geq 0$，在边的另一侧的所有点有$F(x, y) \leq 0$。
- 一个三角形内的所有点(x, y)的z值可以用线性表达式表示。
- Gouraud明暗处理的三角形的像素颜色分量能被表示成线性表达式。

Pixel-Planes芯片通过提供视频数据多路复用器来实现双缓冲，视频数据多路复用器从像素数据特定的位中读取像素数据，同时计算的图像保存在其余的位中。视频数据被扫描转换到芯片上的8个视频数据针脚。

894 ~ 895

在Pixel-Planes上显示图像需要修改我们迄今为止所假定的算法。除了图元的变换和裁剪还和通常一样以外，前端子系统必须计算描述每个图元的边、z值和颜色值的系数集合。同样，因为利用线性表达式树可以一次影响大面积的像素，光栅化也是并行进行。下面几节给出了一个Pixel-Planes上算法的例子。

2. 在Pixel-Planes上光栅化一个三角形

这里，我们简要描述一下在Pixel-Planes系统上绘制Gouraud明暗处理的三角形的算法。虽然算法有些复杂，但Pixel-Planes能够绘制更通用的多边形。

扫描转换。图18-21给出了扫描转换一个三角面片的步骤。第一步是打开所有显示中的像素处理器。像前边描述的那样，用线性表达式$F(x,y) = Ax + By + C = 0$对边进行编码。然后，用线性表达式树并行计算屏幕上每个像素的每个表达式。每个像素处理器测试F的符号，来决定它是否在边的适当的一侧。如果它在边的外侧，像素处理器把它的开关位置为0来关掉自己。在多边形的所有的边被测试以后，只有那些仍然开启的像素处理器在多边形内部。这些像素处理器都独自参与了可见性和明暗处理的计算。

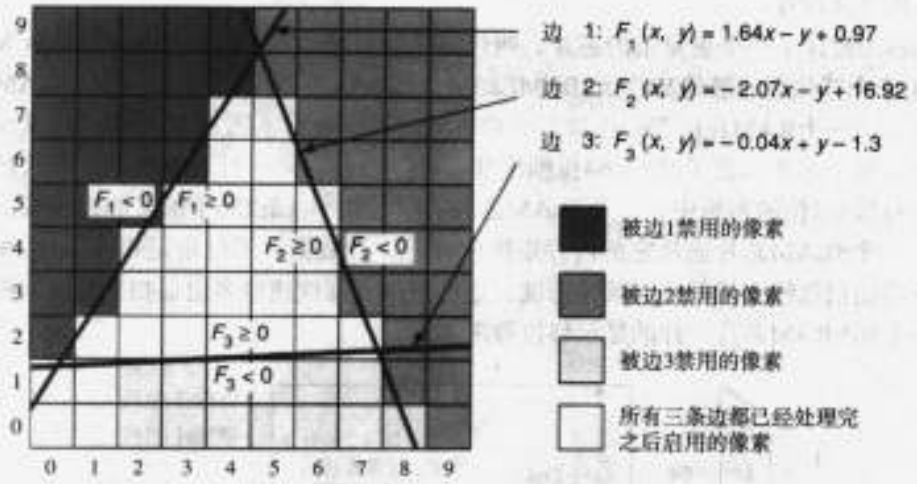


图18-21 在Pixel-Planes上光栅化一个三角面片

z缓存。在一个多边形扫描转换之后，Pixel-Planes并行计算所有像素的z线性表达式。每个像素处理器把这个新的z值同存储在它的z缓存中的数据进行比较。如果新的z值较小，当前多边形在这个像素是可见的，像素处理器刷新它的z缓存，并保持开启状态。如果新的z值较大，像素处理器关掉它自己，不再参与明暗处理计算。

*Gouraud*明暗处理。通过线性表达式树来并行计算每个像素颜色分量R,G,B的线性表达式。现在仍然开启的像素处理器将新的颜色分量写到它的像素颜色缓冲区。 896

注意，Pixel-Planes上的扫描转换与多边形的大小无关，因此一个大的多边形的光栅化同小的一样快。同样要注意，不能够表示为线性方程的操作在Pixel-Planes上执行要比那些能表示成的慢得多（例如，一个二次表达式能够在像素内存中以每次乘以一位数据进行计算）。不过，仍然有许多Pixel-Planes上的画球、阴影计算、反走样、纹理映射等高效率的算法被开发出来[FUCH85]。

1986年，建造了一个最大规模的Pixel-Planes 4系统，它包含有262 144个处理器，组成了512 × 512大小的像素图像。尽管它的性能（每秒40 000个任意大小的Gouraud明暗处理的三角面片）在它那个年代给人留下深刻的印象，它包含了2048个特制的内存芯片——一个商业化图形系统所不能承担的花费。这充分说明了这样一个高度并行的SIMD方法的缺点：图像处理器的利用率很低。由于所有的PE同步工作，尽管只有少量的PE仍旧计算有用的结果，它们也不能用于其他任务。当绘制大量的小多边形时，这个问题尤其突出，因为扫描转换的第一步处理关闭了几乎所有屏幕的像素处理器。

另外几种逻辑增强内存的图形体系结构也已经开发出来，这些系统在牺牲速度或通用性的前提下，节约芯片利用量。尽管下面描述的两种体系结构都不直接支持完整的3D绘制技术，它们都在其特定领域中取得了很高的性能（显示2D的矩形和生成2D中间图像），并且都展示了

逻辑增强内存方法的潜在能力。

3. 矩形区域填充内存芯片

Whelan提出通过修改典型的RAM的2D内存单元网格的行和列寻址以使得整个内存区域能被同时访问[WHEL82]。最小的和最大的行列地址指定了区域的左、右、上、下边界。一个写操作能够将一个数据值存储到区域的任何位置。这使得法向竖直的和恒定明暗处理的矩形能在很少几个时钟周期内完成光栅化——仅仅是足够指定四个地址值和常数数据的时间。

4. 扫描线寻址内存

Demetrescu设计了一个更复杂的芯片，叫作扫描线寻址内存（Scan Line Access Memory, SLAM），用于光栅化更一般的2D图元[DEME85]。同VRAM和Pixel-Planes一样，SLAM利用了这样一个事实，一个RAM在内部一次读写它内存矩阵的一整行。图18-22给出了一个SLAM芯片的结构图。每个芯片包含了256 × 64位帧缓冲内存。内存的每一行对应于一个像素扫描线上的一位。在每像素k位的系统中，一个SLAM芯片能够存储64/k条256个像素的扫描线。在每个内存周期，一个SLAM芯片能从它的内存矩阵中读写一行数据。通过指定相应的x_{min}和x_{max}值，可以寻址在当前扫描线上的任何连续的跨度，这样可以实现快速地多边形扫描转换。视频扫描输出使用一个同VRAM芯片一样的显示移位器来实现。

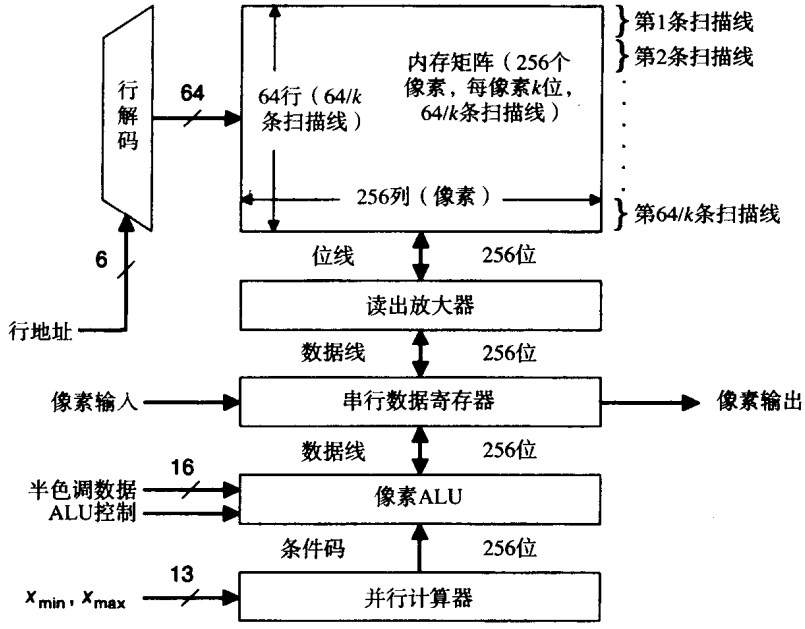


图18-22 （每像素k位的系统的）SLAM芯片的框图

在一个时钟周期内，可以指定或者一个行地址、x_{min}值、x_{max}值，或者16位的重复数据模式（指定过渡色模式）。在进行这些命令同时，SLAM可以将当前扫描线上的段写入它的内存矩阵，并且可选择地将行地址加1。对大多数图元，由于行地址和半色调模式仅需要指定一次（对起始的扫描线），后续的扫描线就能快速处理。因此，SLAM能在$2n+2$个周期内完成一个覆盖了n条扫描线的凸多边形的扫描转换（4个周期来指定起始扫描线的行地址、x_{min}、x_{max}和半色调模式，两个周期来指定其余$n-1$条扫描线的x_{min}、x_{max}）。

一个SLAM系统由一些SLAM芯片组成（芯片数量由显示器的维数决定）。图18-23给出了一个用于刷新512 × 512单色显示器的SLAM系统。每像素更多位面数的系统需要成比例的更多

的SLAM芯片。给SLAM加上一个Pixel-Planes样式的线性表达式树，SLAM还可以被扩展用于显示Gouraud明暗处理的多边形。然而，这种增强版本的SLAM需要大约与Pixel-Planes一样多的硬件，并同样有PE利用率低的缺点，因为任意给定的（小）图元的像素可能仅有少数几个SLAM芯片包含。

尽管Whelan的体系结构和原始的SLAM体系结构都比Pixel-Planes使用更少的硬件，但它们都不能满足绘制真实感3D图像的一般要求。Pixel-Planes和SLAM的增强版本确实提供这样的通用能力，但它们的PE利用率非常糟糕。如果能提高对小图元的PE利用率，用这种每像素一个处理器的体系结构来获取性能或许是一个有用的方法。18.10节给出了一种利用比整个屏幕尺寸小的增强内存矩阵来达到这个目的的方法。

897
↓
898

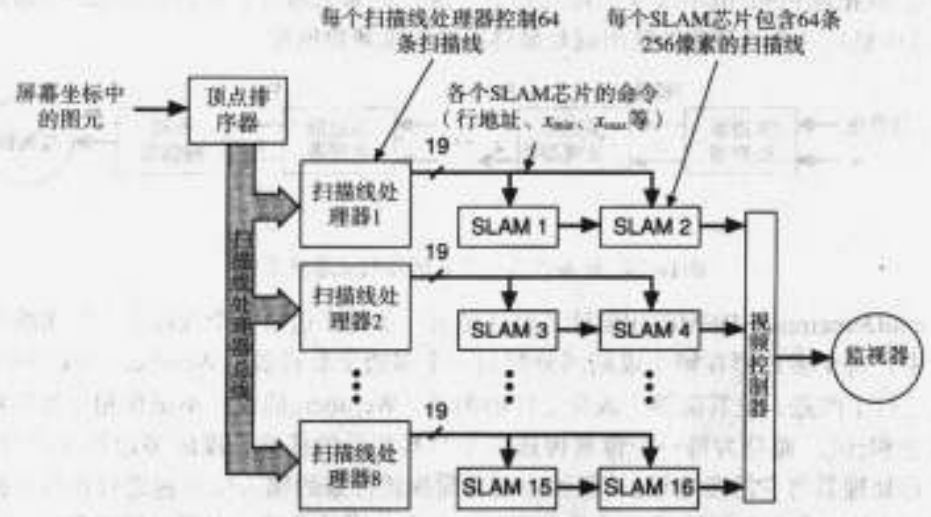


图18-23 用于刷新一个512×512单色显示器的SLAM系统

18.9 物体并行光栅化

目前为止，我们集中讨论了图像并行光栅化系统。物体并行的体系结构将图像为序算法（一般为扫描线）的内循环并行化。在物体并行体系结构中，多个图元（物体）被并行处理，使得最终的像素生成可能更快。

一种有用的物体并行化方法是把图元分配（静态或者动态）给一些相同的物体处理器(object processor)，每个物体处理器都能生成它所包含图元的整个图像。在光栅化阶段，每个物体处理器按特定的顺序（一般是扫描线）列举显示中的像素，生成其颜色、z和可能的图元局部覆盖值。将每个物体处理器的像素流组合成一个像素流来生成最后的图像。虽然可以给每个物体处理器分配任意多的图元，大多数设计都是给每个处理器分配一个图元。它的优点是每个处理器都执行一个明确定义的任务，因此能够非常便宜地复制。

通用电气的NASA II

每图元一个处理器的早期的实时系统是通用电气的NASA II飞行模拟器[BUNK89]，它于1967年交付NASA。NASA II包含了一些叫作面卡（face card）的硬件单元，每个面卡以视频速率执行一个多边形的光栅化。在任意时刻，每个面卡将处理同样的像素。

NASA II使用了深度排序可见性算法，而不是z缓存。每个面卡的输出包括一位信息来指示像素是否被它的多边形所覆盖、像素的颜色和多边形的优先级。这个信息输入到优先级多路复

用器，这样具有最高可见优先级的多边形就输出到每个像素上。由于面卡价格较贵，当它们的
899 多边形不再与当前扫描线相交时，就给它们分配新的多边形。NASA II系统能够在任意扫描线上最多显示60个多边形。

后来，每图元一个处理器的设计已经采纳了z值比较可见性算法，这使得它具有更大的灵活性。处理器一般安排成流水线或二叉树的形式，这样像素流就能合并成最终图像的像素流。

18.9.1 每图元一个处理器的流水线

一个合并多物体处理器生成的像素颜色和z值的多数据流的简单方法是将处理器安排成流水线，这样一个颜色值数据流和一个z值数据流就流过这些处理器（见图18-24）。每个处理器生成它自己的图元在当前像素的颜色和z值，并将自己的z值同输入的z值进行比较。如果输入的z值较小，就把颜色和z值不改变地传送到下一级；如果它较大，就输出自己图元的颜色和z值。流水线中最后一个处理器的输出就是最终图像的视频数据流。

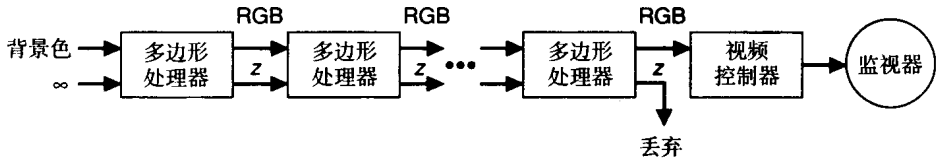


图18-24 每多边形一个处理器的流水线系统

Cohen和Demetrescu[DEME80]设计了也许是第一个每多边形一个处理器的流水线系统。在这个设计中，每个多边形在帧生成期间分配到一个多边形处理器。Weinberg[WEIN81]对这个体系结构进行了改进，使其能够生成反走样的图像。Weinberg的设计不是在相邻处理器之间传送像素颜色和z值，而是为每一个像素传送一个与其相关的任意长度的多边形片段的信息包。每个多边形处理器将它的多边形的z值和边信息同描述像素的输入信息包进行比较。把新的多边形考虑在内修改信息包的数据，并传送到下一个多边形处理器。在流水线的最后，一个过滤引擎集合从最后与像素相关的信息包中计算每个像素的颜色。

在Schlumberger的一个研究小组，提出了一个结合了NASA II，Cohen-Demetrescu和Weinberg设计的新的方案[DEER88]。他们的三角面片处理器（Triangle Processor）和法向量明暗处理器(Normal Vector Shader)使用一个三角面片处理器流水线，这个流水线传送表面法向量和多边形颜色数据，而不是实际的像素颜色。同NASA II的分配方式一样，仅当三角形在当前扫描线上是活动的，才将它分配给处理器。法向量明暗处理器使用Phong光照模型来计算从三角面片处理器流水线输出的像素颜色值。由于光照和明暗处理计算被延迟到流水线的最后，Phong光照计算仅需要对最终图像中的每个像素计算一次，而不是每个多边形的每个像素计算一次——这大大减少了计算量。

18.9.2 基于树结构的每图元一个处理器的体系结构

与流水线物体处理器不同的另一个选择是将它们并行排列，并使用一个比较器的二叉树来
900 合并颜色和z数据流。这种方法使得在物体处理器中的光栅化能够被真正地同步（在两个处理器计算同一个像素的时候没有流水线的延迟），并消除了由长流水线引起的延迟。然而，需要用一个特殊的逻辑来执行比较操作。

Fussell[FUSS82]提出了第一个二叉树结构的每图元一个处理器的设计。他注意到，由于z比较器比较简单，它们可以工作在视频速率。Shaw、Green和Schaeffer[SHAW91]提出了一种使用Fussell的方案进行图像反走样的方法。在他们的方法中，使用一个特制的VLSI合成器在图像合

成树的每个节点合成图像。这些合成器实现了一个Duff的图像合成算法（见17.6节）的简单版本，提供了许多部分像素覆盖情况的合适处理方法。这种方法的一个缺点是反走样是不完美的，相邻多边形之间的颜色扩散是可能的，然而在仅使用z值来决定可见性时，这也是不会发生的。

Kedem和Ellis的光线投射机(Ray Casting Machine)[KEDE84]直接从构造实体几何(CSG)树表示生成光栅化图像（见第7章）。光线投射机通过把每个CSG的图元分配给一个叫作图元分类器(primitive classifier)的处理器，把每个运算分配给一个叫作分类合并器(classification combiner)的处理器，将CSG树映射到硬件上。

图像按照扫描线顺序一个像素一个像素地绘制。对每一个像素，图元分类器计算光线穿过它所代表的CSG图元的光线段。这些段向上传送到分类合并器，分类合并器使用15.10.3节描述的技术执行其上的集合操作。段按照需要被分割和合并。被更新的段在树中向上传递直到它们到达树根为止。

从分类合成树的树根输出的段集合描述了当前光线同整个CSG物体的相交情况。最近一段的近端点包含了此像素可见面的z值。可利用任何光照模型，用这个值来计算像素的颜色值。如果考虑一个像素的所有段，而不仅是最近的一个，光线投射机就能够计算CSG物体的体积或其他几何量。尽管现在由主计算机承担的明暗处理计算需要几秒钟的时间，完成于1988年的光线投射机原型能接近实时地计算32个图元的CSG物体。

18.9.3 物体并行性和图像并行性的比较

尽管物体并行体系结构比较简单，很有吸引力，它们还是比前面讨论的图像并行技术受到的关注少得多。虽然已经提出了几个物体并行的实验系统，但很少能发展成商业化产品，其原因可能有以下几个因素：

- 物体并行系统一般需要特殊的处理器。这意味着过分依靠特制的VLSI芯片，也使得系统的设计十分困难，成本也很高。另一方面，图像并行系统更多地依赖于帧缓冲内存，可以利用商业化部件（如VRAM）来构造它们。
- 物体处理器的特殊特点限制了能够绘制的图元类型和可以选择的明暗处理算法。 901
- 物体并行系统负载平衡性能不好，一般地，只要有足够的物体处理器，物体并行系统就能全速运行。但需要有特殊的措施来处理大的数据库，并且系统的性能会下降很快。

除了这些因素以外，系统设计者已经能够单独使用图像并行技术大大提高系统的性能，因此商业系统的设计者就没有必要不得不面对物体并行系统的挑战。然而，如18.8节中讨论的，图像并行可能正处在一个进展缓慢的点，在未来的系统中物体并行将对设计者越来越具有吸引力。下面几节将讨论提高系统效率和性能的另外一个有前途的方法：构造一个结合了我们目前为止所讨论的方法特点的混合并行系统。

18.10 混合并行光栅化

我们已经看到，当把图像并行和物体并行方法推到极限的时候，系统的利用率都很低。如果将物体并行和图像并行光栅化技术结合在一起，就能改善这种利用率低的情况。这样的混合并行系统通常比我们目前为止讨论的系统更复杂，但是它们会更有效率。它们也为我们提供了一个构造多并行层的高性能系统的方法。

18.10.1 虚拟缓冲区和虚拟处理器

不管是物体并行还是图像并行，高度并行化体系结构设计的一个主要缺点是每个PE的低利用率。如18.8.2节中讨论的，有许多分区的图像并行体系结构有着非常低的PE利用率。例如，

在一个非常极端的每像素一个处理器的系统中，例如Pixel-Planes 4，PE可能只有少于1%的时间在做有用的工作。同样，在物体并行体系结构，如Cohen和Demetrescu的每图元一个处理器的流水线中，PE可能只用少于1%的时间来光栅化它们的多边形。

一种获得高利用率的方法是仅用一小部分硬件，但是按照需要在屏幕上动态分配它的资源。有两种这样的技术：*虚拟缓冲区*（对图像并行系统）和*虚拟处理器*（对物体并行系统）。像传统的虚拟内存一样，两种技术都通过按需动态分配资源的方法使得系统的物理资源显得较多。

1. *虚拟缓冲区*

在虚拟缓冲区系统中[GHAR89]，屏幕被划分（概念上）成几个相同大小的区域，一个同分区一样大小的并行光栅化缓冲区一次计算一个区域的像素。由于这个缓冲区较小，可以以合理的价格使用快速、特制的处理增强内存。通常使用一个全尺寸的常规帧缓存来存储最终的图像。

一个区域可以是一条扫描线、一个水平或垂直的像素带或者是一个矩形区域。虚拟缓冲区
902 与交叉内存划分的“足迹”处理器在一个关键的方面不同：虚拟缓冲区一直处理一个屏幕区域，直到区域中的所有图元都被处理为止。而交叉内存划分的足迹处理器在图像上不断移动来光栅化每个图元，一般要返回一个区域几次。

2. *虚拟处理器*

一个虚拟处理器系统表面上很像一个扫描线虚拟缓冲系统。两种系统中，图像都是按扫描线进行计算。然而一个虚拟处理器系统使用物体并行的PE，而不是图像并行的PE。由于物体处理器的数量仅需要满足处理一条扫描线上的活动图元，物体处理器的数量可以是图元数量的一小部分。18.9节中讨论的通用电气的NASA II和Deering的Triangle Processor都是虚拟处理器系统。

3. *完整桶排序与增量桶排序的比较*

由于虚拟缓冲区/虚拟处理器系统仅访问一个区域一次，这就需要进行桶排序。区域的形状对桶排序有影响。在矩形区域的虚拟缓冲区系统中，因为大多数图元都完全在一个区域内，图元可以存储在它所出现的所有区域的排序桶中。我们称此为*完全桶排序*。在以扫描线进行区域划分的系统中，由于一个图元一般覆盖超过一条的扫描线，完整桶排序是不实际的。一般选择*增量桶排序*，即图元仅存储在它的起始扫描线的桶中。

虚拟缓冲区/虚拟处理器方法既有优点也有缺点。最重要的优点是它仅利用一部分硬件就能达到一个完全极端图像并行系统的光栅化速度。由于每个区域的像素仅写入一次，它还减少了与帧缓存的数据传送。最后，并不需要一个全屏幕大小的z缓存。

然而，虚拟缓冲区/虚拟处理器系统有两个主要的缺点。第一，在桶排序过程中，需要额外的内存来缓存场景。如果变换的图元需要物体坐标系图元一样多的存储，这大约把前端系统内存需求增加了一倍，这也给一个场景中能够显示的图元数量增加了一个硬性限制。这些缺点在一定程度上被不需要全屏幕z缓存的优点抵消了。另一个主要的缺点是增加了显示处理的延迟。尽管桶排序可以同光栅化组成流水线，对一帧场景的桶排序必须在开始这一帧的光栅化之前完成。这给系统增加了一帧延迟，在实时系统中可能是有害的。

4. Systolic Array Graphics Engine(SAGE)

现在我们研究一个使用虚拟缓冲区方法的系统（我们在18.9节讨论了几个虚拟处理器系统）。SAGE是一个扫描线虚拟缓冲区系统，它使用了一个用VSLI实现的一维的像素处理器[GHAR88]。像其他扫描线系统一样，它一次生成一条扫描线的数据。然而，SAGE使用了图像并行z缓存算法对一条扫描线上的图元进行光栅化。

除了像素处理器矩阵，SAGE还含有辅助处理器来维持活动边表和将多边形分割成跨度

（见图18-25）。一个多边形管理器维持了正在处理扫描线的活动多边形表（在每个新的扫描线，它加入下一个桶的多边形，并将不再活动的多边形删除）。它还用于计算连续扫描线区间端点 903
的Delta增量。垂直插值处理器使用Delta增量来计算每个活动多边形在当前扫描线上跨度端点。这些处理器都有内部内存，这些内存可以存储达512个活动多边形的Delta增量数据和x, z, R, G, B值，这样可以避免许多对多边形内存的访问。

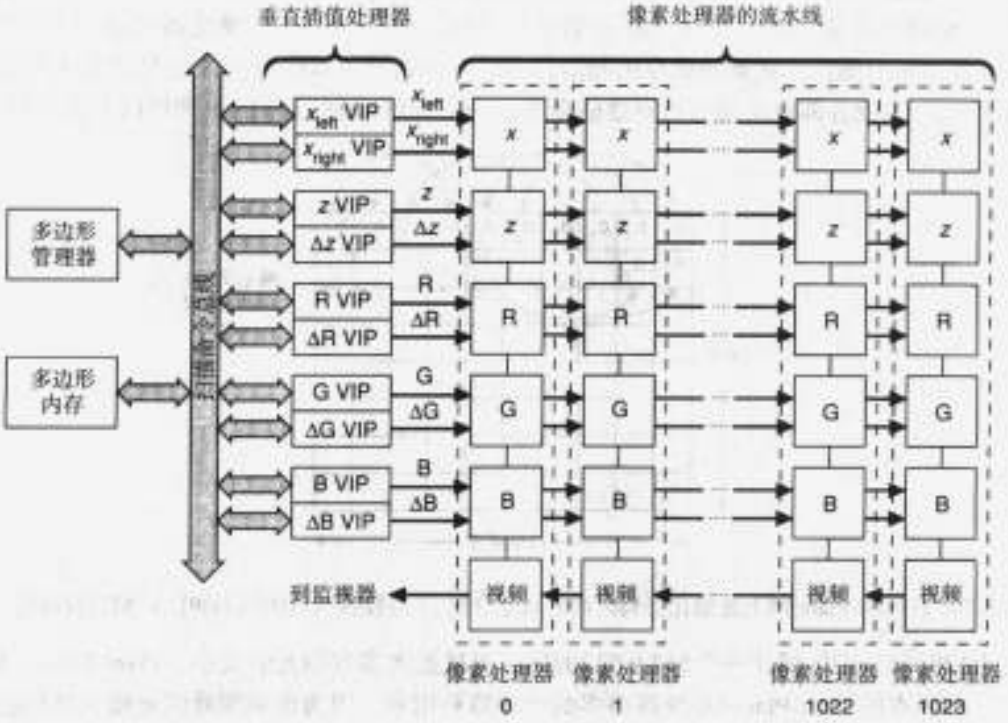


图18-25 一个完整SAGE系统的结构框图

在这些设置操作完成以后，垂直插值处理器把每个活动多边形的当前跨度端点装入到像素处理器的流水线。当一个像素处理器接收到一个跨度以后，它将其局部x值与像素的端点值进行比较。如果像素在跨度内，处理器计算它的像素的z和颜色值（再次利用区间端点的值）。如果新的z值比存储在比像素的z值小，就刷新此像素的z值和颜色值。用一个与VRAM中一样的串行移位器方案将像素值扫描输出。视频流的方向与跨度数据流的方向相反。

18.10.2 并行虚拟缓冲区体系结构

在虚拟缓冲区系统中能获得并行的另一种方法是用矩形区域和完全桶排序。由于完全桶排序允许在与前一个区域完全无关的情况下，对一个区域进行光栅化（不像增量桶排序，需要前 904
一个区域活动多边形的信息），一个使用完全桶排序的系统能有多个光栅化缓存。这些光栅化缓存能并行工作在不同的屏幕区域，能构造更快的系统。

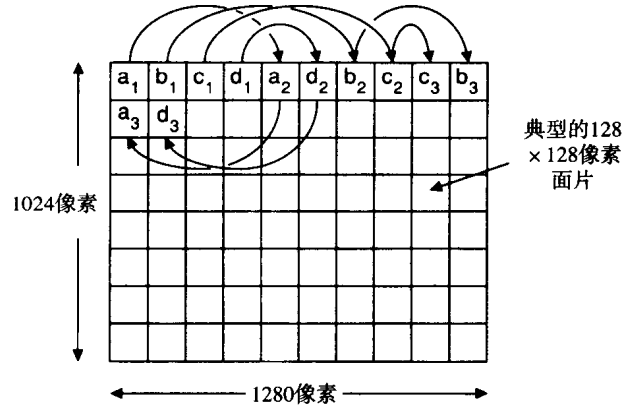
在这样的系统中，每个缓冲区开始分配一个区域。当它处理完此区域桶中的所有图元，就分配给它下一个需要处理的区域。不管单个虚拟内存缓冲区使用什么样的光栅化方法，这都提供了一个粗粒度级的图像并行方法。

1. Pixel-Planes 5

在1989年建造的Pixel-Planes 5是第一批使用并行虚拟缓冲区的系统。Pixel-Planes 5使用10

到16个逻辑增强的内存光栅化缓冲区，光栅化缓冲区的每边有128个像素。每个这样的缓冲区是一个缩小的Pixel-Planes 4矩阵，它能够光栅化落入128 × 128区域内的所有图元。这些光栅化器能被动态分配来完成一个1280 × 1024像素大小屏幕的80个区域中的任意一个区域。

Pixel-Planes 5中的光栅化分两个阶段。第一，前端处理器（16到32个浮点处理器）将图元变换并分到屏幕区域中。分类是通过比较图元的屏幕坐标范围和128像素对齐的区域边界完成的，跨越区域边界的图元被放到它影响的所有区域的桶中（100像素大小的三角形的发生概率是20%）。桶排序完成以后，多个光栅化器并行处理区域。当一个光栅化器完成一个区域后，它把新计算出来的图像传送到帧缓存的相应部分，并开始处理其他没有被分配的区域中的图元（见图18-26）。系统各部分之间的所有通信都是在一个每秒1.28G字节的令牌环网上实现的。



905 图18-26 在Pixel-Planes 5上光栅化器到区域的动态分配。（经授权引自[FUCH89], ACM版权所有。）

由于128 × 128的区域比一个512 × 512的区域更接近大多数图元的大小，Pixel-Planes 5获得了比上一个版本的Pixel-Planes系统高得多的处理器利用率。因为虚拟缓冲区光栅化器能被动态地分配到屏幕区域，系统资源就能集中到那些更需要它们的区域上。

并行虚拟缓冲系统也遇到两个困难。第一，将排序桶并行传输到多个光栅化缓冲区，需要一个前端和光栅化子系统之间的高性能数据网络，以及高级的控制和同步软件[ELLS89]。第二，在一些图像中，数据库中的大多数图元可能落入一个区域内，使得这一层额外的并行没有什么用处。在十分拥挤的区域中的图元应该被分配到更多的光栅化器上，但是就需要合成多个局部图像。尽管这样使得光栅化过程复杂化，它还是能够通过在光栅化的最后将多个图像写入一个缓冲区来实现。

18.10.3 图像合成体系结构

在光栅化之后，合成图像的概念可以用来构造第二种类型的多级并行体系结构，图像合成或合成体系结构[MOLN88；SHAW88]。其中心思想是将图元分布到几个完整绘制系统上。使得多个绘制器能够同步，以使它们能使用相同的变换矩阵，并在同一时间计算同一帧。每个绘制器独立计算它的局部图像，并在它自己的帧缓存中存储中间图像。

基本上按正常方法从每个帧缓存中视频扫描输出，同时扫描输出z缓存的内容。每个帧缓存的扫描输出的过程是同步的，这样每个帧缓存在同一时间扫描输出相同的像素。一个树结构或流水线的合成器使用18.9.2节阐述的技术将每个绘制器的RGB和z数据流合成。图18-27给出了一个每秒显示$8n$个三角形的合成系统，这些三角形由8个每秒绘制n个三角形的绘制器绘制。

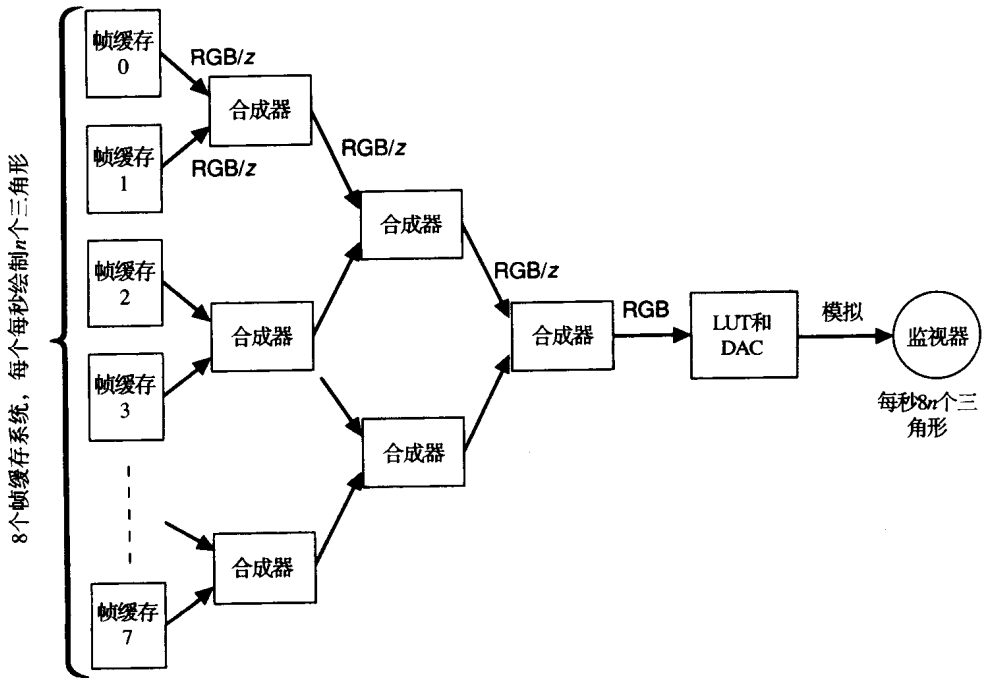


图18-27 一个由8个独立绘制器构成的图像合成系统

通过使用大量的并行绘制器，这个技术能够用来建造任意高性能的系统。这种方法的主要困难有：需要将数据库分布到多个处理器上，这导致了并行前端系统的困难（见18.6.1节的讨论）；由图像合成操作引起的走样或错误像素；图像合成限制了能在机器上实现的光栅化算法的种类，因而缺乏灵活性。然而，这种方法提供了一个实现极高性能系统的重要途径。

18.11 增强的显示能力

本节讨论对标准Gouraud明暗处理多边形绘制算法的各种改进对体系结构的影响。这些不断增多的特点是供应商和用户的需求。在本节的最后，我们将讨论飞行模拟器的几个问题，它们一般都是包含了大量先进特性的系统。

18.11.1 对多窗口的支持

如第10章指出的，显示多个不同应用控制的重叠窗口的能力已经成为目前工作站的必须任务。如果硬件没有对这种能力进行支持，整个图形系统的速度就会受到严重影响。例如，将一个窗口推到另一个后面，弹出一个窗口以使它是可见的等操作，都要求从帧缓存的一个区域向另一区域快速复制内存像素，以及在有些情况下快速重建部分或整个图像的能力。同样，如果同时生成不同窗口中的多个图像，绘制流水线必须能在应用间快速切换。

为提供这种支持而对体系结构进行的改进经常改变整个图形子系统的设计：从显示遍历，到几何变换、裁剪、光栅化和帧缓冲内存组织。我们这里讨论一些建造一个支持多个交互应用窗口系统的主要困难。对这个问题的更完整的论述见[RHOD89]。

1. 不同应用间的上下文切换

当图形系统从运行在一个窗口的应用转换到运行在另一个窗口的应用时，必须保存图形流水线许多地方的状态信息。例如，在模型变换级，必须保存当前应用的当前变换矩阵堆栈，并装载下一个应用的变换矩阵堆栈；在视频控制器，可能必须重新赋值查找表的入口。为快速地

保存状态信息，在绘制流水线的许多地方都需要局部内存。

906 ~ 907

因为局部内存的数量是固定的，这限制了能够同时运行的应用的数量。如果不可能进行快速的上下文切换，图形系统在窗口之间的切换一般比较慢（可能每秒才几次），会导致多个窗口中的物体移动非常缓慢。

2. 非凸屏幕窗口裁剪

当窗口重叠时，像素只需要写到其相应窗口的暴露部分。这需要用非凸的窗口边界对像素进行裁剪。这个问题的可能解决方法有几个。

将整个窗口写到屏幕以外的内存。这将绘制流水线从窗口问题中分离出来。可以在一帧刷新以后，用快速像素拷贝操作将窗口的暴露部分复制到真正的帧缓存。这需要额外的像素拷贝的处理，并增加了图像生成过程的延迟，这都可能严重妨碍交互性。另一种方法是，视频系统将视频内存中的不连续部分组合起来，如在Lexidata Lex 90系统[SUYD86]和Intel 82786图形协处理芯片[SHIR86]中那样。这种方法不增加延迟，但由于将不同内存部位的视频图像组合起来需要大量内存，这限制了窗口的位置和数量。

有选择地更新屏幕像素。第二种选择是为每个窗口计算所有的像素，但是仅更新那些在窗口暴露的部分像素。这同第3章中描述的裁剪比较类似。为使操作能够进行得快，一般需要硬件的支持。一种流行的方法是使用窗口ID位[AKEL88]。这样，几个额外的位（一般4到8位）同每个像素相关联。这些位指定了每个窗口的惟一ID。当帧缓存更新时，硬件把写入像素的窗口ID同存储在帧缓冲中的窗口ID进行比较。仅当这两个窗口ID相同时帧缓存像素才被更新。

在绘制流水线中进行裁剪。上面两种方案的最主要缺点是它们都生成了许多不被显示的像素。另外一种方法是在绘制流水线中裁剪像素，以便仅仅计算那些真正被显示的像素。这种方法的主要困难是窗口的暴露部分可能不是一个简单的矩形：它可能因为被高优先级的窗口局部覆盖而是凹的，也可能因为包含一个小的高优先级窗口而有一个洞。对这样任意边界的裁剪是非常耗时的。一种解决方法是将暴露区域划分成一系列小矩形窗口，并用简单的窗口边界来裁剪图元。当然，由于图元必须被处理多次，每个矩形区域一次，这对系统性能影响也很大。

一般地，为所有窗口配置和所需动作提供全部的性能是非常困难的。系统设计者一般从一个窗口性能需求列表开始，并努力寻找在不引起过分系统代价的实现方法。大多数系统仅在一定的简单情况下才全速运行，反之系统性能下降很大。例如，一个系统仅当绘制窗口完全暴露（或者大部分暴露，或者它的可见部分是个矩形）时才能全速运行，或者当多于一个的窗口被同时刷新时，就会大大影响它的绘制速度。

908

18.11.2 对增加的真实感的支持

目前为止，我们讨论的图形体系结构主要与速度问题相关。随着过去几十年可接受系统速度的持续发展，用户对真实感图像的需求也越来越高。第14、15、16和20章讨论了生成真实感图像的算法。我们在这里将讨论真实感绘制对图形系统体系结构的影响。

1. 反走样

如第4章讨论的那样，反走样的两种主要方法是区域采样和点采样。在区域采样中，我们需要知道每个图元对每个像素的贡献部分（理想地，我们期望使用合适的过滤器计算出贡献的权值）。不幸的是，对3D图像精确计算它的贡献部分是十分困难的。对反走样提供硬件支持的体系结构一般采用超采样的方法。根据光栅化处理器的特点，可以采用均匀超采样和自适应超采样。

均匀超采样：均匀超采样以高分辨率计算整个图像，并合成每个像素的多个采样点来决定像素的颜色值。这种方法看起来仅对SIMD的图像并行系统是可行的。但是，以每像素n个采样

的超采样将系统的性能降低为1/n。因为这个原因，这种类型的一些系统采用了连续细化的方法进行反走样，系统首先快速以需要的刷新速率绘制一个粗糙的图像，当图像没有变化时，系统进一步计算采样点，并用它们来优化图像。

自适应超采样：自适应超采样仅在需要的地方以高分辨率计算图像。这种方法在以图像为序的光栅化系统中执行得更快，因为在这些系统中可以得到哪些像素是被部分覆盖的信息。在以物体为序的光栅化系统中使用的最著名的自适应超采样是A缓存（见15.7.3节）。用链表来存储部分被覆盖的图元对那些每个像素有任意多内存的MIMD系统才是可行的。

2. *透明性*

第16章提到，真正的透明性，包括折射和半透明，在软件中很难建模。甚至不包括折射和半透明的简单模型也很难在许多硬件上实现。图像并行的体系结构在透明性处理方面存在特殊的困难，因为z缓存算法不能够很好地处理多个面（需要额外的z值来处理每个半透明的面）。使用SIMD PE的图像并行系统或每像素有固定量存储器的图像处理系统，一般可以通过多遍的光栅化来显示几个透明的物体——一遍处理所有的不透明图元，然后另一遍处理每个透明面[MAMM89]。如果需要更多的透明面，一般必须使用screen-door透明方法。MIMD的系统能用一个可见面的链表来表示像素，使其可以显示大量的透明物体。

由于物体并行体系结构不需要整个图像的中间存储，并且直到计算每个像素颜色时，还保 909
存着可能可见的多边形，它能更好地处理透明效果。然而，光栅化器必须能够处理多个覆盖的多边形，并能执行最终像素值的加权平均计算。

3. *纹理*

直到最近，能够以交互速率进行纹理映射的惟一系统是数百万美元的飞行模拟器。然而，越来越多的不同形式的纹理映射已经在工作站甚至是低端系统上实现。当然，纹理映射能在任何体系结构上使用无数个在顶点之间进行明暗处理插值的三角面片来模拟。但是这种方法对显示系统处理能力要求很高，只在高性能的工作站上或使用简单纹理的情况下才是可行的。

两种传统的纹理映射的方法是纹理图和过程纹理（见第16章）。这两种方法的缺点是纹理映射是在绘制流水线的后端实现的——在图像已经被转化为像素以后。在许多体系结构中，这时像素数据已经不能由通用处理器访问。飞行模拟器增加了额外的硬件阶段来进行纹理图查找。几种图形工作站（如Stellar GS2000（见18.11.3节））把生成的图像存储在能被系统的通用处理访问的主存中。

在能访问生成图像的粗粒度的MIMD处理器体系结构中，纹理图方法是最合适的方法。在每个像素可以存储纹理坐标，而不是颜色。如果为每个MIMD处理器提供一个纹理图备份，它就能够自己查找纹理图，将纹理坐标转换成相应的颜色。注意，简单的纹理图查找在这里一般是不够的——需要使用一些多分辨率的纹理图和求和区域表来避免走样。

在一些能访问计算图像的细粒度的并行处理器体系结构中，过程纹理更合适。细粒度的处理器可能没有足够内存来存储纹理图。然而，如果有许多PE，它们也能够以交互速率计算简单的程序纹理模型。

4. *阴影*

如16.4节所述，计算真正的阴影需要花费高昂的计算代价；在许多体系结构上，根本就不能计算真正的阴影。尽管有许多技术可以在一定条件下估算阴影，因为这些原因，没有几个高性能系统实现真正的阴影[FUCH85]。如16.4节所述，在商业化飞行模拟器和几个图形工作站系统中使用的通用技术是仅在地面上生成阴影。

5. 光线跟踪体系结构

第15和16章描述的光线跟踪是一个生成高度真实感图像的非常强有力的绘制方法。但是，它需要大量的计算（在典型工作站绘制一个典型的图像需要几分钟或几小时）。幸运的是，光线跟踪算法有多种并行化方法，其中许多方法同常规的方法十分相似。

910

- 分量并行。一条光线的计算可以并行化。例如，反射、折射和求交计算都需要计算向量或点的x,y,z分量。这三个分量能并行计算，能有3倍的加速比。
- 图像并行。由于每条光线的计算是独立的，光线和图元的求交能在不同的PE上计算。然而，利用这一点来实现并行，PE需要访问整个数据库，因为一条光线的光线树可能达到数据库的任何部分。
- 物体并行。数据库中的图元能按空间分布到多个PE上，那么每个PE负责计算穿过它的区域的所有光线。它计算光线与物体的相交，判断光线是否同物体相交，否则将光线传送到下一个PE。

许多光线跟踪体系结构的设计已经使用了这些技术中的一种或多种。由于光线跟踪一般需要大量的分支计算和随机寻址。大部分系统使用大量的MIMD PE。

最简单类型的图像并行体系结构给每个PE分配一条或多条光线，并将数据库复制到各个PE上。1983年在Osaka大学建造的LINKS-1使用了这个技术[NISH83a]。LINKS-1已经用于绘制大量的动画帧。这个技术还用在最近在Nippon Telegraph 和Telephone设计的SIGHT[NARU87]系统中。SIGHT体系结构还在它的TARAI浮点处理单元上利用了分量并行性。

第一个提出的物体并行光线跟踪体系结构使用均匀空间划分来将场景的各部分分配给PE[CLEA83]。由于大量的图元拥挤在少量的区域中，系统在许多场景下的效率很低。其他提出的体系结构采用自适应场景划分来提高PE间的负载平衡[DIPP84]。尽管这种方法提高了PE利用率，它使得将光线映射到PE变得很困难。

由于许多光线跟踪体系结构很接近于具有低成本和成熟编程环境的商业化并行计算机，并行光线跟踪的研究已经转换到开发商业化并行处理器（如Hypercubes, Transputer meshes和Connection Machine)上的高效算法上。两个主要的考虑是获得高利用率和PE间的平衡负载，特别是使用分布式数据模型。

目前已经开发了Thinking Machines的SIMD Connection Machine[DELA88]上的图像并行光线跟踪算法。它把数据库重复地发送到所有的并行执行光线-物体求交的PE上。

在共享内存的多处理机（例如BBN Butterfly）上实现的算法也有报道[JENK89]。这里数据库不需要存储在每个PE上或重复广播，而是PE按需要从内存系统中读取数据库的一部分。不幸的是，随着PE数量的增多对共享内存资源的竞争越来越激烈，因此这样的系统仅能获得中等性能水平。

Nemoto和Omachi[NEMO86]、Kobayashi和Nakamura[KOBA87]、Scherson和Caspary[SCHE88]以及其他人（见[JEVA89]）已经提出了各种各样的自适应地给PE分配物体和在PE之

911

间传递光线的方法。AT&T的Pixel Machine（一个使用基于数字信号处理芯片PE的MIMD多处理机）能以交互速率进行简单图像的光线跟踪[POTM89]。这样的系统给大家看到了并行光线跟踪器在将来可能取得的性能的前景。

18.11.3 Stellar GS2000——促进真实感绘制的紧密集成的体系结构

像Silicon Graphics的POWER IRIS这样的系统使用特制的硬件来快速计算图像，其中许多计算，特别是几何变换或其他前端阶段的计算，与许多科学计算所需要的处理类型很相似。

Stellar GS2000（与它的先驱Stellar GS1000[APGA88]一样）试图使得这部分计算资源既可以用于图像生成，也可以用于机器上运行的其他大计算量的工作。

图18-28给出了GS2000体系结构的框图。多指令流处理器(Multi-Stream Processor)是一个同时执行来自四个独立的指令流中指令的高性能处理器，它的峰值处理速度是每秒25 000 000条指令。向量浮点处理器执行标量和向量浮点操作，它每秒最多能处理40 000 000次浮点操作。绘制处理器使用了一个配置的处理器来执行多边形处理计算和一个4×4 SIMD足迹处理器来执行像素操作。GS2000每秒绘制150 000个伪彩色的、Gouraud明暗处理的、z缓存的、100个像素大小的三角形和30 000个Phong明暗处理的多边形。

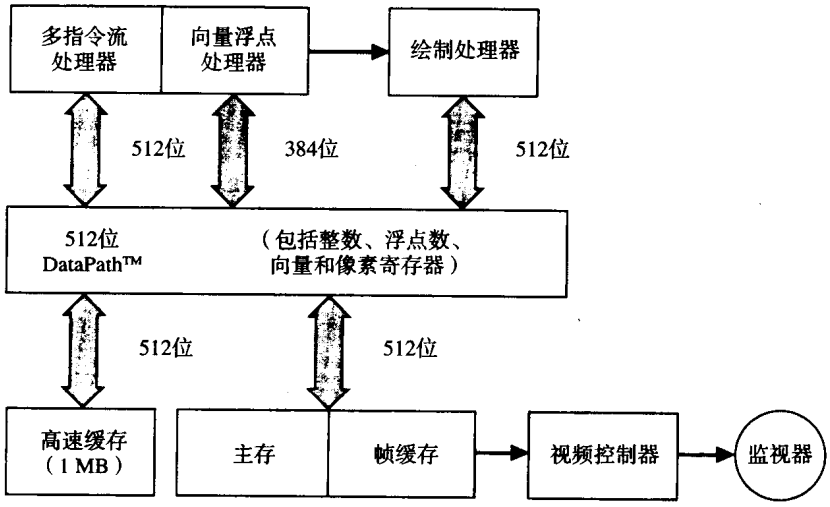


图18-28 Stellar GS2000的结构框图。（基于[APGA88]。）

GS2000的所有处理和内存资源被组织在一个512位宽的称为DataPath的中央通信结构上。DataPath设计独特，它数据通路的每一位的所有I/O连接和寄存器都建造在特制的门阵列芯片上（每个DataPath芯片包含16位电路，因此一共需要32个）。 912

在典型的图形应用中，多数据流处理器遍历存储在主存中的结构数据库。它把数据输入到执行几何计算的向量处理器中。向量处理器把变换的图元发送到绘制处理器。绘制处理器计算图像像素。它不是把图像直接存储到帧缓存中，而是把它们作为*虚拟像素图*存储在主存中，其中的可见部分被复制到帧缓存的相应部分用于显示。

因为虚拟像素图存储在主存中，多数据流处理器和向量处理器都可以访问，可以执行一些后处理操作（和图像处理及其他操作）。例如，为显示纹理，光栅化单元在像素中生成纹理索引，而不是最后的颜色。最后，一个通用目的的处理器忽略中间图像，并以纹理索引查找表中合适的颜色替换。而且，重叠窗口也可以很容易地实现，因为每个屏幕窗口的内容总是能从主存中得到。虚拟像素图的主要缺点是需要额外的带宽和从主存到帧缓存复制像素数据的时间。

18.11.4 对高级图元的支持

目前为止，我们的讨论集中在快速显示多边形的体系结构上。要求系统处理其他类型图元的需求正在不断增长。许多复杂的面图元，例如样条曲面和网格图元，能转换成多边形进行处理，仅有少量时间的代价。然而，对其他类型的图元，这样的转换或者是十分耗时的或者是很困难的。例如，将一个CSG物体转换成边界表示是十分耗时的，并能导致数据量的急剧增长。多边形化一个体数据集也是很慢的，并可能掩盖数据，并产生不必要的人工痕迹。如果能直接

绘制更复杂的图元，就可以避免这些困难。在许多情况下，系统性能还能提高。

1. 曲面

如第11章中讨论的，多边形是比较简单、规则的、容易绘制的图元（特别是用硬件）。但是，由于它们非常低级，它们不适合于造型，并导致不准确性，特别是对复杂的曲面。曲面的其他表示，例如Bezier曲面和NURBS，更容易在造型中指定，但是直接绘制需要复杂的处理。

大多数硬件系统在显示高级图元时，出于绘制原因将图元分解成多边形。对许多类型的曲面图元（特别是Bezier曲面和NURBS），这种分解可以在硬件中使用向前差分引擎快速地进行[LIEN87]。而且，在某些情况，它可能比直接绘制曲面要快，特别是在使用可编程并行光栅化处理器的系统中。在本书写作的时候，没有几个系统提供这样的能力[MCLE88]。未来的系统
913 是继续将曲面划分为多边形，还是直接绘制这些曲面，还在继续争论。

2. 体数据

体绘制——直接绘制由3D标量场绘制表示的数据（见20.6节的讨论）——正在成为计算机图形学的一个重要分支。在许多方面，它都比多边形绘制更需要硬件加速，因为体数据集通常比多边形数据库大得多（一个典型的CT采样数据集可能有$64 \times 256 \times 256 = 4\ 200\ 000$个体素，却很少有多边形数据库超过1 000 000个多边形）。而且，体素计算比多边形计算要简单。

被设计来加速体数据绘制的第一个体系结构把体素分为“空”或“非空”，并显示非空体素。这种方法降低了处理需求，但遮挡了物体内部的数据，并且在生成图像中产生“方糖”效果。Phoenix Data System的Insight系统[MEAG85]使用这种方法：它能交互式绘制一个八叉树编码的体数据集。Kaufman的Cube体系结构[KAUF88b]提供了能并行访问一个像素所对应的整行体素的特殊硬件，而且这个硬件能在对数时间内决定一行中最近可见的体素。从任意角度观察是通过在Cube的3D内存中旋转数据集来实现的。

最近的体绘制方法给每个体素指定部分透明度（或不透明度），而不是使用二分法，并试图消除采样带来的人工痕迹（见20.6节）。因为同二分法相比，有更多的体素对每个像素有贡献，这些算法大大增加了图像生成的时间。

基于这些算法的体绘制体系结构一般使用我们前面看到的图像并行和物体并行方法的变种。图像并行体系结构为PE分配像素或像素组[LEVO89]。物体并行体系结构通常将体素划分成连续的区域，并分配给每个分别的PE[WEST89]。在绘制期间，PE计算它们的体素对图像中的像素的贡献，就像它们的体素是仅有的一个数据库。最后使用这些颜色和不透明度的集合来合成计算多个区域的计算。

物体并行主要的优点是它在PE间分割数据库，缺点是一个PE计算的像素值可能被另一个其体素离视点更近的PE计算的像素遮挡。图像并行有互补的优缺点：每个PE仅处理可能可见的体素，但是每个PE需要访问整个（非常大的）数据库。

Dynamic Digital Display的并行Voxel Processor系统是一个物体并行体系结构的例子[GOLD88]。一个超立方体体系结构通过把每个图像片上（或整个数据库中）的每个体素分配给分别的PE，已经应用到物体并行方法中。Levoy提出了一个使用Pixel-Planes 5的混合系统，此系统中体素光照计算是按物体并行方式来进行的，图像生成是按图像并行方式来进行的[LEVO89]。Pixar Image Computer以并行方式分别计算一个图像的红、绿、蓝分量[LEVI84]。因为体绘制是一个相对较新的领域，很难预测哪种体系结构（或甚至算法）会成为将来的主流。

3. 构造性实体几何(CSG)体系结构

914 像我们在第12章看到的，CSG是一种更流行的实体造型技术。第15章给出了直接从物体的

二叉树表示中绘制一个CSG物体的方法。直接绘制CSG的两种流行方法推广了本章以前描述的图像并行和物体并行光栅化技术。

以图像为序CSG像素化。这种方法在18.9.2节中给出。Kedem和Ellis的Ray Casting Machine是这个算法的一个硬件实现。他们现在的系统能以交互速率显示小的CSG物体。

以物体为序CSG像素化。第15章也描述了物体为序（或深度缓存）的CSG像素化算法，它推广了z缓存算法。CSG深度缓存算法能在任何有足够内存来为每个像素存储两个z缓存、两个颜色缓冲和三个1位标志的体系结构上实现[GOLD86]。许多现在的高性能工作站为帧缓存提供了足够的内存。如果有足够的像素级处理能力，深度缓存的CSG显示算法能以交互速率显示中等大小的物体[GOLD89]。如果每个像素有更多的内存，更复杂的CSG物体也能更快地显示[JANS87]。

18.11.5 对增强的3D感知的支持

回忆第14章中，显示在2D屏幕上的3D图形，仅有少量的3D深度提示：遮挡、动力学深度效应、光照处理和有时的阴影。真实的3D世界提供了更丰富的立体信息，如立体观测和头部运动视差等。本节讨论试图增加这些额外的立体信息的体系结构及其改进方法。

1. *立体显示*

计算左右眼的分离图像，并把每个图像投射到对应的眼上，也可以在2D显示上获得立体视觉。图像对可以在分离的显示器上显示（或同一个显示器的不同窗口），或者左右眼图像可以交替的帧中被显示。在前一种情况，可以用一个立体阅读器来观察图像对，立体阅读器用光学通道将两个图像投射到屏幕前面观察者所在的位置。这种方案的缺点是同一时间只能有一个人观察场景，并且仅能利用屏幕的1/2的分辨率（除非使用两个显示器，这样系统的成本就急剧增长）。

多路复用显示左右图像是更流行的技术。这种技术需要快速连续地交替显示左右眼图像，并在一个眼的图像被显示的时候，遮挡住另一个眼的视线。一个显示立图像的图形系统一般需要一个足够存储四个完整图像的帧缓存——足够提供每个眼图像的双缓冲区。而且，一些系统需要在适当的时候遮挡一个眼的视线。一种方法是使用一个与帧缓存同步的机械百叶窗以使得你能在合适的时间看到左右图像[LIPS79]。但是，机械百叶窗比较重，噪音也比较大，而且在有些情况下有危险（一种设计是使用快速旋转的离眼不到1英寸的圆柱）。

一个更流行的机制是电子百叶窗，它可以使得一个方向的偏振光通过。电子百叶窗可能与显示屏幕一样大，并安装在屏幕上，或者更小，可以戴在特殊的眼镜上。无论哪种情况，每个眼前都放置一个方向相反的偏振透镜。当电子百叶窗的偏振方向与两个透镜的偏振方向一致时，屏幕是可见的。当偏振方向相反时，视线被阻挡。把百叶窗放置在显示屏幕前面可以让几个用户同时观察图像，如果每个用户有一个便宜的、被动式的眼镜。然而，大的电子百叶窗十分昂贵。使用电子百叶窗的立体显示也比机械式的偏暗，因为偏振胶片只透过显示器的一部分光线。

915

2. *变焦镜*

*变焦镜*是一种不常用的设备，它使用振荡镜来显示真3D图像。这些图像不需要用户佩带特殊头盔就能提供立体观察和头部运动视差。基本的思想是放置一个焦点可快速改变的柔软的镜子，使它能够将显示的图像反射给用户（图18-29）[TRAU67；FUCH82；JOHN82]。当这个镜子与普通的扩音器一起振动，镜子的焦距就以正旋曲线变化。一般是以30 Hz进行振荡。

镜子周期地改变它的焦距，使得镜子的距离显得以30 Hz的频率不断增加或减少几英寸，或者更多。点、线及相似数据在点式绘图（不是光栅扫描）显示器上进行显示。一个点的感觉距离是由它在显示列表的位置决定的："近"的点被存储在刷新列表的开始和最后；"远"的点被存储在刷新列表的中间。注意在列表中的两个地方点看起来有相同的深度——当镜子向前和当向后移动时。

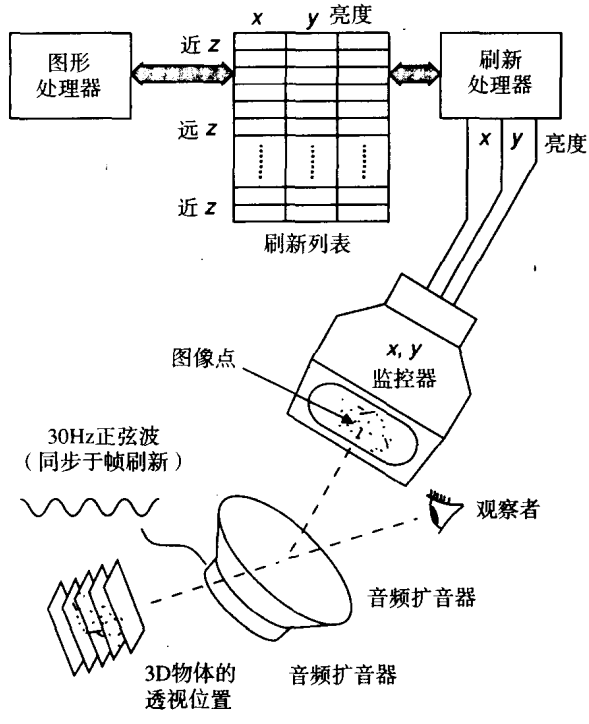


图18-29 变焦镜显示系统

变焦镜显示有几个限制。一个缺点是较近的物体不能遮挡较远的物体，因此只能显示不遮挡的图元，如点、线、透明的体数据等。第二个困难是在一个时间显示的数据量有一定限制——在一个镜面振动周期内能够刷新的数据。除了这些限制，变焦镜的商业化产品很少，Genisco SpaceGraph是其中一个[STOV82]。

3. 头盔显示器

在1965年国际信息处理联合会大会的演讲中，Ivan Sutherland提出终极显示器（ultimate display），一个能生成图像，并有传感器输入的设备，它能使得观察者不能区分模拟和真实的物体。在1968年，他展示了一个戴在头上的原型显示器，并示范了终极显示器的重要性质——它允许用户在虚拟世界中漫游。特别地，包括如下设备：

1) 有两个小显示设备的头戴设备，每个显示器被光学系统投射到一个眼上。

2) 一个跟踪系统使得计算机在所有时间都知道用户头盔（也就是头部）的精确位置。

3) 有一个手持的手杖，它的位置也被系统跟踪，使得用户可以在虚拟环境下抓取和移动物体。

4) 一个实时的图形显示系统，能持续不断地根据用户的移动在显示设备上生成图像，给用户一种围绕着房间中的虚拟物体漫游的错觉。

这种系统，具有丰富的头部运动视差和立体观察等3D立体信息，并能简单直接地在3D虚拟世界中操纵物体。它很有说服力地向许多用户证实传统的使用桌面CRT的3D场景交互方法是不令人满意的，这种受限制的机制把用户排斥在3D虚拟世界的CRT窗口以外。但是，几个技术问题使得头盔显示器还不能达到它的全部效率——实际上，它们还不能达到Sutherland的1968年原型的水平[FISH86; CHUN89]。这些技术问题包括：

1) 开发可佩带在头上的高分辨率显示器，其图形显示器屏幕的视见场景宽度应该能达到真实世界的水平（一个困难的光学问题）。

2) 开发头盔和手的跟踪系统，其分辨率在一个房间范围内应能达到1毫米，且响应时间是几毫秒或更少。

3) 设计一个具有很小延迟的每秒至少生成30帧图像的图形系统（低延迟在这里特别重要，因为在头盔显示系统中的延迟能用户头晕[DEYO89]）。 916 ~ 917

现在能够看到的头盔显示器的大量应用是军用飞机驾驶仓中的头盔显示器。然而，头盔显示器仅给用户以辅助信息，而不是创建一个虚拟世界和完成一个能被直接操作的物体。在1989年，Autodesk和VPL Research两个公司发布了基于现有技术的商业化头盔显示系统。这样系统的支持者预测这类系统将是21世纪中与Sony Walkman个人立体声录音机相等价的图形显示设备，它不仅被用于实时3D应用，还将被用于通用便携式交互计算。

4. 数字全息术[⊖]

全息术是不使用特殊头戴式或观察者位置跟踪的显示真3D图像的另一种方法。传统的全息术是把照相胶片同时暴露在从被记录物体散射的激光线和从同一激光源的参考光线束中生成的。记录在胶片上的干涉图编码了从一个区域的视点看到的物体的外表。用反方向的激光照亮胶片就可以观看全息图。

物体的全息影像也可以通过在计算机上模拟激光干涉过程，并把计算结果写到高分辨率的胶片上来产生。但是，用这种技术生成的全息图需要10^{12}个傅里叶变换（每个需要大量的乘法和加法），并需要VLSI类型的电子束写入器将结果写到胶片上[DALL80；TRIC87]。因为这些原因，这种技术目前对大多数应用来说是太贵了；然而全计算的全息图包含了比人眼需要的信息要多。一个折中的方法是将计算量减少到实际需要的水平。这将是进一步研究课题。

全息立体图是通过计算机生成的一系列透视的景象来构造的，是现在很有吸引力的技术。在全息立体图中，一系列大约100个从稍微不同的视点观察的景象被用激光投射到全息胶片上，每条光线从它被计算的方向来照射。第二个从同一光源来的参考光线束与投射光线束重叠，并以干涉图来记录视图方向。在曝光和处理以后，用一个反方向的参考光线束照亮胶片（现在是全息立体图）。图像光线束就散射回它们的投射方向。从一个视点移动到另一个视点的眼睛就能接收到一个平滑连续的透视信息，就能产生一个实体或胶片附近的3D场景。结合其他的3D计算机图形学的常规单眼深度信息，这个全息图像给人一种特别有效的形状和空间的感觉[BENT82]。

尽管全息立体图的生成成本比传统的全息图要低得多，但它们仍然需要大量的计算（每个立体全息图一打甚至几百个图像）。而且，需要在摄影胶片上记录立体信息也增加了图像生成过程的成本和时间。因此，尽管数字全息术已经证明对记录3D静态图像十分有用，就像今天的摄影胶片记录2D静态图像，但它在近期还不可能用于生成交互的3D图像。 918

18.11.6 实时飞行模拟器

能够生成用于交互任务的最真实的3D场景模拟的整体系统是数百万美元的飞行模拟器。飞行模拟不是惟一能从真正实时系统受益的领域。然而，它却是用户一直肯在一个系统上花费数百万美元的应用领域，这主要是因为在实际飞机上训练飞行员的高成本和高风险性。由于飞行模拟器的用户群很少，系统也大都使用厂商提供的专有的、面向特定硬件的软件，有关飞行模拟器体系结构的细节信息没有发表。

早期的飞行模拟器，包括通用电气的NASA II（见18.9节）和Evans &Sutherland基于扫描线的设计，是由犹他大学在20世纪60年代后期开发的。这些早期的飞行模拟器系统能实时显示

⊖ 本节的材料由MIT媒体实验室的Stephen A. Benton提供。

1000个或更多的多边形，但是都使用简单明暗处理方法，没有多少图像改进。后期的系统没有充分提高能够显示的图元数。例如，Evans & Sutherland现在的高端系统ESIG-1000，仅以60 Hz显示2300个多边形[EVAN89]。但是，系统开发者已经通过集成反走样、雾效果、点光源、云和过滤的纹理等特征，提高了系统的真实感，并减少了人工痕迹[SCHA83]。从彩图I-5a和彩图I-5b可以看出这些技术的效率。

主要制造商（如Evans & Sutherland、通用电气、McDonnell-Douglas和Singer/Link等）的飞行模拟器都有一些同样的系统特征：因为飞行模拟器包含对大数据库的可预测的交换，这些系统一般都使用比其他图形系统更特殊的处理。例如，经常使用特制的处理器来管理图像数据库、变换图元、光栅化图像和执行后面的图像增强操作。一个模拟器系统一般由专用处理器的长流水线组成[SCHA83]。

飞行模拟器中有时也可以做一些在通用图形系统中所不能的简化。例如，因为典型的模拟数据集一般包括一小部分运动物体和一个大范围的、不变的背景，可能就不需要一个通用的z缓存可见性算法，使用简单的深度排序算法就足够了。

飞行模拟器也必须有能力管理复杂数据库。一个典型的数据库表达了100平方英里范围的场景。当飞机在40 000英尺时需要的低空飞行细节就不能显示。这需要系统维护可以实时进行交换的不同级别物体的细节描述。体系结构还必须很好地处理过载的情况，因为图像复杂度可能在许多关键时刻（如在起飞、降落和紧急状态期间）急剧增长[SCHU80]。即使在这些情况
919 下，也至少每秒生成30帧图像。

18.12 小结

本章概述了用来构造高性能图形系统的体系结构技术。我们已经看到许多交互应用对计算的需求很快就超过了单处理器的能力，并且现在许多类型的处理都是为了满足3D应用（如计算机辅助设计、科学计算可视化和飞行模拟）的性能需要。

我们已经展示了如何使用两种基本的并行方法（流水线和并行）来加速显示处理的每一个阶段，以及这些方法的优点和限制。每个真实的图形系统（实际上，一般任何复杂系统）的设计都是大量的一系列相关因素（如性能、通用性、效率和成本）之间的妥协和折中。结果是，许多实际系统组合了我们上面讨论的体系结构技术。

硬件技术的当前进展也在决定什么样的体系结构是可行的问题中起到了重要作用。例如，目前作为主流的使用大量内存的体系结构，在十年以前便宜的DRAM内存出现以前，还是不符合实际的。在将来我们可以期望快速发展的技术会持续提高图形系统的性能。

对不仅仅是绘制点、线、Gouraud明暗处理多边形等标准图元绘制的需求，而是更高级性能（透明性、纹理、全局光照模型和体数据等）的需求，会使未来系统的设计更加复杂。其复杂化的不仅仅是计算，还包括显示处理的基本结构，这就使得系统的设计更加困难，它必须同时具有足够高的基本性能和处理高级特征的通用性。

习题

18.1 在5.6节中，我们说明了可以通过乘以点变换矩阵的转置逆矩阵的方式来变换一个平面方程。一个平面的法向量就是一个平面的方程去掉D分量。如果点变换矩阵由平移、旋转和缩放组成，那么变换一个表面法向量需要多少个乘法和加法？（提示：注意变换矩阵的形式。）

18.2 在显示多边形网格时，一个减少前端计算量的简单方法是使用网格图元，例如三角形带。一个三角面片带是一系列三个或多个顶点，其中每连续的三个顶点定义了一个三角形。因此，一个$n+2$个顶点的三角形带定义了n个连续的三角形（而定义n个单独的三角形需要$3n$个顶点）。如果10 000个三角面片包含在：

a. 5000个三角形带中，每个有2个三角形。

b. 1000个三角形带中，每个有10个三角形。

c. 一个包含10 000个三角形的三角形带。 920

估计一下显示18.3.9节的数据库所需要的加减法和乘除法。通过将数据库中的离散三角形转换成三角形带，能在前端子系统中取得的最大加速比是多少？使用三角形带有什么缺点？

18.3 假设18.3.9节中描述的10 000个三角形的数据库，以24 Hz在一个具有下列特征的流水线图形系统上显示：1280 × 1024分辨率，72 Hz刷新频率，32位颜色值，32位z值。

a. 估计显示流水线下列各点之间的数据带宽：(1)在显示遍历和模型变换之间（假设模型数据库中每个三角面片需要24个32位字数据）；(2)在前端和后端子系统之间（假设每个变换的三角面片需要15个32位字数据）；(3)在光栅化和帧缓存之间；(4)在帧缓存和视频控制器之间。

b. 对有100 000个多边形，平均10个像素大小和相同重叠率的数据库，重复(a)中的估计。

c. 对有100 000个多边形，平均100个像素大小，但假设只有10%像素是初始可见的数据库，重复(a)中的估计。

18.4 考虑18.7节中描述的流水线物体为序的光栅化体系结构。如果为多边形处理、边处理和跨度处理提供分别的处理器，所有这些操作能够重叠在一起。假设我们使用一个高级的跨度处理器，它能在一个250ns的时钟周期内处理整个像素（即计算它的RGB和z值，比较z值，刷新帧缓存）。如果忽略各帧之间清屏的时间，计算在下列条件下这个系统每秒能显示多少三角面片：

a. 100像素的三角面片；忽略多边形和边处理时间。

b. 100像素的三角面片；多边形处理中每三角面片20ms；忽略边处理时间。

c. 100像素的三角面片；多边形处理中每三角面片20ms；边处理中每扫描线2ms（假设一个三角面片一般包含15条扫描线）。

d. 10像素的三角面片；多边形处理中每三角面片20ms；边处理中每扫描线2ms（假设一个三角面片一般包含4条扫描线）。

e. 1000像素的三角面片；多边形处理中每三角面片20ms；边处理中每扫描线2ms（假设一个三角面片一般包含50条扫描线）。

18.5 如果构造一个每像素32位的帧缓存，对帧缓存的访问是通过一个像素（32位宽的内存系统）。那么使用下面的商业化内存部件来构造这个内存系统，并且不浪费内存（即帧缓存的所有内存都要用来存储可见像素），有1：1，5：4和2：1纵横比的帧缓存可能的大小是多少？

a. 64K × 4VRAM（256K位）。

b. 256K × 4VRAM（1M位）。

c. 512K × 8VRAM（4M位）。

假设帧缓存的大小是512 × 512，1024 × 1024和2048 × 2048（以60 Hz刷新），回答下列问题：

d. 每个内存芯片的串行部分在视频扫描输出时的访问速度必须达到的频率是多少？（假

设垂直和水平回扫时间是可忽略的，并且VRAM的输出不是多路复用的。）

e. 如果给定最快VRAM的串行部分的周期时间是大约35ns，其中哪些帧缓存是可能构造的？（同样，假设没有多路复用。）

[921]

f. 如果使用18.1.5节所述的多路复用，多个像素可以同时读出，哪些帧缓存是可能构造的？（同样假设VRAM的周期是35ns。）对每种帧缓存，必须同时读取多少个像素？

18.6 考虑18.7.1节中描述的流水线化物体为序的光栅化体系结构。

a. 如果使用下列显示器：320 × 200PC显示器，1280 × 1024工作站显示器和1840 × 1035的高清晰度TV显示器，顶点是在它们真实位置的1/10像素内指定，确定一下屏幕空间顶点坐标(x,y)的计算精度（即必须使用多少位精度）是多少？

b. 如果(a)中所有系统的多边形处理中，跨度的左右两端点位于它们的真实位置的1/10像素内，在计算多边形的左右x斜率时需要多少小数位？（假设顶点坐标以无限精度计算，且加法计算的精度是完美的。）

c. 在(a)中的所有系统中，它们的最大和最小可能的Δx值是多少？（假设在计算增量值计算以前，已经识别并去除了所有的水平边。）

d. 对这些系统，如果使用定点加法，Δx需要使用多少位才能表示(c)中Δx最大值和最小值之间的所有值并满足(b)中的计算精度？

18.7 在将帧缓存划分为连续的像素块的图像并行化系统中，如果许多图元都落入多个区域，它的性能就会下降。假设显示屏幕被分割为一些宽为W高为H的块，一个典型的图元覆盖了显示屏幕上宽为$w(w<<W)$高为$h(h<<H)$的矩形区域。假设图元出现在屏幕上任何地方的概率是相同的，用W, H, w, h构造一个表达式来表示一个典型图元所影响的平均区域数目。

[922]

第19章 高级几何与光栅算法

第3章我们介绍了图元的裁剪与扫描转换方法。本章我们将从讨论更高级的裁剪技术开始。这些都是纯几何的技术，用于几何地定义的区域对几何图元的裁剪。

在这些裁剪算法之后我们通过分析与图元相关连的属性来重新考察第3章讨论过的有关图元的描述和扫描转换。这种分析是必要的，因为诸如线型之类的属性早已在对光栅图形软件包的各种要求中提了出来。有些线型是装饰性的，比如虚线；另一些线型则是几何的，机械制图中的点划线则属于这一类。理解这些差异非常重要。

接着，我们还将探讨扫描转换过程中像素选取准则的问题，不同的准则往往会导致不同的选取结果。在简单地考察一下解析几何（analytic geometry）之后，我们将给出非整数直线、非整数圆及一般椭圆的扫描转换算法，并进一步讨论在整数世界中把任意曲线表示为折线段所存在的危险性。此外，反走样问题及其在带宽度的直线、折线和一般曲线绘制中的应用也是本章研究的内容。在此之后，我们对与在位图图形学中及带灰度级反走样的文本生成相关的问题做一些探讨，并对某些方法进行评估。另外还有可用于加速图元扫描转换操作特别是用于二值显示的数据结构，以及用于二值显示(bitBlt)的快速像素拷贝(copyPixel)操作技术。最后以重叠窗口管理、填充算法和二维页面描述图形学三个主题结束这一章。一个例子是Interpress [HARR88]，另一个是PostScript [ADOB85b]。事实上，后者远不只是一个页面描述模型，它还 923
提供了一个相对复杂的编程语言所应具有的全部功能，因此通过利用图像元素迭代概念和过程定义可以非常紧凑地描述复杂的图像。这样的页面描述语言现在不仅用于静态图像的描述，还被用于交互式图形学的屏幕描述。

19.1 裁剪

在对特定剪算法进行详细描述之前，我们首先讨论裁剪的一般过程及其在直线段裁剪中的特殊应用。正如第2章所指出的那样，裁剪是决定图元落在某一区域内部分的过程，这个区域称为裁剪区域。典型情况下裁剪区域是屏幕上的窗口或是一个视见体。其中，第二种情形可由第3章中讨论的Sutherland-Hodgman算法来处理，所以这里我们把注意力放在第一种情形。典型情况下，图元总是画在矩形画面上，因此第3章中的裁剪区域总是矩形。这是一种重要的特殊情形。在诸如Macintosh操作系统和 X Windows系统之类的多窗口环境下，各种矩形窗口相互重叠，裁剪区域可以是任意一个只有水平和垂直边的多边形构成的集合。而在PostScript这类系统中，一个裁剪区域可以由平面上任意的轮廓线集合来定义。并且，被裁剪的图元可能是一维的(例如直线)也可能是二维的（多边形围成的区域）[⊖]。很容易有这样的想法，即一旦线裁剪问题解决了，多边形裁剪就会迎刃而解：只需用裁剪窗口对多边形的所有边进行裁剪然后画出来即可。但是这种假设是不成立的，当多边形完全包含裁剪窗口时上述做法就不可行。事

⊖ 一维和二维图元的维数与其内在的几何特征有关：直线上点的位置可以用一个参数指定，从而称它为一维的。面上点的位置由可两个参数确定，因此称它是二维的。这样，即使是三维空间中的螺旋状曲线也是一个一维的图元。

实上二维图元裁剪比一维裁剪要复杂得多。

有一种画裁剪后图元的方法值得一提，尽管它本身不能算是一种裁剪方法。该方法由两步构成，首先计算出在无限画布上应画的点，然后只画那些落在裁剪区域内的部分。这种方法（第3章中把它称为截剪（scissoring））有一个缺点：绘制一个基本上落在裁剪区域外面的图元代价太大。不过截剪技术的简单性和普适性使这一缺点得到了弥补，因此对于很多系统来说这是一种较为合理的方法。如果在帧缓存中绘制一个像素所需时间长于像素的计算时间的话，那么可以在可见像素排队等候绘制时计算出其他不可见像素。用这个方法解决多窗口问题也变得简单了。由一个应用程序维护当前窗口的概念，绘画算法只是简单地在当前窗口内绘制，然后把控制返回给应用程序，应用程序再把下一个作为裁剪区域的窗口置为当前窗口。不过很多应用基本上只对一个基本的裁剪区域进行裁剪，因此研究一些先考虑裁剪区域的几何特性再进行
924 扫描转换的方法是必要的。

19.1.1 矩形区域对直线的裁剪

直线对竖直的矩形区域裁剪是一个纯几何问题，它完全独立于像素的尺寸甚至不管它是否存在，而只涉及欧几里得平面中直线与矩形的求交计算。第3章我们讨论了Cyrus-Beck/梁友栋-Barsky直线裁剪算法。Nicholl、Lee和Nicholl提出了适用于二维情形的更高效的线裁剪算法[NICH87]。尽管该算法需要考虑有很多种情形，但它的基本思想却很简单，只要理解其中一种情形其他情形就可以类推出来。

具体讨论这个算法之前，先来重述一下我们的问题：对给定的线段（零宽度）集和一个水平裁剪矩形，求出线段与矩形的交生成的端点集（可能是空集）。每一条线段由一对端点表示，水平裁剪矩形则由如下四个方程给出：$x = x_{min}$，$x = x_{max}$，$y = y_{min}$和$y = y_{max}$（参见图19-1）。为方便起见，暂时假定被裁剪线段既不是竖直的也不是水平的。

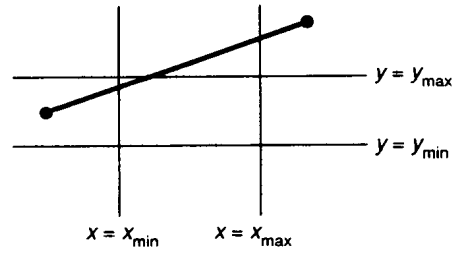


图19-1 定义裁剪矩形的方程及一条典型的被裁剪线段

最直接了当的算法是求出包含线段的直线方程，然后与裁剪矩形的边界直线求交。除了退化情形外，这四个交点要么有零个要么有两个落在裁剪矩形内（见图19-2）。如果是后者，则与原来线段的端点进行比较从而得出裁剪结果。当然这样比较要计算四个交点，即使线段完全落在裁剪矩形之内（或之外）也不例外。回顾一下第3章的参数化算法，只计算线段与裁剪矩形的边交点的参数值，并与0和1比较以确定线段是否落在裁剪矩形内（线段已被参数化使得参数值0和1正好代表它的两个端点）。只有当计算出来的参数值落在区域内时才进一步求出交点。

Nicholl-Lee-Nicholl(NLN)算法正是基于这种简单延迟策略的改进。假设被裁剪线段为PQ。我们首先确定P位于何处。如果像参数化裁剪算法一样，把平面区域分成九个相同的区域（见
925 图19-3），那么P一定落在其中一个区域里（每条边界线算到与其接触的一个区域中）。通过确定Q与从P到每个角点间直线的相对位置，就可以判断PQ与裁剪矩形的哪些边相交。

假设P位于矩形的左下角区域，如图19-4所示。如果Q位于y_{min}的下方或x_{min}的左边，那么PQ肯定不会与裁剪区域相交（这可以通过检查Cohen-Sutherland外码来判断）。对Q位于从P到左上角连线左边的情形，上述结论同样成立。还有很多情形都可以通过这样的检查很容易地排除掉。继续检查Q与P到矩形的左下角射线的相对位置关系。这里，我们将讨论Q位于该射线上方时的

情况，如图19-4所示。如果Q位于裁剪区域顶部边界的下方，那么它只能位于裁剪区域的内部或右边，从而PQ要么只与裁剪矩形的左边相交要么与左右两边都相交。如果Q位于裁剪区域顶部边界的上方，它可能位于P到左上角射线的左边；不然的话，它可能位于裁剪区域右边界的右方。后者又分两种情形：Q位于P到右上角直线的左方或右方。图19-4中区域的标号是这样规定的：与P和区域中任一点连线相交的裁剪矩形的边的名称首字母加入到标号中。例如，LT区域意思是“P与该区域任一点连线都与裁剪矩形的左边界（Left）和上边界（Top）相交”。

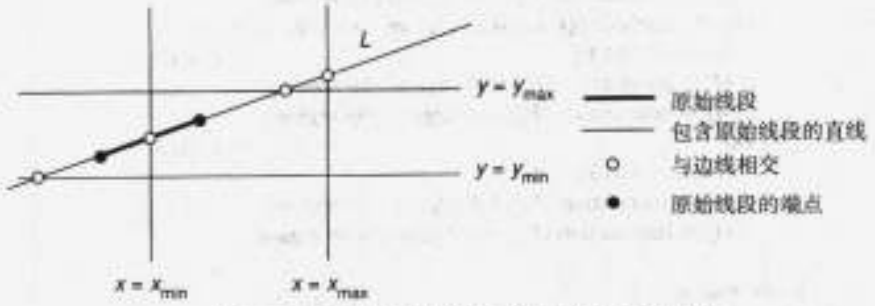


图19-2 求出与所有裁剪矩形边界线交点的裁剪

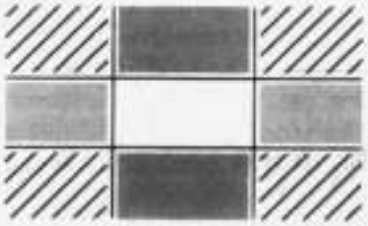


图19-3 Nicholl-Lee-Nicholl算法中用到的平面九个区域

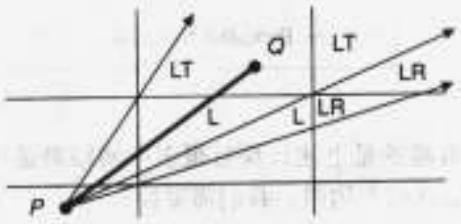


图19-4 由P到各角点间射线形成的区域

假定用函数LeftSide(*point*, *line*)来检测一个点*point*何时位于直线*line*的左边，函数Intersect(*segment*, *line*)则返回线段*segment*与直线*line*的交点。那么关于上述各种情形的算法的结构可用图19-5来表示。其中P和Q是拥有x和y两个成员的类型“point”的记录。

```
/* 矩形区域对线段PQ的裁剪，矩形由xMin, yMin, xMax, yMax确定。 */
/* 这段代码只处理了P位于矩形左下角区域的情形， */
/* 其他情形类似。 */
void PartialNLNclip(point *P, point *Q)
{
    boolean visible;      /* 如果裁剪后的线段非空则为TRUE */

    if (Q->y < yMin)
        visible = FALSE;
    else if (Q->x < xMin)
        visible = FALSE;
    else if (LeftSide (Q, ray from P to lower-left corner)) {
        if (Q->y <= yMax) {                          /* 区域L或LR */
            visible = TRUE;
            *P = Intersection (PQ, left edge of clip region);   /* 把交点存入P中 */
            if (Q->x > xMax)                         /* 区域LR */
```

图19-5 Nicholl-Lee-Nicholl算法当P位于矩形左下角区域时的代码

```
            *Q = Intersection (PQ, right edge of clip region);
        } else {
            /* 顶边之上 */
            if (LeftSide (Q, ray from P to upper-left corner))
                visible = FALSE;
            else if (Q->x < xMax) {                          /* 第一个区域LT */
                visible = TRUE;
                *P = Intersection (PQ, left edge of clip region);
                *Q = Intersection (PQ, top edge of clip region);
            } else if (LeftSide (Q, ray from P to upper-right corner)) {
                visible = TRUE;                              /* 区域LT */
                *P = Intersection (PQ, left edge of clip region);
                *Q = Intersection (PQ, top edge of clip region);
            } else {                                         /* 区域LR */
                visible = TRUE;
                *P = Intersection (PQ, left edge of clip region);
                *Q = Intersection (PQ, right edge of clip region);
            }
        } /* else */
    } else
        /* Q位于P到矩形左下角连线右边的情形 */
        ...
} /* PartialNLNclip */
```

图19-5（续）

有趣的是上述计算有很多中间结果是可重用的。例如，在计算Q是否位于P到右上角(x_{max}, y_{max})射线的左边时，我们需要检查

$$(Q.y - P.y)(x_{max} - P.x) - (y_{max} - P.y)(Q.x - P.x)$$

是否为正（见习题19.25）。对其他边也用到类似的计算，而且式$Q.y - P.y$和$Q.x - P.x$在所有情形中都出现，因此这些式子一旦计算好后就保存起来。公式中的两个乘积也被重用到，所以也把它们保存下来。举例来说，如果Q位于射线的右边，那么求PQ与右边界的交点时，y坐标可由下式给出：

$$P.y + (Q.y - P.y)(x_{max} - P.x)(1 / (Q.x - P.x))$$

这说明第一个乘积可重用（上述公式由直线的点斜式方程得到）。由于与右边界的求交还用到$Q.x - P.x$的倒数，它也被保存起来。我们把如何尽可能提高这部分特殊代码的效率作为练习留给读者（习题19.1）。剩下的情形，如P位于中心区域或旁边区域是类似的。因此，识别各种情形的对称性然后写一个程序把三种一般的情形（P在中心区域、角区域或边区域）转换成九种不同情形是有意义的。

Nicholl、Lee和Nicholl通过对NLN算法、Cohen-Sutherland（CS）算法和梁友栋-Barsky（LB）算法进行分析之后发现：(1)NLN算法除法最少，与输出的交点数相同；(2)NLN进行的比较也最少，大概是CS算法的三分之一，LB算法的一半。此外他们还注意到，如果减法比加法慢，除法比乘法慢并且第一个差小于第二个差那么他们的算法的效率是最高的。当然，不像另两种算法，NLN算法只能处理二维裁剪。

更一般的区域对直线的裁剪是这类区域对一般图元裁剪的特殊情形。下一小节我们将讨论任意多边形区域对多边形的裁剪。在PostScript和其他一些图像模型中定义的任意区域对一般图元的裁剪则在本章后半部分论述，因为那是由光栅算法实现的。

926 ~ 928

19.1.2 矩形和其他多边形区域对多边形的裁剪

在一个矩形区域中画一个多边形时，我们可能希望先对多边形进行裁剪以节约绘图时间。对于真正意义上的一般裁剪区域（例如，被裁剪区域的内部通过给定一个与屏幕某一区域对应的画面来指定，而裁剪区域的内部则由与画面中黑色像素对应的点组成），截剪技术是完全可行的办法：我们简单地计算图元的所有（光栅化的）点，而把落在光栅化裁剪区域内部的那些点画出来。19.7节描述的形状代数也可用来快速确定裁剪区域像素与图元像素之间的重叠部分，如果裁剪区域不是极端复杂——如灰度级位图，这种方法是很有效的。因为解析的裁剪方法本身就很有趣，而且除了窗口系统以外还有其他应用，这里我们打算详细讨论两个算法，即梁友栋-Barsky（LB）算法[LIAN83]和Weiler算法[WEIL80]。

对于多边形被裁剪后结果的构成有不同的观点。图19-6显示一个多边形被矩形裁剪后有两种可能的输出，一种是连通的，另一种是不连通的。Sutherland-Hodgman算法[SUTH74b]和梁友栋-Basky算法都生成连通的多边形，虽然这种多边形可能包含有退化的边（例如与别的边重叠或长度为0的边）。Weiler算法则生成非退化多边形，因而有时是不连通的。因为Weiler算法不是专为裁剪设计的（它可以产生多边形的任意布尔组合），很显然，当多边形完全不相交时，这些组合需要由不连通的多边形来表示[⊖]。图19-7给出两多边形A和B，在不引入交点的情况下（通过给两者增加一对退化边），$A-B$及$B-A$不可能同时表示成连通的多边形。

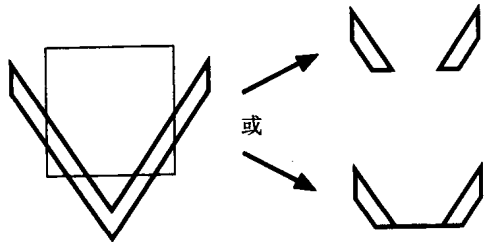


图19-6 V形多边形的裁剪结果应该包含退化边吗?

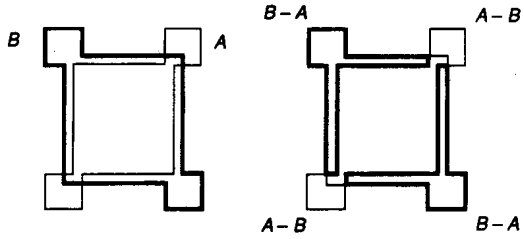


图19-7 如果不引入交点，区域$A-B$和$B-A$不可能表示成（原始多边形表示的）连通多边形

实际上，输出多边形中的退化边可能是与实际结果无关的。如果多边形仅仅用来定义一个填充区域，由于退化边之间没有面积，因而不会引起什么问题。但如果是用来定义一条折线，退化边就应该删掉，详情可参见习题19.2。

19.1.3 矩形区域裁剪:梁友栋-Barsky多边形算法

我们首先来讨论梁友栋-Barsky（LB）算法。为区别被裁剪多边形和用来裁剪多边形的矩形，我们把水平的裁剪矩形称为*窗口*，而被裁剪的多边形称为*输入多边形*，裁剪的结果称为*输出多边形*。每条被裁剪的边用参数形式表示，它们与窗口边的交点在需要时才计算。不过与直线裁剪不同的是，完全位于窗口之外的边也有可能对输出多边形作出贡献。考虑输入多边形完全包围窗口的情形：输出多边形是窗口的边界，如图19-8所示。

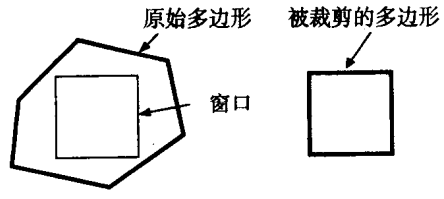


图19-8 位于窗口外的边也可能对输出（裁剪后的）多边形有贡献

倾向于严格数学表示的读者可能会对图

⊖ Weiler算法也处理一般多边形——有洞的多边形。这些多边形由多重嵌套的轮廓表示。

19-9中的结果有异议：输入多边形的内部与窗口完全不相交，但是LB算法却产生一个包含所有窗口边界的多边形（尽管在多边形中，窗口的每条边界边沿两个方向各被包含一次，如图19-9中的虚线所示，注意，为清楚起见，虚线被画在稍离窗口边界之外了）。习题19.2讨论删除这类多余边以获得裁剪结果的最小形式输出的一些算法。

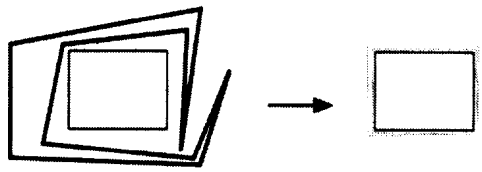


图19-9 输入多边形完全位于窗口之外，但梁友栋-Barsky算法却生成一个非空的结果

假定输入多边形由顶点序列$P_1, P_2, \cdots, P_n$确定，$P_1P_2, P_2P_3, \cdots, P_nP_1$，为多边形的边。每条边可看成是从$P_i$到$P_{i+1}$的向量，因而其参数形式为$P(t) = (1-t)P_1 + tP_2$，$t$的值介于0和1之间。（更精确地，设$0<t\leqslant 1$表示边上的点，使得每条边的起始点不被包含在内。从而多边形的每个顶点恰好被包含在一条边中。这里删除起始点的选择与第3章中不同，但它能使算法的解释稍为简单些。）其他的t值则表示这条边所在直线的不在边上面的点。在LB算法中，一次只考虑一条边P_iP_{i+1}，用L_i表示包含P_iP_{i+1}的直线。

最初我们只考虑斜线——那些既不是水平也不是竖直的直线。这样的直线肯定与窗口的每条边界都有交点。事实上，如果我们把平面划分成由窗口边界确定的九个区域，如图19-10所示，显然每条斜线都从一个角区域穿到相对的另一个角区域。每条窗口边界把平面分成两个半平面。我们把包含窗口的半平面称为*内半平面*。图19-10的九个区域都标上了包含它们的内半平面的个数。当然，窗口是惟一位于四个半平面内的区域。我们把角点处的区域（标有“inside 2”的区域）称为*角区域*，其他的外部区域（标有“inside 3”的区域）称为*边区域*。

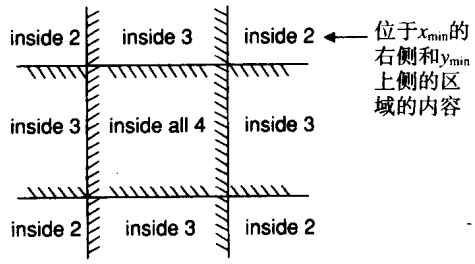


图19-10 由窗口边线确定的平面的九个区域，每个区域至少位于两条边线的内侧

进一步讨论细节之前，我们先通过几个例子来说明这一算法的使用。首先，如果多边形边（注：不是包含它的直线）的某部分位于窗口内，那么这部分肯定也是输出多边形的一部分。这条边对多边形顶点的贡献可能是多边形边的两个端点（此边完全位于窗口内）或者是与窗口边的两个交点（此边的两个端点都位于窗口之外），也可能是一个端点和一个交点。

929
~
931

另一方面，如果某条边整个位于窗口之外，那么下一条边可能会与窗口相交(见图19-11)。如果是这样的话，交点位置将由它的起始点确定：起点在上边区域的的边，它只能首先与窗口的上边界相交；起始于左上角区域的边只能与窗口的上边界或左边界相交，等等。

假设多边形前一条边与窗口相交产生位于窗口上边界的顶点，而下一条边与窗口相交产生位于窗口右边界的顶点，如图19-12所示，那么输出多边形将包含右上角点作为其顶点。因为一次处理多边形的一条边，我们不得不在此处把角点加入到顶点序列中，以期寻求与裁剪窗的下一个交点。当然如果下一条边与窗口的上边界相交，此顶点将成为多余，不管怎样，先把它加进去，而把冗余顶点的删除留在后处理部分完成。其思想是，一条边被处理完后，任何由下一条边加入的交点，必须能够由上一个输出顶点到达。

一般说来，进入角区域的边将加入相应的窗口角点作为输出顶点，梁友栋和Barsky称此为*转折顶点*。（原算法的处理顺序稍有不同：不是在某条边进入角区域时加入转向顶点，而是把它延迟到某条后续边离开角区域之后。这不能完全消除退化边问题，而且我们发现这使得算法更难于理解，因此使用了我们的描述。）

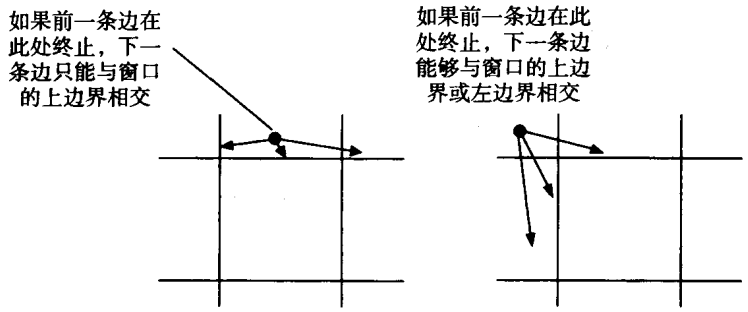


图19-11 如果前一条边终止于上边的中间区域而后一条边与窗口相交，那么交点只能在上边界。如果前一条边终止于左上区域而后一条边与窗口相交，那么交点只能在窗口的上边界或左边界

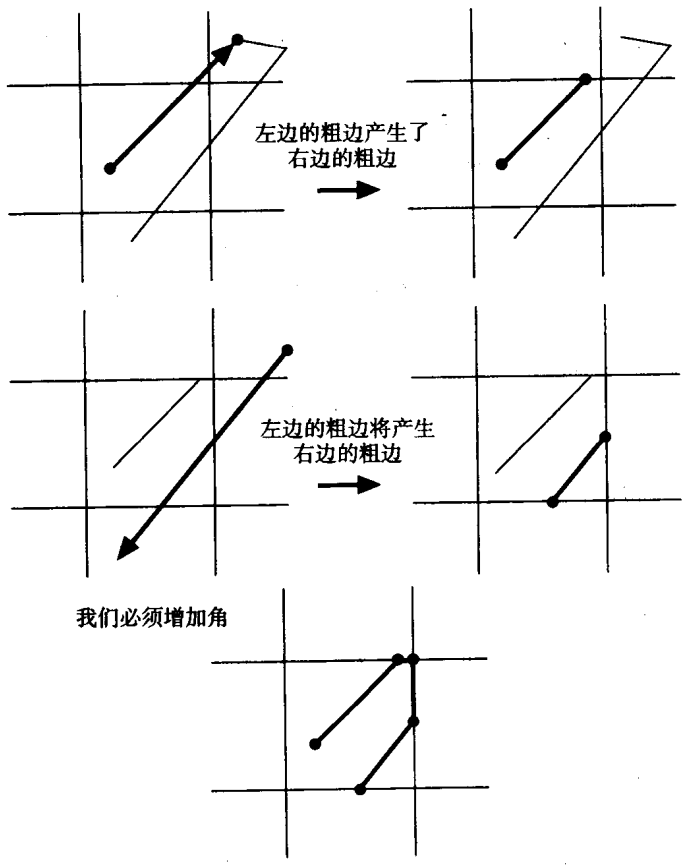


图19-12 在窗口的右上角的点必须加到输出多边形中:下一条边（它的上一条边与窗口不相交）或者与上边界或者与右边界相交。我们必须加入可到达所有可能交点的顶点

现在我们利用第3章裁剪算法中对参数形式的分析来仔细考查各种情形。包含$P_i P_{i+1}$的直线L_i与窗口的四条边界相交。其中两个是潜在的进入交点，而另两个是潜在的离开交点。计算出交点的参数值，并分别把它们记为t_{in_1}，t_{in_2}，t_{out_1}和t_{out_2}。注意t_{in_1}是其最小值，而t_{out_2}是其中的最大值，这是因为每条斜线都从一个角区域跨到另一个角区域。t_{in_2}和t_{out_1}处于两者之间，顺序不定。正如第3章中提到的，如果$t_{in_2} \leqslant t_{out_1}$，则直线与窗口相交；反之，如果$t_{in_2} > t_{out_1}$时直线经过一个角区域（见图19-13）。

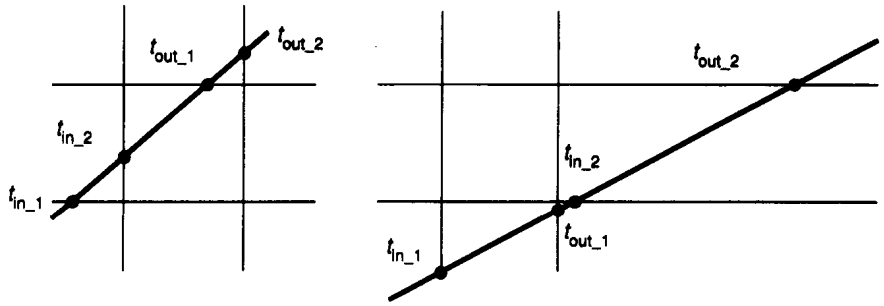


图19-13 跨越窗口的直线的两种可能情形

参数值$t=0$和$t=1$定义了直线L_i上多边形边的两个端点。这两个参数值与t_{in_1}，t_{in_2}，t_{out_1}和t_{out_2}的关系刻画了边对输出多边形的贡献。如果边与窗口相交，那么它的可见的一段应被加到输出多边形中，此时$0<t_{out_1}$且$1\geq t_{in_2}$，也就是说多边形的边在包含该边的直线离开窗口之前开始而在此直线进入窗口之后终止——多边形边并非完全位于窗口之外。

如果多边形的边与窗口不相交，那么它所在的直线肯定是从一个角区域开始，穿过第二个角区域，然后在第三个角区域终止。如果多边形的一条边进入后两个角区域中的任一个，那么应加入一个转折顶点到输出多边形中。边进入第二个角区域由$0<t_{out_1}\leq 1$表征（因为t_{out_1}是直线进入角区域时的参数值）。边进入第三个角区域则由$0<t_{out_2}\leq 1$刻画。

上述最后一个结论对于直线与窗口相交的情形也是成立的，即如果$0<t_{out_2}\leq 1$那么应加入一个转折顶点到输出多边形中。

图19-14给出算法的一个框架。注意各条直线所贡献的顶点必须按顺序加入到输出多边形的顶点序列中。因为边为竖直或水平时的情形也必须考虑在内，完整的算法会略微复杂一些。有两种可能的方法来处理这些情形。第一种是把此类边当作特殊情况处理，简单地扩展上述代码去检测并处理它们。检测的方法很简单：*deltaX*和*deltaY*（这是边的方向向量的两个分量）有一个为0即属于这种情况。处理过程则是对可能的情况逐一进行分析。例如，位于窗口右边的竖直边可能进入右上角区域也可能进入右下角区域，因而有可能贡献一个转折顶点；一条介于窗口左右边界之间的竖直边则可能与窗口相交从而贡献一段可见线段，也可能对输出多边形无任何贡献，等等。另一种方法是给进入和离开参数值赋予$\pm\infty$，从而强制性地使每条水平或竖直边遵从算法余下部分的模式。

```
for（每条边e）{
    确定边的方向
    用此来确定包含线首先与裁剪区域的
        哪条边界线相交
    寻找退出点的t值
    if (tOut_2 > 0)
        寻找第2个进入点的t值
    if (tIn_2 > tOut_1) {                    /* 无可见线段 */
        if (0 < tOut_1 && tOut_1 <= 1)
            Output_vert (turning_vertex);
    } else {
        if (0 < tOut_1 && 1 >= tIn_2) {      /* 有一些可见部分 */
            if (0 <= tIn_2)
```

图19-14 梁友栋-Barsky多边形裁剪算法的框架

```
                Output_vert（适当的面交点）；
            else
                Output_vert（起始顶点）；
            if (1 >= tOut_1)
                Output_vert（适当的面交点）；
            else
                Output_vert（终止顶点）；
        }
    }
    if (0 < tOut_2 && tOut_2 <= 1)
        Output_vert（适应的角点）；
}   /* 对每条边 */
```

图19-14（续）

图19-15给出了采用后一种解决方案的完整算法。我们假定实变量可被赋予一个*infinite*的值使得对所有不等于*infinite*的*x*都有*x*<*infinite*。如果不能定义*infinite*值，则只能使用特殊方式来处理水平和竖直边的情形，此时算法的细节作为习题19.3请读者完成。为了简化算法，我们定义一个宏AssignTwo实现同时对两个分量赋值。代码中还使用了几个可以消去的变量（比如*Xin*和*Xout*分别用来表示边所在的直线进入和离开时裁剪窗口的边）。不过这些变量可以减化代码的case结构，从而增强可读性。OutputVert例程（没有写出）把结果存到*u*和*v*中并使计数器加1。

```
#define MAXPT 50;
#define MAX2 150;

typedef double smallarray[MAXPT];
typedef double bigarray[MAX2];

/* 完成两个赋值 */
#define ASSIGNTWO( x, y, a, b) { \
    (x) = (a); \
    (y) = (b); \
}

/* 将n边输入多边形裁剪到一个窗口 */
void LiangBarskyPolygonClip(
        int n,
        const smallarray x, const smallarray y,     /* 输入多边形的顶点 */
        bigarray u, bigarray v,                     /* 输出多边形的顶点 */
        double xMax, double xMin,                   /* 裁剪窗口的边 */
        double yMax, double yMin,
        int *outCount)                              /* 输出顶点计数器 */
{
    double xIn, xOut, yIn, yOut;                    /* 进入和退出点的坐标 */
    double tOut1, tIn2, tOut2;                      /* 相同的参数值 */
    double tInX, tOutX, tInY, tOutY;                /* 相交的参数值 */
    double deltaX, deltaY;                          /* 边的方向 */
    int i;
    x[n] = x[0];
    y[n] = y[0];                                    /* 使多边形封闭 */
```

图19-15 梁友栋-Barsky多边形裁剪算法

```
*outCount = 0;                                  /* 初始化输出顶点计数器 */
for (i = 0; i < n; i++) {                       /* 对每一条边 */
    deltaX = x[i + 1] - x[i];                   /* 确定边的方向 */
    deltaY = y[i + 1] - y[i];
    /* 用此来确定包含线首先与裁剪区域 */
    /* 的哪一条边界线相交 */
    if ((deltaX > 0) || (deltaX == 0 && x[i] > xMax))
        ASSIGNTWO (xIn, xOut, xMin, xMax)
    else
        ASSIGNTWO (xIn, xOut, xMax, xMin)
    if ((deltaY > 0) || (deltaY == 0 && y[i] > yMax))
        ASSIGNTWO (yIn, yOut, yMin, yMax)
    else
        ASSIGNTWO (yIn, yOut, yMax, yMin)
    /* 寻找退出点x, y的t值 */
    if (deltaX != 0)
        tOutX = (xOut - x[i]) / deltaX;
    else if (x[i] <= xMax && xMin <= x[i])
        tOutX = ∞;
    else
        tOutX = -∞;
    if (deltaY != 0)
        tOutY = (yOut - y[i]) / deltaY;
    else if (y[i] <= yMax && yMin <= y[i])
        tOutY = ∞;
    else
        tOutY = -∞;

    /* 对两个退出点排序 */
    if (tOutX < tOutY)
        ASSIGNTWO (tOut1, tOut2, tOutX, tOutY)
    else
        ASSIGNTWO (tOut1, tOut2, tOutY, tOutX)
    if (tOut2 > 0) {                            /* 可以有输出——计算tIn2。 */
        if (deltaX != 0)
            tInX = (xIn - x[i]) / deltaX;
        else
            tInX = -∞;
        if (deltaY != 0)
            tInY = (yIn - y[i]) / deltaY;
        else
            tInY = -∞;
        if (tInX < tInY)
            tIn2= tInY;
        else
            tIn2 = tInX;
        if (tOut1 < tIn2) {                     /* 无可见线段 */
            if (0 < tOut1 && tOut1 <= 1) {
                /* 线穿过中间角区域 */
                if (tInX < tInY)
                    OutputVert (u, v, outCount, xOut, yIn);
                else
```

图19-15（续）

```
                OutputVert (u, v, outCount, xIn, yOut);
            }
        } else {
            /* 线穿过窗口 */
            if (0 < tOut1 && tIn2 <= 1) {
                if (0 < tIn2) {                    /* 可见线段 */
                    if (tInX > tInY)
                        OutputVert (u, v, outCount, xIn, y[i] + tInX * deltaY);
                    else
                        OutputVert (u, v, outCount, x[i] + tInY * deltaX, yIn);

                    if (1 > tOut1) {
                        if (tOutX < tOutY)
                            OutputVert (u, v, outCount, xOut, y[i] + tOutX * deltaY);
                                else
                                    OutputVert (u, v, outCount, x[i] + tOutY * deltaY, yOut);
                                }
                            } else
                                OutputVert (u, v, outCount, x[i + 1], y[i + 1]);
                        }
                    }
                    if (0 < tOut2 && tOut2 <= 1)
                        OutputVert (u, v, outCount, xOut, yOut);
                } /* if tOut2 */
            } /* for */
        } /* LiangBarskyPolygonClip */
```

图19-15（续）

19.1.4 Weiler多边形算法

现在我们转到任意多边形对另一个多边形的裁剪问题，这在15.7.2节进行可见面计算时出现过的。图19-16是几个多边形裁剪及其结果的示意图。应当注意到，裁剪的结果多边形有可能是不连通的，而且即使原被裁剪多边形是凸的其结果多边形也可能是非凸的。

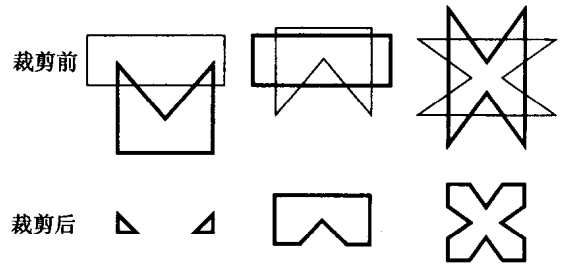


图19-16 多边形对多边形裁剪的几个例子

Weiler多边形算法[WEIL80]是对更早的Weiler-Atherton算法[WEIL77]的改进。它基于如下的观察，如果我们用黑色铅笔在一张白色的纸上画出被裁剪多边形*A*和裁剪结果多边形*B*的边，那么纸上仍为白色的部分被分成几个不相交的区域（如果把多边形的边想像成地图的边界，那么这些区域就是一个个的国家）。每个这样的区域可能完全被包含在*A*中、包含在*B*中或者既包含在*A*中又包含在*B*中，也可能既不在*A*中又不在*B*中。算法通过找到平面中这些不相交区域边界的封闭折线表示而达到目的。裁剪结果多边形由所有既包含在*A*中又包含在*B*中的区域构成。在这个算法里，被裁剪多边形与输入多边形是同样的意思；不过由于我们想要找到位于两个多边形中的区域，所以以下我们将把被裁剪多边形和裁剪多边形分别用*A*和*B*来表示。（注意到这一裁剪方法与第12章多面体构造实体几何有相似之处。事实上，我们可以把多边形-多边形裁剪称为构造平面几何）。

具体讨论算法的细节之前，我们通过一个例子来揭示它的大部分细微之处。图19-17a是两个相交的多边形*A*和*B*。在图19-17b中，交点作为顶点加到了两多边形中，就是图中的实心圆点。如果两条边只交于一点（称为跨越交点），就把这个交点作为新顶点加到两多边形中；如果两条边有一段重叠，那么此两边中任一边的落在重叠段上的端点都被作为新顶点加入到另一条边中。图19-17c则是把原来两个多边形的边都变成双线的结果。尽管图中把加双线产生的新多边形画得与原多边形稍有些偏离，但应把新多边形的边看成是与原始多边形的边无限接近的。这样得到的曲线段称为轮廓线。一个多边形的每条边贡献两条轮廓线。我们用段来表示轮廓线的一个部分，而边则仍保留给原始多边形。生成这些轮廓线之后，我们用多边形的名字（*A*或*B*）来标识每个多边形内部轮廓线的每一段，而外轮廓线上的段则用*X*标记。（图中只显示了部分标记；同样，两条边重叠的部分实际有四个轮廓段，但图中只画出了两个。）

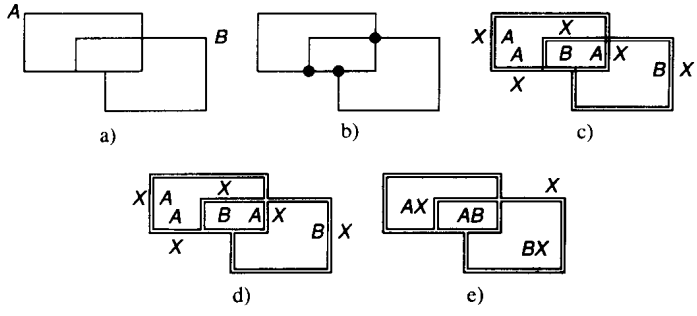


图19-17 应用于两个多边形的Weiler多边形算法。a)两个多边形*A*和*B*，b)交点作为顶点加入多边形中，c) 把多边形轮廓变成双线并打上标记的结果，d)重新连接轮廓线使得它们互不相交，e)收集轮廓线标记，赋给它们所包围的区域

算法的思想是重新安排轮廓线，使得它们能够形成前面提到的不相交区域的边界，这正是图19-17d中所做的工作。图19-17e则收集每个轮廓线的标记以给出区域的标记。这些标记决定该区域位于哪一个原始多边形中。对两多边形相交的区域，我们只对标有*A*和*B*的区域感兴趣（图中用*AB*来标明）。在实际操作中，双轮廓线在第一次读入多边形时产生；作为算法的一个高效实现，轮廓线应在求出交点之后合并。

事实上该算法可用于任意个数的多边形。当多于两个的多边形相交时，可能会发生一些区域完全包含在另一些区域中的情形（例如，一个正方形包含在另一个更大的正方形中）。在这样的情况下，我们需要判定包含关系以使算法更完善。

算法包括三个步骤：初始化，区域判定及选择同时位于*A*和*B*中的区域。初始化包括两步：确定多边形*A*的边与多边形*B*的边的所有交点（因为所有交点将作为输出多边形的顶点），然后重新定义两多边形使得它们把这些交点也作为顶点。计算几何的标准算法可用来确定所有边的交点［PREP85］。

为确定区域必须对廓线进行重排。我们想要使所有轮廓线互不相交。因此，应该对两个多边形的相交顶点处进行调整。假设我们有一个两个多边形的跨越交点，如图19-18a所示。相应的轮廓线如图19-18b所示。轮廓线用双向链表来实现，我们希望重新安排链接关系以得到如图19-18c所示的模式。在跨越交点处，取出与一个多边形相关联的段合并到与另一个多边形相关联的轮廓线段中，一次一对，这样可以达到上述目的。因此，在图19-18d中，我们从竖直的轮廓线段开始，并希望合并与边*R*相关联的两个段。图19-18e则显示了通过重新安排连接关系来实现的把这些段合并到竖直轮廓段中的例子。与边*T*和*B*相关联的段都有两面（就是说有两条

竖直轮廓线），把新的段加入到适当的一面是至关重要的。我们通过比较边R的远顶点（不是位于交点处的那个顶点）与竖直线（包括T和B两边）来确定R在哪一面。至此，只剩下与边L相关联的段没有处理了。在图19-18f中，与L相关的段也被加入到了轮廓线中，从而最后形成的轮廓线在此顶点处不再有交点。注意，这一过程只用到了局部信息，即我们只需知道所考虑的边的位置，而无须涉及多边形的其他内容。

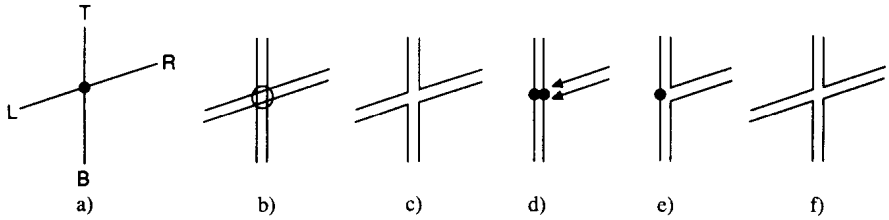


图19-18 边相交时的轮廓线合并。a) 两多边形交于一个顶点，b) 边的轮廓线，c) 轮廓线重排的最终结果，d) 把边R贡献的轮廓线段加到竖直轮廓线段中，e) 所得的结果，f) 来自边L的类似的轮廓线段合并的结果

在非跨越交点处轮廓线的合并更为困难，如图19-19所示。图19-19a中是两个多边形的一个非跨越交点的情形：它们共有一段短竖直线段。当然在实际的数据结构中，这一竖直线段是每个多边形的一条边。我们称这样的边是*重合的*。值得注意的是重合边较容易处理。每条这样的边都贡献两个段给轮廓线，每面一个段，就像图19-19b所示的那样，图中一个轮廓线集合被画在另一个轮廓线集合内以示区别，而虚线则表示原来多边形共有的边。每段有一个标记（我们已经在图19-19b中给出了标记的样例）。在重合边的同一面的每两个段的标记合并在一起，而把与其中一个多边形中的重合边对应的段删掉。这样产生的中间结构如图19-19c所示。合并后的标记用来标明剩下的与重合边相关的段。箭头则表示未处理的指针。在图19-19d和19-19e中，与未处理指针相对应的段和以前一样与另一条轮廓线合并。如果绕着一个多边形按顺序处理轮廓线，那么我们最多只能够得到一个未处理指针集合。如果这个集合对应于一个跨越交点，则可以像前面论的那样处理；如果对应的是另一条重合边，则可以像这条边一样处理。

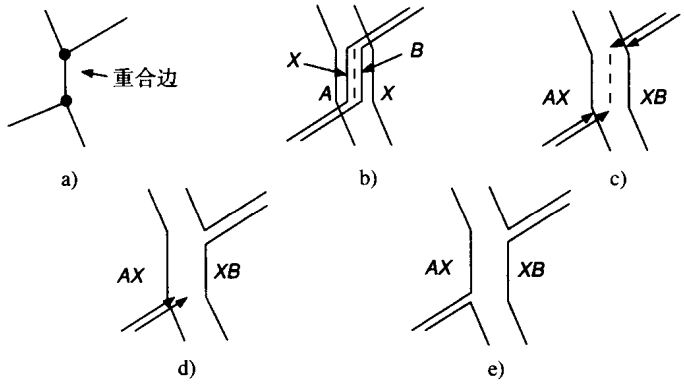


图19-19 沿重合边合并轮廓线。a)一条曲折轮廓线与另一条竖直轮廓线共一段短的竖直重合边；b)与边的这种排列相关连的轮廓线；原始的竖直边用虚线表示；c)曲折多边形的竖直段已被删除，它们的标记被合并到别的段的标记里；d)合并一条相连边的轮廓线段之后；e)另一段相连边的轮廓线段合并之后

图19-20a显示了多边形另一类非跨越相交的情况。因为每条对角边与每条竖直边都是跨越

相交，其轮廓线合并的过程不会比最初的跨越相交情形复杂。合并前后的轮廓线结构分别如图19-20b和19-20c所示。

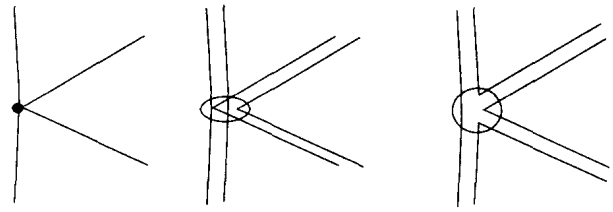


图19-20 在相切交点处的轮廓线合并。a) 边的构形，b) 初始轮廓构形，c) 最后轮廓构形

最后，需要指出的是，在处理交点的过程中，与原始多边形相关联的内外轮廓线被分裂成子轮廓线。由于输出多边形是由这些子轮廓线导出的，因此必须把它们记录下来。我们通过对每个轮廓线至少维护它的一个段的指针来保持上述记录。如果我们跟踪任何一次求交处理过的所有段，这种技术要求保证提供一个入口点的表，尽管对某些轮廓线来说这个表可能有几个入口点。事实上，对每个交点，可以为每段创建一条新的轮廓线，并且把轮廓线的startingPoint域设置为相应的段。此外，还设置一个从段到轮廓线的反向指针（即contourPtr域），稍后会用到它。

通过上述各种任务的描述，我们得到Weiler算法的数据结构：这就是图19-21所示的顶点、边和轮廓线。一条边有顶点和两个面。这些面是最初形成多边形内轮廓线和外轮廓线的段，最后会对输出轮廓线有贡献。因此，一个边面也就是我们原来所说的一条轮廓线的一段。每个边面指向它的顺时针和逆时针邻居，同时还有一个它的拥有者的表（即标记）。除刚才提到的情形外，每个边面的contourPtr域总是空指针NULL。

当算法第一次读入多边形时，应对它们进行处理以建立相应的数据结构。如何有效地建立这样的数据结构，我们把它作为习题19.4让读者完成。第二步是根据*A*和*B*的数据结构创建一个由许多轮廓线组成的统一的数据结构。每一条轮廓线是某个区域的一条边界。由于*A*和*B*的所有边的交都将作为轮廓线的顶点，所以需要首先求出交点。

932
~
941

```
#define SIDE 2      /* 每个多边形有两个面 */
#define END  2      /* 每条边有两个端点 */

typedef enum {CW, CCW} direction;
        /* 轮廓线的顺时针、逆时针方向 */

typedef struct {                    /* 外部区域A和B */
    unsigned int X:1;
    unsigned int A:1;
    unsigned int B:1;
} owners;

typedef struct {
    double x, y;
} vertex;

typedef struct contourStruct contour;
typedef struct edgeStruct edge;
```

图19-21 Weiler算法中使用的数据结构

```
struct contourStruct {
    owners belongsTo;                /* 在遍历期间填充 */
    struct {                         /* 轮廓线遍历开始的位置 */
        edge *entryEdge;
        int entrySide;
    } startingPoint;
};

struct edgeStruct {
    vertex *vertices[END];           /* 边的端点 */
    edge *edgeLinks[SIDE][2];        /* 双向链表结构 */
    owners edgeOwners[SIDE];         /* 内部和外部的拥有者 */
    contour *contourPtr[SIDE];       /* 指向任何其入口点在这个 */
                                     /* 边面上的轮廓线 */
};
```

图19-21（续）

接下来是按前面所描述的那样在交点处合并轮廓线，并生成新的轮廓线数据结构。至此，我们得到一个轮廓线数据结构集，每个结构的startingPoints成员指向各种边面；从这样一个边面开始，我们可以沿所有链表遍历一条轮廓线。这些边面的拥有者标记可以彼此不同。整个轮廓线的拥有者可以通过合并轮廓线的边面的拥有者得到。

我们必须逐条遍历所有的轮廓线来收集这些信息。对每条轮廓线，沿着与startingPoints相关联的边面的表前进。如果遇到一个顶点指回到另一个轮廓线的数据结构，就把这个数据结构删掉，因此对平面上的每条轮廓线恰好剩下一个数据结构。这个过程的伪代码如图19-22所示。

```
for (每一个轮廓线c) {
    c.belongsTo = NULL;
    for (c上的每个边-面对e和s，在c.startingPoint开始) {
        /* 如果边面不指向c，删除它指向的轮廓线
        if (e.contourPtr[s] != &c)
            delete *e.contourPtr[s];
        c.belongsTo |= e.edgeOwners[s];
    }
}
```

图19-22 轮廓线合并的伪代码

现在我们得到了一个完备的轮廓线集，每个对应一个惟一的入口点。如果输入多边形是平面上的简单闭曲线，问题就算解决了：输出多边形也是简单的闭曲线，轮廓线之间没有相互包含关系，因此我们可以简单地选取那些其拥有者集同时包含A和B的轮廓线。但是，通常情况下问题更为微妙：Weiler算法可用于求出A和B的任意布尔运算，其中的一些组合可能会产生空洞（如果A和B都是正方形，且A在B中，那么$B-A$就有一个洞）。A和B本身也可以有洞。这种情况下的交将变得很复杂。由于这类情形在某些应用中很重要，例如三维裁剪（见第15章），我们继续进行一些分析。注意到算法可以应用于由轮廓线集定义的多边形，因此这类多边形的并、差和交都应能够由这个算法获得。

至此我们得到了不相交的轮廓线集，每条轮廓线都有一个拥有者集合以指明它位于哪一个原始多边形中。另一方面，由这些轮廓线围成的区域有可能重叠（正如两个嵌套正方形所形成的那样）。为确定算法的输出，必须确定轮廓线的嵌套结构。我们通过用二叉树存储轮廓线来

实现这一目标，在二叉树中轮廓线的左子节点代表嵌套在它里面的轮廓线，而右子节点则是与它有相同嵌套深度的轮廓线。（我们先给轮廓线的结构增加两个域containedContour和coexistingContour，每个域都是一个指向一条轮廓线的指针。图19-23显示了一个嵌套的轮廓线集及其相应的树结构（树结构一般不惟一）。

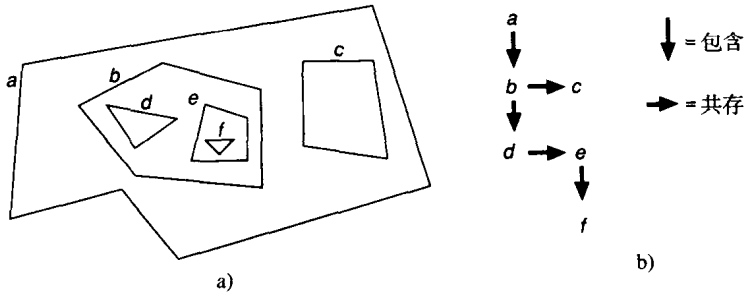


图19-23 a)平面上的轮廓线集合，b)相应的树结构

942
~
943

图19-24列出了一个构造这种树结构的算法的伪代码。注意到每次遍历表时至少有一条轮廓线被分配，因此算法最终会停止。读者可以尝试着找出一个效率更高的算法来生成包含树（见习题19.5）。

```
for（每一条未赋值轮廓线c）{
    将这个轮廓线与未赋值列表中的所有其他轮廓线比较；
    if（它在惟一一个内部，q）{
        从列表中删除c；
        if (q.contained == NULL)
            q.contained = &c;
        else
            Insert (&c, &q);      /* 在q将c插入树中 */
    } else if（它被多个包含）
        跳过这个轮廓线并进入下一个轮廓线；
}

void Insert(contour *c, contour *q)
{
    if (q->contained == NULL)
        q->contained = c;
    else if (*c contains *(q->contained)) {
        c->contained = q->contained
        q->contained = c;
    } else if (*(q->contained) contains *c)
        Insert(c, q->contained);
    else if (q->contained->coexist == NULL )
        q->contained->coexist = c;
    else if (*c contains *(q->contained->coexist)) {
        c->contained = q->contained->coexist;
        q->contained->coexist = c;
    } else if (*(q->contained->coexist) contains *c) {
        Insert(c, q->contained->coexist);
    }
}  /* Insert */
```

图19-24 从嵌套轮廓线集合构造包含树结构的伪代码

算法结束后，我们就得到一棵确定轮廓线包含关系的树结构。此后，就可以根据我们所感兴趣的布尔运算选择相应的子树（对裁剪操作，这样的子树由所有其拥有者表中包含标记A和B的轮廓线组成）。

结束Weiler算法的讨论之前，我们来看一下处理两个以上的多边形时还需要做些什么。很显然，边和轮廓线的拥有者域应该扩大以便能够包含我们一次想要处理的多边形的个数。除了这一改变之外，算法基本保持不变。当然，由于C的边与A和B的交的边相交的数量可能比直接地与A和B的所有边相交的数量来得小，先计算A与B的交，再用得到的结果与C求交可能会比直接求A、B和C的交效率更高。 944

19.2 图元的扫描转换

现在我们回到第 3 章讨论过的扫描转换这一主题。每个进行扫描转换的图元都有一个基本几何性质（即它的形状）和若干属性，诸如线型、填充图案及线段连接类型。

对一个给定的物体，判定需要写什么像素的过程与任何裁剪无关，也与像素的写入模式无关。它只依赖于能改变形状的属性及像素选取准则。所以，我们首先讨论属性问题，然后讨论直线和圆锥曲线的几何性质及像素的选取准则。最后以一个一般椭圆的扫描转换算法结束这一节。文本的处理则在讨论过反走样之后。

19.2.1 属性

正如第 3 章描述的那样，图元按一定的属性进行绘制的。这些属性包括线型、填充图案、线宽、端点风格以及线段连接风格。既可以把它们当作修饰属性也可以当作几何属性来看待。例如，如果一条线段以 4 像素亮、1 像素灭的线型绘出，当以比例因子2进行放大时，是用8像素亮、2像素灭的线型来绘制吗？如果回答是，那么就是把线型当成几何属性看待，否则是当成修饰属性来处理了。这里，我们把直线的所有属性作为修饰属性来处理。由于曲线、折线、圆和椭圆也是无限细的形状，我们同样把它们的所有属性当成修饰属性看待。那么矩形、填充的圆以及其他一些带面积的区域怎么办呢？根据第 3 章所讨论，当把一种填充图案应用于图元时，可以有三种选择：把它当作图元、任意的点，还可以看作是位图，而图元则画在位图内。对于第一种情形，属性是几何的，第二种情形可以是几何的也可以是修饰的，第三种情形则是修饰的。不过第一种情形也不总是几何的，在大多数系统中图元旋转时填充图案并不旋转。另一方面，在一些向量系统中，交叉阴影（一种填充图案）是基于图元本身的坐标系来做的，因此旋转图元时交叉阴影填充图案也就自然发生旋转，这类系统中，交叉阴影属性总是几何的。

属性是修饰的还是几何的，像素位于网格中心还是在网格交叉点上，以及窗口是否包含边界，所有这些构成了一个图形系统的*参考模型*。一个参考模型是决定图形系统语义的所有规则的集合，用以排除描述时所有可能出现的二义性。首先应在抽象的层次对它进行设计，然后再嵌入到实际的算法中。在构造扫描转换算法时，牢记参考模型是很重要的；只有清晰地定义好参考模型，才能评估算法成功和正确与否。 945

19.2.2 评价扫描转换算法的准则

第3章绘制一条线段时，我们总是画最靠近线段的那些像素。对于斜率大于 1 的线段，每行画一个像素，从线段经过的两个像素中选取；当然，如果在某一行，线段只经过一个像素，那么选取此像素即可。从一个像素到线段的距离有两种度量方式：沿网栅线方向的距离和沿垂直方向的距离。图19-25表明，根据相似三角形的性质，这两种距离成正比，因此选择哪种度量方式没有什么关系。

3.17.4节中推导Gutpa-Sproull反走样算法时，我们发现，如果直线方程为$Ax + By + C = 0$，那么点(x, y)到直线的距离与$F(x, y) = Ax + By + C$成正比。有时函数值$F(x, y)$被称为点(x, y)的*残差*，而直线本身则可以认为是由Euclidean平面上残差为0的点构成的集合。因而残差也可以用来作为到直线距离的度量，这种距离产生的像素选择与垂直距离和网格距离的是一样的。

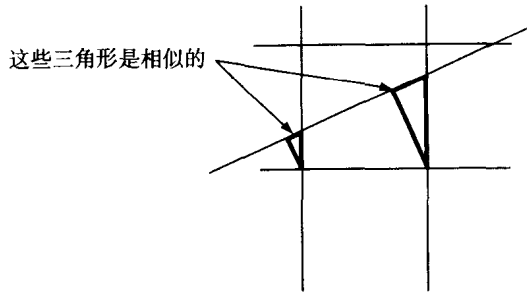


图19-25 根据三角形相似性，点到直线的网格距离和垂直距离成正比

对于圆，我们同样可以用栅格距离、垂直距离或残差值来选择像素。McIlroy已经证明[MCIL83]，如果圆心和半径（甚或半径的平方）都是整数，三种距离得到的像素是一致的。但是，如果不满足上述条件，一般说来所选择的像素是不同的，必须仔细考虑选用哪种度量方式。（对圆心或半径为半整数的圆，选择不是惟一的，因为在不连续状态下作出的选择可能不一致，不过McIlroy证明，无论哪种距离度量，总能保证只有一个“最近的”像素。）

对于椭圆，情况就更为复杂。同样，仍有三种度量方式判定像素是否位于椭圆上，即栅格距离、垂直距离和残差值。虽然栅格距离是明确定义的，但它可能导致一些特异的像素选取。例如，对于图19-26a中的细长、倾斜的椭圆，像素A和B都是与椭圆上边相交的栅格线的两个端点。不幸的是椭圆的下边也与这一栅格线相交。采用栅格距离进行度量时，B到椭圆的上下两边的距离都比A更近。如果对上下两边进行扫描转换时都选择像素B，那么一个像素将被写两次。点到椭圆的垂直距离同样是明确定义的（就是这一点到椭圆上所有点的距离的最小值），但这个点可能与椭圆上的几个点相近。例如，对图19-26b中的椭圆的底边进行扫描转换时，尽管像素A比像素B更靠近椭圆，我们还是倾向于选择像素B，因为A所靠近的部分与B所靠近的部分无关。这种情况下，残差值方法也导致同样（错误）的选取——像素A处的残差值比像素B处的残差值小。由此看来，如何选择靠近椭圆的度量没有简单的答案。对弯曲较厉害的椭圆，所有的距离度量都不成功。

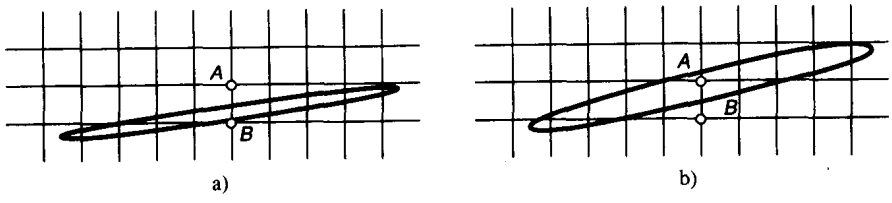


图19-26 用不同准则选择椭圆点。a)对上下两条边进行扫描转换时都会选到B——它应该被写两次吗？b)对椭圆的底边进行扫描转换时，采用栅格距离和残差值方法都会选中A，尽管明显地B是更好的选择

采用残差作为误差度量的困难部分是隐含在残差的本质之中。例如，对于圆来说，半径为R圆心为原点的圆由满足$x^2 + y^2 - R^2 = 0$的所有点(x, y)组成，因此$x^2 + y^2 - R^2$可以作为点(x, y)远离这个圆的程度的一个合理度量。但是，上面的圆也可以定义成满足$(x^2 + y^2)^{1/2} - R = 0$的点集，因而$(x^2 + y^2)^{1/2} - R$也可以用来计算残差（但它们与前面定义的残差是不同的，甚至也不满足正比关系），此外对任意正数a，$(x^2 + y^2)^a - R^{2a}$都可以定义为残差。可见残差的是一种比较随意的度量方法。

最后，对于一般曲线，没有明确的选择。我们仍然发现中点准则较为可信：在曲线近似于

水平的地方，通过判断竖直方向上的两个相邻像素的中点位于曲线的哪一侧来进行选取（这实际就是栅格距离准则），后面我们将讨论一个基于这一准则的算法。

在设计一个扫描转换算法时，选择一种误差度量方式然后最小化这一误差是非常重要的。只有这样的度量选定了，算法才可能被证明正确与否。此外，指定进行逼近时所用到的点同样很重要。对于以线段逼近曲线然后对直线段进行扫描转换的算法，如果用点到直线的距离作为误差度量的话，得到误差可能很小，但用点到曲线的距离作为误差度量则可能导致很大的误差。 946 ~ 947

19.2.3 直线的其他考察方式

第2章我们只讨论了那些在光栅系统中易于描述的线段，即端点坐标为整数的直线段。对于这些线段，我们给出了一个增量算法（中点算法）来计算最靠近直线段的像素。对于每个整数x值，通过检查一个判定变量来在两个整数y值中进行选择，然后给判定变量加上一个增量供处理下一个x值时使用。由于直线方程$(Ax + By + C = 0)$的系数都是整数而且线段的起始点(x_0, x_0)满足直线方程（即$Ax_0 + By_0 + C = 0$），所以初始化这个判定变量是很容易的。

但并不是所有线段的端点坐标都是整数，我们怎么在光栅设备上画端点坐标为实数的直线段呢？假设要在一个黑白光栅设备上画一条从(0,0)到(10,0.51)的直线段。很显然，(0,0)到(9,0)及(10,1)是应画的像素。但是如果我们采用简单地对端点进行四舍五入得到整数端点再绘制，则选取的像素是(0,0)到(5,0)及(6,1)到(10,1)，与前者完全不同。不过我们可以计算出这条直线的方程：

$$1.0y - 0.051x + 0.0 = 0.0$$

并且应用中点算法来画这条直线段。直接利用上面的方程需要一个浮点版本的中点算法，但代价太大。另一种可能性是，注意到给直线方程两边乘以1000时，从而获得一个整系数的直线方程：

$$1000y - 51x + 0 = 0$$

用这个方程来进行中点算法。一般地，可以先把浮点数转换为定点数，然后通过乘以一个适当大小的整数而得到直线的整系数方程[⊖]。大多数情况下，乘数决定了直线段端点的子像素分辨率：如果希望把端点放在1/4整数位置，就必须乘4；对1/10整数位置，则应乘以10。

然而，当要画的第一个像素不是正好在直线段上时，又会出现另一个问题。考虑从(0, 0.001)到(10, 1.001)的直线段。它的方程为

$$y - 0.1x - 0.001 = 0$$

或

$$1000y - 100x - 1 = 0$$

第一个要画的像素是(0, 0)，显然不在直线段上。在这种情形下，需要明确地计算起始点处的残差值$Ax + By + C$来对判定变量进行初始化，在这里原始算法的任何简化都是不可能的。这种情形下选择起始点也是一个问题。如果实际端点为(x_0, y_0)，我们必须选择靠近(x_0, y_0)的整数点。对斜率小于1的直线段，可以对x_0四舍五入得到一个整数起始x值，然后计算y值使(x, y)在直线上，再对y四舍五入（见习题19.6）。 948

具有非整数端点直线的提出是很自然的，即使在整数世界里也是如此。正如我们在第 3 章已经看到的，这些直线段经常在裁剪之后产生。假设有一条从(0, 0)到(10, 5)的直线段，希望画

⊖ 必须小心避免乘一个太大的整数而导致溢出。

出位于$x=3$和$x=7$之间的条带内的部分。该段的两个端点是非整数坐标$\left(3,\frac{3}{2}\right)$和$\left(7,\frac{7}{2}\right)$。需要注意，在对这一线段进行扫描转换时，应采用原来直线段的方程，以免舍入误差改变直线的形状（见习题19.7）。

总之，对任给的直线段，可以把它的端点坐标化为分母固定的有理数，然后求出直线段的整系数方程，再用第3章的算法进行扫描转换（在此之前需要对判定变量进行明确的初始化）。

19.2.4 高级折线算法

生成光栅化折线（未填充的多边形）时，基本上是重复画线算法。不过这里出现了参考模型的问题。考虑一条如图19-27所示的具有尖点的折线。对位于上面的边进行扫描转换时，产生图中的水平阴影线像素，对位于下面的边扫描转换时产生竖直阴影线像素。有交叉阴影线的像素则两次都画到了。如果以**xor**模式绘折线，则两次都画到的像素将被异或两次，这可能不是我们所需要的。有两种可能的解决办法：先以**replace**模式把图元画到屏幕画面外的位图中，然后用**xor**模式把得到的位图拷贝到屏幕，或者建立一个表示整个图元的数据结构，再把数据结构绘制到位图中。19.7节的形状数据结构为此提供了手段（而且还能适应具有图案的直线和折线）。

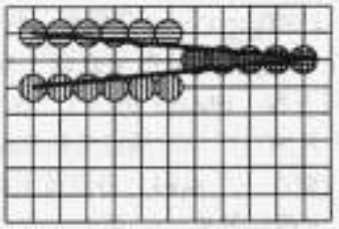


图19-27 具有尖点的折线可能导致某些像素被画多次

19.2.5 画圆算法的改进

我们有一个生成圆的快速算法（中点算法）。在第3章我们还讨论了用这个算法画具有宽度和填充属性的圆。但是那里给出的算法只处理半径和圆心都是整数的情形。当半径和圆心不是整数时，原来的算法的对称性将不再适用，每个八分圆都必须单独画出来，如图19-28所示。

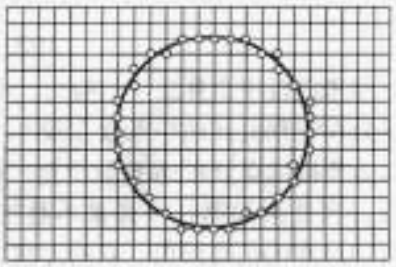


图19-28 圆心和半径都不是整数的圆，它的八分圆互不相同，因此无法用对称性来加速扫描转换过程

但是，并不是任意的圆（半径和圆心都是实数）都能转化为整系数圆锥曲线，如果圆的半径和圆心限制在具有特别分母的有理数范围内[⊖]，我们可以在计算（以浮点方式）圆的方程系数之后，乘上公分母的四倍并对系数进行四舍五入。把中点算法应用于所得的整系数方程所产生的点与把算法应用于原始的浮点系数方程所得的点几乎相同。为清楚起见，假设圆心为$(h/p, k/p)$，半径为R，则圆的方程为

$$S(x, y) = \left(x - \frac{h}{p}\right)^2 + \left(y - \frac{k}{p}\right)^2 - R^2 = 0$$

乘以$4p$后得

$$4pS(x, y) = 4px^2 - 8hx + 4py^2 - 8ky + 4\left(\frac{h^2}{p} + \frac{k^2}{p} - pR^2\right) = 0$$

⊖ 我们选择使所有的数都有相同的分母以使得代数更简单。因为任意两个分数都可以通分，使具有公分母，这一限制不算太苛刻。

回顾一下，在中点算法中我们只是检查判定变量的符号。对上式中的圆括号内的项（惟一的非整数项）进行四舍五入，用一个整数来代替它，对$4pS(x, y)$的改变小于1。因为我们只在x和y有一个是整数而另一个是整数加1/2处估算S的值，$4pS(x, y)$表达式的前面几项都是整数，一个整数作小于1的变化不会改变它的符号，除非这个整数是0[⊖]。因此可以使用四舍五入后的$4pS(x, y)$作为中点算法的判定变量；惟一的后果是，当圆通过两个像素的中点整数和浮点两个版本的算法可能做出相反的选择。

949 ~ 950

偏差分与以前一样，只不过这里要乘以一个常量$4p$。所以，每次不是递增2，而是递增$8p$。初始化就更复杂了。以$(0, R)$加上圆心坐标得到的不再是整数点，所以必须在此处选择一个邻近点开始画圆。为了画右上方的八分圆，应从圆的最高点或它右边的点开始。我们通过求大于h/p的最小整数获得x坐标，但必须明确地计算y的坐标和判定变量的初始值。这一步在算法中只进行一次，所以代价不算太大。图19-29列出的代码只生成八分之一圆，要画完整个圆还需要添上七个类似的代码段。

```
void MidpointEighthGeneralCircle (
        int h, k;                          /* 中心x, y坐标的分子 */
        int p,                             /* 中心x, y坐标的分母 */
        double radius)                     /* 圆的半径 */
{
    int x, y,                              /* 所画的上一个点 */
        d,                                 /* 判定变量 */
        A,D,E,F,                           /* 方程的系数 */
        A2, A4,                            /* 为提高效率而设的A的倍数 */
        deltaE,                            /* d(x + 2,y - 1/2) - d(x + 1, y - 1/2) */
        deltaSE;                           /* d(x + 2,y - 3/2) - d(x + 1, y - 1/2) */
    double temp;

    /* 初始化x, y坐标和差分 */
    A = 4 * p;
    A2 = 2 * A;
    A4 = 4 * A;
    D = -8 * h;
    E = -8 * k;
    temp = 4 * (-p * radius * radius + (h*h + k*k) / (double)p);
    F = round (temp);                      /* 引入误差小于1 */
    x = ceil (h / (double)p);              /* 最小的整数>=h/p */
    y = round (sqrt (radius*radius - (x - h/(double)p) * (x- h/(double)p)) +
        k/(double)p);
    /* 下一条线必须使用实数算术计算，而不是整数； */
    /* 为了避免这样的要求它必须被重新写 */
    d = round (A * ((x + 1.0) * (x + 1.0) + (y - 0.5) * (y - 0.5)) +
              D * (x + 1.0) + E * (y - 0.5) + F);
    deltaE = A2 * (x + 1) + A + D;
    deltaSE = A2 * (x - y) + 5 * A + D - E;
    DrawPixel (x, y);
    while ((p * y - k) > (p * x - h)) {    /* 在这个八分圆内循环 */
        if (d < 0) {                       /* 选择E像素 */
            d += deltaE;
```

图19-29 一般中点圆算法

⊖ 在设计中点算法时，我们必须决定0是正还是负。加一个较小的数到0上，可以改变这一符号选择。

```
            deltaE += A2;
            deltaSE += A2;
            x++;
        } else {
            d += deltaSE;                    /* 选择SE像素 */
            deltaE += A2;
            deltaSE += A4;
            x++;
            y--;
        }
        DrawPixel (x, y);
    } /* while */
} /* MidpointEighthGeneralCircle */
```

图19-29（续）

19.2.6 一般圆锥曲线算法

第3章给出了一个长短轴与坐标轴平行的椭圆扫描转换算法。本节我们介绍一个由Van Aken［VANA89］提出的一般圆锥曲线扫描转换算法，圆锥曲线包括长短轴倾斜的椭圆、双曲线、圆和抛物线。这个算法基于Pitteway的曲线跟踪算法，Pitteway算法于1967年发表［PITT67］，仅比Bresenham引入增量算法晚两年。那时Pitteway算法没受到太多的关注，它的主要内容已经被重复发现好几次了。

圆锥曲线算法由两个独立的部分组成：圆锥曲线描述和扫描转换。由于一般圆锥曲线描述为具有下述形式的方程的解：

$$S(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

它可由六个系数A,B,C,D,E和F指定。但这些系数很不直观，因此考虑另一种方案。这里只讨论椭圆，类似的技术可用于双曲线和抛物线（参见习题19.9）。

平面上的圆可以很好地拟合一个单位正方形（见图19-30a）。如果在此平面上做一个仿射变换（即齐次坐标下的线性变换），正方形将被变成平行四边形而圆被变成椭圆，如图19-30b所示。这是椭圆的SRGP描述的一种推广，在SRGP描述中水平或竖直矩形被用作水平或竖直椭圆的包围盒。平行四边形边的中点位于椭圆上。通过指定这些中点及平行四边形的中心，平行四边形就被惟一确定，从而椭圆也被惟一确定。所以我们用平行四边形的中心J及两边的中点P和Q来给定椭圆。容易看到，如果能够确定中心J为原点时的系数，那么也能够处理一般问题：只须将更简单的情形用于$P' = P - J$和$Q' = Q - J$、扫描转换以及把J的坐标加到输出像素再绘制即可（当然，这只是在J的坐标为整数时才可行）。所以我们假定J就是原点，而P和Q也相应地被调整。另外还假定按顺时针方向从P到Q时走过的是椭圆的短弧。

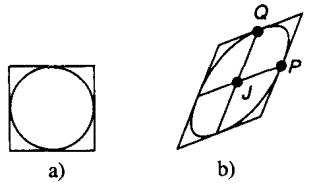


图19-30 a)一个圆的包围正方形；b)变换后的正方形和它包围的椭圆

为找到椭圆的方程，我们首先求出把点$\begin{bmatrix}1\\0\end{bmatrix}$和$\begin{bmatrix}0\\1\end{bmatrix}$变成$P$和$Q$的变换（它可由第一列为$P$且第二列为$Q$的矩阵给出）。这个变换把单位圆上形如$\cos(t)\begin{bmatrix}1\\0\end{bmatrix} + \sin(t)\begin{bmatrix}0\\1\end{bmatrix}$的点变成椭圆上形为$\begin{bmatrix}x\\y\end{bmatrix} = \cos(t)P + \sin(t)Q$的点。从这个方程求出$\cos(t)$和$\sin(t)$，它们都是关于$x$和$y$的线性表达式。再把

这些线性表达式代入到恒等式$\cos^2(t) + \sin^2(t) = 1$中，就得到关于$x$和$y$的二次表达式，这就是所需的椭圆方程（参见习题19.8）。如果P的两个坐标分量分别记为P_x和P_y，类似地Q的分量为Q_x和Q_y，最终得到的方程的系数是：

$$A = P_y^2 + Q_y^2, \quad B = -2(P_xP_y + Q_xQ_y), \quad C = P_x^2 + Q_x^2$$
$$D = 0, \quad E = 0, \quad F = -(P_xQ_y - P_yQ_x)^2$$

现在我们把得到的椭圆平移到原点为$-P$的新坐标系下；也就是说，我们用新方程替换椭圆的方程：

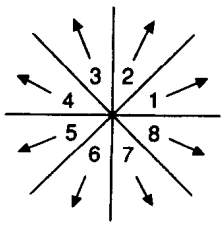
$$A(x + P_x)^2 + B(x + P_x)(y + P_y) + C(y + P_y)^2 + D(x + P_x) + E(y + P_y) + F =$$
$$A'x^2 + B'xy + C'y^2 + D'x + E'y + F' = 0$$

其中，新坐标系下的系数为

$$A' = A, \quad B' = B, \quad C' = C$$
$$D' = 2Q_y(P_xQ_y - P_yQ_x) \quad E' = -2Q_x(P_xQ_y - P_yQ_x) \quad F' = 0$$

此时坐标原点位于新的圆锥曲线上，如果对这个圆锥曲线进行扫描转换，但画点时先对每个点加上坐标(P_x, P_y)，就可以得到原来中心位于原点的圆锥曲线上的点。因为A, B和C没有改变，我们仍用D, E记D'和E'（因为原来的D, E都为0）。

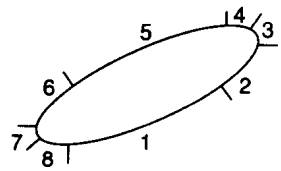
至此，已导出了椭圆方程的系数，下面需要对它进行扫描转换。我们把扫描转换过程分成*八个画八分弧*（对于第3章中圆的生成算法，是对应于八分圆；但这里不是）。画八分弧确定算法跟踪的方向。在第一个八分弧里，可选择方向是向右移动和沿对角线向右上方移动。可做的移动又分为两种，即*平直移动*和*对角移动*，这取决于是一个坐标分量改变还是两个坐标分量都改变。图19-31列出了八个八分弧的运动方向及其在一个典型的椭圆上对应的弧段。



a)

八分弧	平直运动		对角运动	
	ΔX	ΔY	ΔX	ΔY
1	1	0	1	1
2	0	1	1	1
3	0	1	−1	1
4	−1	0	−1	1
5	−1	0	−1	−1
6	0	−1	−1	−1
7	0	−1	1	−1
8	1	0	1	−1

b)



c)

图19-31 a) 八个画八分弧，b)对应的运动方向，c)一个典型椭圆

我们的算法有两个不同的循环——一个用来跟踪奇数序号的八分弧，到达一个对角八分弧边界时停止；另一个则是用来跟踪偶数序号的八分弧，到达一个平直八分弧边界时停止。为确定起始八分弧，注意到在由方程$S(x, y) = 0$给定的圆锥曲线的任意一点上，$S(x, y)$的梯度是$(2Ax + By + D, Bx + 2Cy + E)$，与圆锥曲线垂直（这在第3章的算法中用来确定椭圆的八分弧）。我们使用梯度的分量来确定运动的方向（让它绕逆时针方向旋转90°）从而画八分弧。因为起始点为$(0, 0)$，此处梯度为(D, E)。图19-32所示的

```
int GetOctant (int D, int E)
{
    if (D > 0 && E < 0)
        return (D < -E) ? 1 : 2;
    if ...          /* 处理其余6种情况
} /* GetOctant */
```

图19-32 由梯度分量确定画八分弧的函数

代码段说明了八分弧划分的过程。

现在需要对每个八分弧确定其判定变量d的值、判定变量的含义以及如何对它进行修改。沿对角方向移动时，给d加上增量v，而沿平直运动时增量记为u。如果$S(x, y) = 0$是椭圆的方程，那么d被定义为下两个待选像素的中点处的函数值S。由于$S(x, y)<0$为椭圆内部点，而$S(x, y)>0$为外部点，d的值为负说明上述中点位于椭圆内，因此应该选择外面的像素；d的值为正时，选择内部的像素；d是零时两个像素任选一个——Van Aken的选择是（对奇数序号的八分弧），d为负数时，做平直运动，而其他情况一律做对角运动，对偶数序号八分圆，则当d为负数时，做对角运动，其他一律做平直运动。

在第一个八分弧中，如果已画像素(x_i, y_i)，记d_i为选取下一个像素的判定变量，待选像素为$(x_i + 1, y_i)$和$(x_i + 1, y_i + 1)$。再以u_{i+1}和v_{i+1}表示从d_i求d_{i+1}时用到的增量。那么对每个像素需要做的工作是：(1) 画像素；(2) 根据d_i的值选择下一个要画的像素；(3) 根据所选的像素，修改u_i和v_i的值得到u_{i+1}和v_{i+1}；(4) 通过对d_i加上增量u_{i+1}或v_{i+1}得到d_{i+1}；(5) 检查八分弧是否改变。

951
~
954

回忆一下d_{i+1}可通过差分技术从d_i求得。假如在第一个画八分弧刚画完像素(x_i, y_i)，并已根据$d_i = S(x_i + 1, y_i + 1/2)$决定了下一个像素的选取。如果是做平直运动，那么$x_{i+1} = x_i + 1, y_{i+1} = y_i$。新的判定变量$d_{i+1}$的值为$S(x_i + 2, y_i + 1/2)$，它与$d_i$的差是

$$\begin{aligned}
u_{i+1} &= d_{i+1} - d_i \\
&= A(x_i + 2)^2 + B(x_i + 2)(y_i + \tfrac{1}{2}) + C(y_i + \tfrac{1}{2})^2 \\
&\quad + D(x_i + 2) + E(y_i + \tfrac{1}{2}) + F \\
&\quad - [A(x_i + 1)^2 + B(x_i + 1)(y_i + \tfrac{1}{2}) + C(y_i + \tfrac{1}{2})^2 \\
&\quad + D(x_i + 1) + E(y_i + \tfrac{1}{2}) + F] \\
&= A[2(x_i + 1) + 1] + B(y_i + \tfrac{1}{2}) + D \\
&= A(2(x_i) + 1) + B(y_i + \tfrac{1}{2}) + D + 2A
\end{aligned}$$

另一方面，如果是做对角移动，那么d_{i+1}[⊖]的值是$S(x_i + 2, y_i + 3/2)$，因而增量为

$$\begin{aligned}
v_{i+1} &= d_{i+1} - d_i \\
&= (2A + B)x_{i+1} + (B + 2C)y_{i+1} + A + B/2 + D + E \\
&= (2A + B)x_i + (B + 2C)y_i + A + B/2 + D + E + [2A + 2B + 2C]
\end{aligned}$$

如果置$u_i = A[2(x_i) + 1] + B(y_i + 1/2) + D$，那么对平直移动有$u_{i+1} = u_i + 2A$。类似地，如果$v_i = (2A + B)x_i + (B + 2C)y_i + A + B/2 + D + E$，则对于对角移动有$v_{i+1} = v_i + (2A + 2B + 2C)$。为了保证无论做平直还是做对角运动，$u_i$和$v_i$的值都正确，即使在它们不被用到时也应进行修改。这样，对于平直移动，

$$\begin{aligned}
v_{i+1} &= (2A + B)x_{i+1} + (B + 2C)y_{i+1} + A + B/2 + D + E \\
&= (2A + B)(x_i + 1) + (B + 2C)y_i + A + B/2 + D + E \\
&= v_i + (2A + B)
\end{aligned}$$

对于对角移动，$u_{i+1} = u_i + 2A + B$。希望读者用表格的形式清楚地列出对每个八分圆的平直和对角运动增量u_i和v_i的变化情况（见习题19.10）。代数运算虽然繁琐但却很有用。

⊖ 原书为d_i。——译者注

如果$k_1 = 2A, k_2 = 2A + B, k_3 = 2A + 2B + 2C$，那么$u$和$v$的修改过程可用下列规则描述：

平直运动：

$$u_{i+1} = u_i + k_1, \qquad v_{i+1} = v_i + k_2$$

对角运动：

$$u_{i+1} = u_i + k_2, \qquad v_{i+1} = v_i + k_3$$

955

下面我们来看看如何判断八分弧的改变。容易看出，当梯度向量指向右下角方向，也就是(1, - 1)的倍数时，将离开画第一个八分弧。换言之，当梯度向量的两分量的和从负数变成零时离开第一个八分弧（见图19-33）。

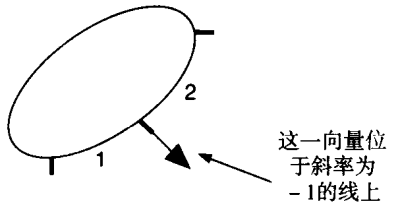


图19-33 画第一八分弧时，梯度分量的和为负。进入第二八分弧后，和变为正

现在来看梯度向量的分量，

$$\left(\frac{\partial S}{\partial x}, \frac{\partial S}{\partial y}\right) = (2Ax + By + D, Bx + 2Cy + E)$$

可以写成前面给出的u_i和v_i的表达式形式：

$$\frac{\partial S}{\partial x} = u_i - \frac{k_2}{2}, \qquad \frac{\partial S}{\partial x} + \frac{\partial S}{\partial y} = v_i - \frac{k_2}{2}$$

所以可以通过查看$v_i - k_2/2$的符号来检测什么时候离开画第一个八分弧。要看是否离开画第二个八分弧可检查$u_i - k_2/2$的符号。

但是，在可以写出实际代码之前，我们还需要考虑最后一个问题。当从画第一个八分弧过渡到画第二个八分弧时，d_i的定义将发生改变，u_i和v_i也是如此。尽管u_i和v_i仍分别对应做平直运动和对角运动时的增量，但现在的平直运动已是竖直运动而不是水平运动，判定变量的修改值不再是$S(x_i + 1, y_i + 1/2)$而是$S(x_i + 1/2, y_i + 1)$。用带撇的记号表示第二个八分弧的数据，不带撇的记号表示第一八分弧的数据，则有

$$\begin{aligned} d'_i - d_i &= S\left(x_i + \frac{1}{2}, y_i + 1\right) - S\left(x_i + 1, y_i + \frac{1}{2}\right) \\ &= \frac{v_i}{2} - u_i + \frac{3}{8}k_3 - \frac{1}{2}k_2 \end{aligned}$$

$$\begin{aligned} v'_i - v_i &= [(2A + B)x_i + (B + 2C)y_i + \frac{B}{2} + C + D + E] \\ &\quad - [(2A + B)x_i + (B + 2C)y_i + A + \frac{B}{2} + D + E] \\ &= -A + C \end{aligned}$$

$$u'_i - u_i = v_i - u_i - \frac{k_2}{2} + \frac{k_3}{2}$$

956

导出第一个和第三个方程的计算尽管很直接但较长。此外，用于计算u_i和v_i增量的变量也随着改变。假设u_i和v_i的增量表已经做出，很显然有$k'_3 = k_3, k'_2 = k_3 - k_2$的$k'_1 = k_1 - 2k_2 + k_3$。

到此我们已准备好了完成算法的所有工具，至少对两个八分圆是如此。我们在算法的代码中包含了系数F（圆锥曲线的常数项），尽管假设它的值为零。对于中心不是整数的更一般的圆锥曲线，算法中用到的起始点不一定恰好位于曲线上，此时F可能不为零。我们把这种情形的实验留给读者。

图19-34中的算法代码之后，列出了一个名为Conjugate的过程，用来确定由两共轭直径端点指定的椭圆（中心在原点）方程的系数。如果端点为P和Q，那么包围椭圆的平行四边形的顶点是$P+Q$，$P-Q$，$-P-Q$和$-P+Q$；椭圆与平行四边形在边的中点处相切（如图19-30所示）。

```
/* 画圆锥上点(xs, ys)和(xe, ye)之间的二次曲线的弧线，该二次曲线 */
/* 是由Ax² + Bxy + Cy² + Dx + Ey + F = 0给出的。如果该二次曲线 */
/* 是一条双曲线，这两个点必须位于同一分支上。 */
void Conic (
        int xs, ys,                          /* 起始点 */
        int xe, ye,                          /* 终止点 */
        int A, B, C, D, E, F)                /* 系数 */
{
    int x, y;                                /* 当前点 */
    int octant;                              /* 当前八分弧 */
    int dxsquare, dysquare;                  /* 平直运动(x, y)的变化 */
    int dxdiag, dydiag;                      /* 对角运动(x, y)的变化 */
    int d,u,v,k1,k2,k3;                      /* 判定变量和梯度 */
    int dSdx, dSdy;                          /* 梯度分量 */
    int octantCount;                         /* 被画的八分弧数 */
    int tmp;                                 /* 用于完成交换 */

    octant = GetOctant (D, E);               /* 起始八分弧序号 */
    switch (octant) {
        case 1:
            d = round (A + B * 0.5 + C * 0.25 + D + E * 0.5 + F);
            u = round (A + B * 0.5 + D);
            v = round (A + B * 0.5 + D + E);
            k1 = 2 * A;
            k2 = 2 * A + B;
            k3 = k2 + B + 2 * C;
            dxsquare = 1;
            dysquare = 0;
            dxdiag = 1;
            dydiag = 1;
            break;
        case 2:
            d = round (A * 0.25 + B * 0.5 + C + D * 0.5 + E + F);
            u = round (B * 0.5 + C + E);
            v = round (B * 0.5 + C + D + E);
            k1 = 2 * C;
            k2 = B + 2 * C;
            k3 = 2 * A + 2 * B + 2 * C;
            dxsquare = 0;
            dysquare = 1;
            dxdiag = 1;
            dydiag = 1;
            break;
        ... 其他6种情况 ...
    }  /* switch */

    x = xe - xs;                             /* 平移(xs, ys)至原点 */
    y = ye - ys;
```

图19-34 一般椭圆算法

```
dSdx = 2 * A * x + B * y + D;          /* 端点的梯度 */
dSdy = B * x + 2 * C * y + E;
/* 这确定结束八分弧 */
octantCount = GetOctant (dSdx, dSdy) - octant;
if (octantCount <= 0) octantCount += 8;

/* 现在实际画该曲线 */
x = xs;
y = ys;
while (octantCount > 0) {
    if (octant & 1) {
        while (v <= k2 * 0.5) {
            DrawPixel (x, y);
            if (d < 0) {
                x += dxsquare;
                y += dysquare;
                u += k1;
                v += k2;
                d += u;
            } else {
                x += dxdiag;
                y += dydiag;
                u += k2;
                v += k3;
                d += v;
            }
        }  /* while v <= k2 * 0.5 */
        /* 现在穿过对角八分弧边界 */
        d = round (d - u - v * 0.5 - k2 * 0.5 + 3 * k3 * 0.125);
        u = round (-u + v - k2 * 0.5 + k3 * 0.5);
        v = round (v - k2 + k3 * 0.5);
        k1 = k1 - 2 * k2 + k3;
        k2 = k3 - k2;
        tmp = dxsquare;
        dxsquare = -dysquare;
        dysquare = tmp;
    } else {      /* 八分弧是偶数 */
        while (u < k2 * 0.5) {
            DrawPixel (x, y);
            if (d < 0) {
                x += dxdiag;
                y += dydiag;
                u += k2;
                v += k3;
                d += v;
            } else {
                x += dxsquare;
                y += dysquare;
                u += k1;
                v += k2;
                d += u;
            }
```

图19-34（续）

```
            } /* while u < k2 * 0.5 */
            /* 现在穿过平直八分弧边界 */
            d += u - v + k1 - k2;          /* 先求v，它与u有关 */
            v = 2 * u - v + k1 - k2;
            u += k1 - k2;
            k3 += 4 * (k1 - k2);
            k2 = 2 * k1 - k2;
            tmp = dxdiag;
            dxdiag = - dydiag;
            dydiag = tmp;
        } /* 八分弧是偶数 */
        octant++;
        if (octant > 8) octant -= 8;
        octantCount--;
    } /* while octantCount > 0 */
    输入最后一个八分弧，继续画，直到达到最后一个像素;
} /* Conic */

/* 根据椭圆的共轭直径的端点P = (xp, yp)和Q(xq, yq) */
/* 来定义并画出椭圆。该端点是按相对于 */
/* 椭圆中心的偏移指定的， */
/* 这里假设椭圆中心是原点。 */
void Conjugate (int xp, int yp, int xq, int yq, int mode)
{
    int xprod, tmp, xe, ye, A, B, C, D, E, F;

    xprod = xp * yq - xq * yp;

    if (xprod != 0) {                    /* 如果它是0, 点共线！ */
        if (xprod < 0) {
            tmp = xp; xp = xq; xq = tmp;
            tmp = yp; yp = yq; yq = tmp;
            xprod = -xprod;
        }
        A = yp * yp + yq * yq;
        B = -2 * (xp *yp + xq * yq);
        C = xp * xp + xq * xq;
        D = 2 * yq + xprod;
        E = -2 * xq * xprod;
        F = 0;

        if (mode == FULL_ELLIPSE) {      /* 设置起始点和终止点相等 */
            xe = xp; ye = yp;
        } else {      /* mode == ELLIPSE_ARC; /* 只画P和Q之间的弧 */
            xe = xq; ye = yq;
        }
            Conic (xp, yp, xe, ye, A, B, C, D, E, F);
        } /* if */
    } /* Conjugate */
```

图19-34（续）

代码以下列操作作为结束：进入最后一个八分弧，统计为达到最后一个像素所需的平直方向的步数，继续算法直到所有步骤都完成之后。我们把这一步的细节留给读者（见习题19.13）。

令人惊奇的是，前两个八分弧变化期间用来修改各种增量的代码对所有其他八分弧都适用。

有关画椭圆还留下最后一个问题。如图19-35所示，有时算法中的一个对角步所选取的绘画点明显跨到椭圆的另一边——从而立刻改变几个八分弧。这种情形出现时，算法将崩溃，并且像图中所示一样沿远离椭圆的方向继续前进。显然这是走样的一种形式——我们所采样的信号$S(x, y)$包含太高的频率，通过整数栅格点采样无法分解。

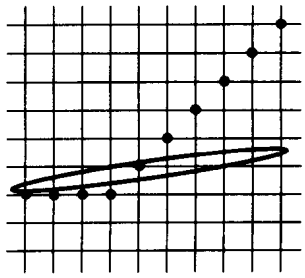


图19-35 对一个狭长的椭圆，算法崩溃

Pratt在[PRAT85]中提出了解决这个问题的一种方法。算法画第一八分弧的过程中正在跟踪像素时，如果出现跨过椭圆的一次跳越，梯度方向将发生剧烈变化。事实上，画第一八分弧时，梯度方向的y分量总是负的而x分量总是正的。如果跨越八分弧之后，到达第3,4,5或6八分弧上的某一点。下面来确定这些八分弧与八分弧7,8,1和2之间的分界线。置梯度向量的x分量为零，即令$2Ax + By + D = 0$。因为$u_i = 2Ax_i + By_i + D + A + B/2$，所以$u_i$可用来检查是否出现跨越。对每个八分弧都可得到类似的检查方法。注意，对每个八分弧，只有在一种运动类型期间需要进行这些检查：一种运动远离椭圆，因此不需要做检查，另一种运动走近椭圆才需要做检查。有关圆锥曲线跟踪算法的进一步的内容，可参见习题19.14和参考文献[PITT67；PRAT85；DASI89]。

Wu和Rokne[WU89]提出了圆锥曲线的另一类算法，但只适合于灰度级显示。它不是一次扫描转换一个像素而是一个像素块。假设正沿水平方向前进，而曲率向上凹。如果已画像素(x, y)，且两步或更多几步之后画$(x + 2, y + 2)$，那么中间插入的像素一定是$(x + 1, y + 1)$。类似地，如果两步或更多几步之后画$(x + 2, y)$，则中间插入的像素为$(x + 1, y)$。只有两步之后画$(x + 2, y + 1)$时存在二义性，此时中间像素可能是$(x + 1, y)$也可能是$(x + 1, y + 1)$。但在灰度级显示时，我们可以简单地用一半的亮度画这两个像素。因此一次可以前进两个像素，大大地减少了计算量。在八分弧的连接处，必须对算法进行调整，以避免重复，不过这并不是很困难。显然，算法只适用于灰度级显示，但它给出了反走样变得可能时如何对算法进行简化的一个例子。我们在文本的讨论中将再次见到。

19.2.7 宽图元

这一节我们准备讨论宽图元：宽直线、折线和加宽的一般曲线（宽直线、圆和椭圆弧第3章已经讨论，这节我们进一步扩展和完善那里提出的思想）。对宽图元进行扫描转换，第一步是要确定图元中什么实际作为几何对象处理——就是说为带宽度的对象定义好参考模型。例如（见图19-36），一条宽直线段看起来是像一个矩形、两端为半圆的矩形还是别的什么？

如果我们想像宽直线段由一支宽头笔画成，上述问题等于要问这支笔的笔尖是扁的还是圆的。从数学的角度看，则可描述为如下二者之一：到中心线距离小于线宽二分之一的所有点构成的集合（这包括圆角）或沿法向（垂直方向）的中心线到中心线距离小于线宽二分之一的所有点构成的集合。第三种数学描述是考虑中心线的两份平行拷贝，把它们无限延长，然后分别沿相反方向离开中心线同等距离，最后用另两条经过原线段的端点的线进行截断，这种描述后面将会证明很有用。如果用于截断的两条线



图19-36 一条宽线可能看起来像一个矩形或端部带圆角、斜角。哪一个正确呢？

段与原来线段垂直的话，得到的就是矩形宽线，否则宽线的端点将是倾斜的（见图19-37）。这些倾斜的端点可用来生成宽折线；按这种方式生成转角称为*斜面接合*。

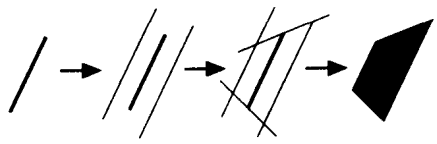


图19-37 通过让两个平行线沿相反方向离开原始线段然后用经过端点的任意直线进行截断，可以定义一类广泛的宽线。这些倾斜的端点通过一种称为斜面接合的技术接在一起，从而得到宽折线

一种特殊的情形把直线段看成是用一支笔尖排列整齐的笔画成，就是说笔尖总是要么水平要么垂直的线段。用这样的笔画出的水平或竖直线看起来像一个矩形，其他情况下则像一个平行四边形。读者可试画几条这样的线以便看看为什么这不是一种画宽线的好方法。我们之所以提到这种方法，是因为它能够快速实现：只需对中点画线算法稍做修改，一次画几个像素而不是原来的一个即可。水平和竖直线段的连接则需要延长线段超过接合处，把它们加到原始线段上，并画出介于两个超出部分中间的点。

每一类上述的宽线都可以用于生成宽折线，如图19-38所示。粗端线段的连接不理想——在接合处有凹槽（见图19-38a）。圆端的线段连接较好——每个接合点弯曲处的外部是一条光滑的曲线，而内面则是一个尖角（见图19-38b）。斜端线（有相同宽度）在两面的倾斜度都相同时，连接得也比较好，否则两条线段的倾斜边会有不同的长度。为得到相同长度的倾斜边，可选择处于两线段中间位置的截线。如果要接合两条几乎平行的线段，接合点将被充分延长至超过中心线的实际交点（见图19-38d）。有时这种延长是不想要的，因此必须对接合点进行裁剪。所采用的裁剪线是与接合线垂直、与中心线有一定距离的线段（见图19-38e）。

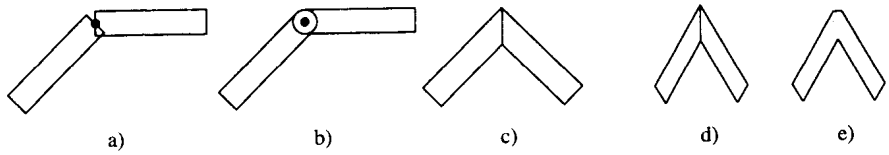


图19-38 各种类型线的连接。a)粗端线段接合效果较差；b)圆端线段接合效果较好，但有弯曲段，对某些应用而言可能是不理想的；c)用斜端线连接可实现斜面接合；d)斜面接合点处角度太小，看起来效果特差；e)对这样的接合进行截断

上述各种接合方法在很多绘图软件中都有提供，这些软件包括PostScript、QuickDraw和X Windows系统。大多数这类图形包不仅提供线段连接风格，同时还提供线端风格，因此，我们可以生成圆端、斜面接合的不封闭折线段。

有几种方法对曲线进行加宽，包括手绘或其他非解析方法得到的曲线。可以用第3章提供的方法把曲线转化成多条短线段，然后对每条短线段进行加宽。如果线段足够短而输出设备的分辨率又足够高，则结果是适宜的。重要的是不要对短线段的端点进行四舍五入取整（尽管用一般扫描线方法可能会把直线方程的系数变成整数）。如果线段端点被舍入到栅格点上，那么可供选择的线段的斜率就很少，容易导致曲线严重扭曲。

957 ~ 963

一种不同的方法是以一支圆形的笔来画曲线；Posch和Fellner[POSC89]提出了一种颇为巧妙的方法来完成此项工作。他们的方法之所以特别有趣是因为该方法是为硬件实现设计的。基本思想是，首先把曲线扫描转换成一个像素表，像素沿对角、水平或竖直方向上相邻（称为*八路步进*）。然后扩充像素表，使得任意两个像素是水平或竖直邻接的（称为的*四路步进*）。在曲线的起始点处，画一个实心圆（圆盘）。对每个后续像素，以它为中心画一个空心半圆，半圆的直径与当前像素和前一像素间的连线垂直。由于只有四种可能的运动方向，也就只有四种可

能的半圆。图19-39说明应用八路步进技术于曲线的结果，从图中可以看到有些像素丢失了，四路步进技术则不会产生这种情况。

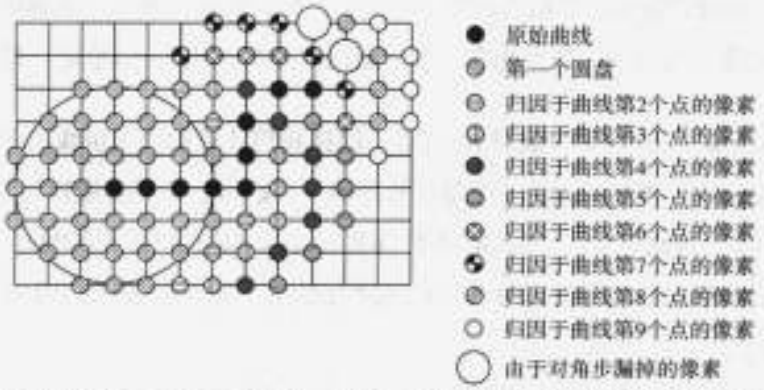


图19-39 在宽线起始像素周围画一圆盘。之后，曲线上的每个其他像素生成一个半圆。注意有两个像素漏掉了（图中用大圆圈表示的部分）

这种方法带来的一个困难是，对于宽度为奇数的曲线，该算法应能生成以原曲线为中心、半径为半整数的圆上的点。这可以通过一个改进的中点画圆算法来实现，所做的修改就是把算法中圆的系数乘以四。另一个困难是，必须用四路步进技术来避免出现裂缝，但是这会产生较粗糙的曲线。该算法也是逐一生成像素，而不是生成跨度，因此把像素快速写到屏幕上也存在困难。解决办法是拷贝到屏幕之前，用19.7节描述的形状数据结构来收集像素使之构成跨度。

19.2.8 填充图元

第3章我们讨论了填充图元及诸如圆、矩形或椭圆之类标准几何图元的内部问题。这些概念相当明确。但对自交图元的内部定义就不那么显然了。我们在第3章讨论了封闭（可能自交）折线段内部的两种定义。一种基于奇偶规则，从待定点引一条到远离折线的某个点的线段，如果这条线与折线的交点个数为偶数，那么认为待定点位于区域的内部，否则就在外部。这里保 964
证被测试线段不与折线交于顶点很关键，不然的话交点计数会出现二义性。按此定义得到的内部区域的外观类似于一张棋盘，如图19-40a所示。另一种规则是基于非外部点规则，选取一个远离折线的点，称为种子点。对任何点如果存在一条与折线不相交的到种子点的路径，那么就认为它位于折线之外。所得到的内部区域由所有这样的点组成，即如果要求填充时，所有位于曲线之内的点。换一种不同的描述方式就是，如果把折线看成围栏，内部区域是指那些可把动物圈起来的部分，图19-40b是这种定义的一个例子。

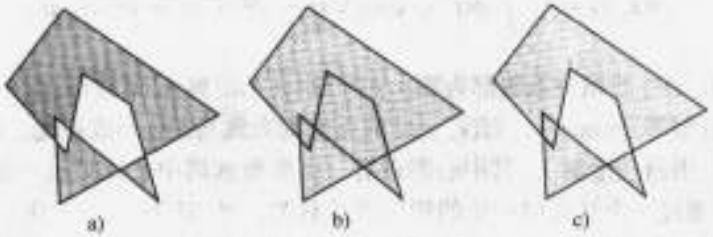


图19-40 a)采用奇偶规则填充的多边形，b)采用非外部点规则填充的多边形，c)非零环绕规则填充的多边形

第三种定义采用所谓的非零环绕规则。一个点P对应于不包含该点的曲线C的环绕次数按下述方法定义。考虑点Q沿曲线C行走一周。对P到Q的向量进行规范化，所得单位向量的末端

沿以P为中心的圆周运动。如果把这个端点的轨迹想像成一条橡皮筋并让它收缩，最后它将缠绕圆圈若干次。环绕次数就定义为缠绕的次数（对顺时针的缠绕，其次数是负数）。

对封闭折线，环绕次数的计算相当简单。跟前面一样，从待定点引一条到远离折线点的射线，当然要保证射线不与折线的顶点相交。如果射线由参数形式$P + t\mathbf{d}$给定，其中$\mathbf{d} = \begin{bmatrix} x \\ y \end{bmatrix}$，定义$\mathbf{d}' = \begin{bmatrix} -y \\ x \end{bmatrix}$，事实上$\mathbf{d}'$可由$\mathbf{d}$逆时针旋转90°而得。同样我们计算射线与折线的相交次数，不过在这里，每次相交按下列规则被赋予一个带符号的值 + 1或 - 1：如果折线边的方向向量与$\mathbf{d}'$的点积为正，则取 + 1，否则取 - 1。对所有这些值求和就得到曲线C关于点P的环绕次数。环绕数规则规定，那些环绕次数不等于零的点为折线的内部点。图19-40c给出了由环绕规则定义的内部区域的例子。

19.3 反走样

第3章已讨论过，对图元进行反走样处理就是要用模糊的边界画图元。我们必须考虑那里及第14章讨论的滤波理论，从这些章我们已经看到对一个有高频分量的信号进行采样时会产生
965 走样。这个事实意味着对信号进行采样之前，应对它进行滤波；正确的滤波器应是$\mathrm{sinc}(x) = \sin(x)/x$。下面我们来考虑这种情形的若干数学结果。

如果计算了两幅包含不重叠图元的图像的亮度值，我们可以逐个把像素的亮度值简单地相加来获得包含两幅图像图元的另一幅图像的亮度（这一叠加原理是用积分定义图像的结果）。据此，如果把图元分解成许多不相重叠的小块，就可以通过计算每个小块的图像，然后对结果求和来得到图元的反走样图。

这种方法的一个极端是，如果可以画单个点的反走样图像，就可以画任意图元的反走样图像，只须把图元看成一个点集，然后对所有这些点的反走样值进行求和（用某种积分）就可以得到最终图像的亮度值了。我们提到这种极端情形，并不是把它作为可行的解决反走样问题的方案，而是作为各种技术的出发点。为了避免定义类似一个点这样的面积无穷小区域的亮度所遇到的技术上的困难，我们将代之以非常小的区域（圆点）的概念[⊖]。这样一个圆点的反走样图是什么？为计算像素(x, y)处的亮度，我们可以把一个sinc滤波器放在像素之上，然后与函数ϕ做卷积，其中ϕ是在圆点内为1其他位置为零的函数。假定圆点位于(a, b)，像素位于(x, y)，我们有

$$I(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \phi(t, s)\ \mathrm{sinc}(\sqrt{(t - x)^2 + (s - y)^2})\ ds\ dt$$

除靠近(a, b)的点(t, s)外被积分函数都为零；对于到(a, b)距离为r的点(x, y)（即从像素到圆点的距离）被积函数近似等于$\mathrm{sinc}(r)$，而(x, y)处的亮度则大致与$\mathrm{sinc}(r)$成正比。因此一个反走样圆点（在黑色背景上用白色绘制），其中心附近有一亮斑而远离中心处则是一组渐暗的同心圆环。（这些圆环与光束通过一个针孔时产生的衍射模式有关，见[BERK68]）。注意，尽管$\mathrm{sinc}(r)$有时为负数，但精确表示这些值是不可能的，因为我们不能画一种比黑色更暗的颜色。在极限情况下，当圆点变得很小时以$\mathrm{sinc}(r)$作为近似的误差也随之变得精确（尽管为避免结果接近于零，

⊖ 有数学基础的读者可以用delta函数来做精确定义。

必须用圆点区域面积的倒数来放大）。

如果我们用上述关系来对一条直线（可看成是一个点集）进行反走样处理，结果是单个点的反走样模式的叠加[⊖]，来自一个点的正波纹与另一个点的负波纹相互抵消，最后的结果是直线附近的亮区域朝远处逐渐变黑。此外，由于平面上画直线非常对称（是沿它自身方向平移的不变量），反走样值的和是到直线距离的函数，而与直线的位置无关。这一不变性意味着，如果计算好沿直线的某个垂直方向向外移动时亮变的变化，我们将可以以不同的方式来使用它。假设线段有轻度弯曲。如果在直线段的每一点都沿该点处的法线方向复制一份亮度变化的值，所 966
有这些变化的值之和将给出一个很不错的弯曲线段的滤波图像。当然，法线可能在曲线外的某个地方重叠，但如果曲线的曲率较小，则在很远处才会重叠，而此时变化的值已几乎等于零。Hatfield采用这一思想创建了一个一般的反走样模式[HATF89]，在19.3.3节我们将会谈到。

对一个区域进行反走样处理，同样可以把区域想像为点集，或想像为一族线段，则可用前面讨论过的线段反走样方法。完全位于区域内的部分具有全部亮度，区域外的亮度则随距离而衰减。如果想计算一个反走样半平面看起来像什么，可以对其他区域进行反走样处理，不过要假定该区域的边界局部地与半平面相近。然而，在实际操作中我们首先对边界进行反走样处理，然后用完全亮度值填充区域内部。

19.3.1 直线的反走样

我们在第3章论述了生成反走样直线的Gupta-Sproull方法。对直线附近的每个像素，用它到直线的距离来确定像素的辉度。在快速实现中，这个距离被转换成介于0和一个较小数（比如16）间的整数，这个整数作为一个灰度值表的索引值，而灰度值则是通过确定锥形滤波器与直线表示的区域的重叠部分来计算的。对于某个固定宽度的直线，滤波器与直线的可能重叠可预先计算好，但对一般宽度（比如19.3.5讨论的矩形），计算滤波器底部与半平面的重叠的加权更好些。滤波器与直线的重叠则是两个滤波器与稍有偏移的两个半平面的重叠的差（见图19-41）。

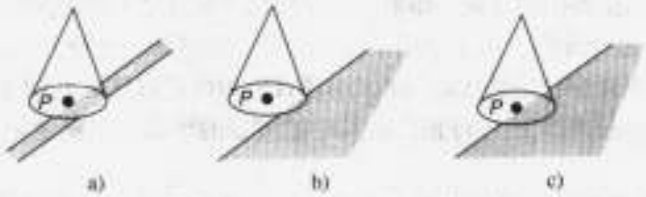


图19-41 中心为P的锥形滤波器与细线的两边都重叠。可以计算该滤波器与两个半平面（b和c）的重叠，每个半平面由直线的一条边作为边界，两者的差可得到a)中的重叠部分

回想一下，从像素中心到直线的距离必须是带符号的数，因为被覆盖四分之三和被覆盖四分之一的像素中心到直线距离是相同的。这里给出一个特别巧妙的解决方案：在中点算法中，判定变量d用来决定像素中心位于直线的一侧还是位于另一侧。判定变量的值与所选像素到直线的距离密切相关。如果记$D(x, y) = ax + by + c$，那么在点(x_i, y_i)处的判定变量是$D(x_i, y_i - 1/2)$，而当$D = 0$时(x_i, y_i)恰好位于直线上。因此，D不等于零时，它的值正好反映了(x_i, y_i)的有向距离。第3章的实现正是以此确定到直线距离的。 967

要处理直线段的端点，必须建立一个更复杂的索引模式。如果线段是粗端的，必须计算一个正方形角与锥形滤波器底部的所有可能的重叠，并且把它们保存到一个表中。这个任务实际上是画反走样矩形这样一个更为一般问题的组成部分，我们把它留到19.3.5节之后讨论。

⊖ 更精确地，结果是来自单个点的贡献的积分。

Gupta-Sproull方法适用于以某种前景颜色在背景上画直线，但并没有处理第二条线段与第一条线段相交的情形。考虑在黑色背景上画两条白色直线段。如果在画第二段的过程中每个像素的亮度作为白色与黑色背景的混合，有两种情况可能发生。如果当前像素值作为背景亮度，那么线段交叉处的点将会过亮。另一方面，如果背景亮度仍采用黑色，则两直线段交点附近的点又偏暗。

引入颜色后问题更复杂。首先必须对有关交点处的问题做出判断。交点处的颜色是上一条线的颜色吗？或者它是已有的颜色与上一条线颜色的混合？本质上说，我们必须从第17章描述的合成操作中选择一种。但是，Duff-Porter合成方法中所做的假设（即如果一个图元覆盖像素的比例为α，那么它对该像素处任何其他图元的覆盖比例也是α）在线段相交的一般情况下并不成立，所以在这里使用α值合成方法来选择交点处像素的颜色肯定会令人失望。

Texas 仪器公司的TI34020芯片提供了取两个像素值*src*和*dest*最大值的运算符MAX，并以两个数中较大的那个代替*dest*。这种以较大的值作为反走样彩色直线交点处的像素值的方法，是一种较成功的折中。

Barros和Fuchs[BARR79]的一篇早期论文描述了一种在灰度值设备上生成反走样直线的方法。该方法本质上是一种多边形区域（单像素宽直线是它的一种特殊情形）的扫描线绘制器。所有要画的多边形被收集起来，然后通过确定像素不被直线遮挡的部分计算该像素的光强值。细分方法可用来确定每个像素的不相交的非遮挡区域，然后累计这些区域的面积即可。

另一种值得一提的反走样直线方法用来处理设备分辨率很低的情形，也许只有两个位（正如NeXT机那样）。此时，像素值的选择是如此粗糙以至于应用Gupta-Sproull反走样方法在计算上是一种浪费。而一个严格的面积采样技术就可以做得非常好了。（一般地，当离散化到4层中的一层时，区域采样技术与Gupta-Sproull方法的差别就变得很小，因而结果基本上一样）。有几位作者[PITT87；PITT80]提出把直线$Ax + By + C = 0$中点算法的判定变量d（对于斜率在 - 1到1之间的直线，它的值介于 - A和B之间）用于计算反走样值：对紧邻几何直线的像素，只须简单地计算$(d + A)/(A + B)$，此式的值在0和1之间。如果仅有少数几个灰度值（例如在每像素2位的机器上），
968 这种对加权重叠的逼近是合理的。由于应用于像素位很少的显示，这种方法被称为2位反走样。

最后，将在19.3.3节详细讨论的Hatfield方法[HATF89]也可以迅速地用于直线；在这种情形下，它退化为Gupta-Sproull模式的一种推广，其中滤波器的形状可以随意定义。

19.3.2 圆的反走样

在讨论圆的反走样处理前，首先引入一些术语。一个圆盘是一个实心圆。一个圆是平面上到一个点距离不变的点集，因此是无限细的。而我们提到宽圆时，是指到某中心点的距离介于两个值之间的点集。两个值的差是带宽度圆的宽度。因此一个宽圆可以看成是一个较大的圆盘与一个较小圆盘的（集合意义上的setwise）差。

可按下面的方法来对一支单位宽度的笔所画的圆做反走样处理。对圆附近的每个点(x, y)，记$S(x, y)$为圆算法中的判定变量，那么$S(x, y)$与点到圆的符号距离成正比。像处理直线那样，对这个数进行适当的比例缩放，可以用来作为Gupta-Sproull式查找表的索引值。回顾一下，对于直线可用两个稍有偏移的半平面的差来计算重叠部分。对于圆也有类似的方法，可以先计算两个同心圆盘与滤波器底部的加权重叠，两个结果相减就可以得到宽圆与滤波器底部的重叠部分。不过，稍有不同的是，对于圆来说，圆盘与锥形滤波器底部的重叠不仅与像素到圆盘边界的距离有关而且与圆盘的半径有关(见图19-42a)。

根据上面的分析知道，对于不同半径的圆盘需要不同的查找表。但是，注意到，如果有一个小区域（比如包含在半径为2的圆盘内的区域）位于半径为10个像素的圆盘边界附近，如图19-

42b所示，而另一个则是位于半平面的同样距离处，两个图元覆盖的区域是相似的（对大多数距离而言，差别小于0.004）。因此，用半平面作为半径为10或更大圆盘的近似是很安全的。因而，为进行反走样像素值计算这一目的，对于半径较大的圆，可以直接用直线反走样算法中的亮度表；对较小的圆，则需要逐一计算，即对每个整数半径的圆盘建立一个索引表，对非整数半径，光强通过插值得到；这样就可以获得比较理想的结果了。

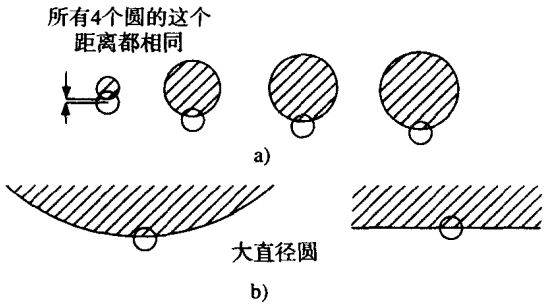


图19-42 a)锥形滤波器底部（图中表示为一个开圆）与圆盘的重叠部分随着半径的不同而不同，即使与圆盘的距离相同的情况也是如此；b) 锥形滤波器底部（或任何其他小的区域）与半径为10的圆盘的重叠部分和半平面与圆锥滤波器底部的重叠部分大致相同

图19-43给出了用于反走样圆算法的伪代码。该算法确定一个具有单位宽度的圆附近像素的亮度值。其方法是，对半径为r的圆附近的像素，先确定它与半径为$r+1/2$和$r-1/2$的两个圆盘的重叠部分，然后两个值相减即得到与半径为r、宽度为1的圆的重叠。当然，所有这些重叠计算都进行了加权。惟一的困难是，如果r不是整数，两个圆盘的重叠计算必须由线性插值得到。例如，如果$r=6.25$，我们需要知道像素（注意像素表示一个加权面积采样中的圆形区域）与半径为5.75和6.75的两个圆盘的重叠部分。第一个圆盘的重叠部分通过计算半径为5和6的两个圆盘的重叠部分，然后在5和6之间的四分之三处插值得到。第二个圆盘的重叠部分可以类似求得。

```
/* 画一个已反走样的半径为r的圆 */
void AntialiasedCircle (double r)
{
    int r0;                          /* (r + 1/2) 的整数部分 */
    double f;                        /* (r + 1/2) 的小数部分 */
    double i1,                       /* 归因于半径为 (r0 - 1) 的圆盘的亮度 */
           i2,                       /* 归因于半径为 r0 的圆盘的亮度 */
           i3;                       /* 归因于半径为 (r0 + 1) 的圆盘的亮度 */
    double iInner,                   /* 归因于半径为 (r0 - 0.5) 的圆盘的亮度 */
           iOuter;                   /* 归因于半径为 (r - 0.5) 的圆盘的亮度 */

    r0 = 小于等于r + 0.5的最大整数
    f = 0.5 + r - r0;
    for (半径为r的扫描转换圆附近的每个像素) {
        d = 至半径为r的圆的距离;
            /* 与判定变量S(x, y)成比例 */
        i1 = 用半径为(r0 - 1)的圆盘加权的像素区域覆盖;
        i2 = 用半径为(r0)的圆盘加权的像素区域覆盖;
        i3 = 用半径为(r0 + 1)的圆盘加权的像素区域覆盖;

        iInner = (1 - f) * i1 + f * i2;     /* 插值 */
        iOuter = (1 - f) * i2 + f * i3;
        intensity = iOuter - iInner;
        WritePixel (当前像素,    intensity);
    }
} /* AntialiasedCircle */
```

图19-43 反走样圆算法的伪代码

969
~
970

最后，为画一个较宽的圆（比如5个像素或更多），可以画两个反走样同心圆，然后再调用填充算法；对较细的圆可以像第3章那样画跨度。

基于下面的观察，可得到一个稍微简单点的算法。首先，滤波器与加宽圆（圆的半径足够大）的重叠直接与滤波器中心和圆的距离有关。其次，这个距离与滤波器中心处关于圆的残差直接相关。容易证明，对圆附近的点(x, y)，残差$F(x, y) = x^2 + y^2 - R^2$近似等于$2Rs$，其中$s$是$(x, y)$到圆的距离。因此，用半径的2倍去除残差就可以算出点到圆的距离。用这个距离来对亮度值表进行索引，就可以确定圆附近点的亮度值（见习题19.26）。

19.3.3 圆锥曲线的反走样

为找到一种生成反走样圆锥曲线的好方法人们做了大量的工作。Pitteway的2位算法[PITT87]对像素位很少的机器效果很好，但不能很好地推广到像素位较大的情形。当以几个位对像素进行显示时，这种方法对加权重叠的线性逼近与正确值的差别明显可感觉到。Field的圆和椭圆算法[FIEL86]只能处理中心为整数的圆和水平或竖直的椭圆，并且只做未加权区域采样。如果可以算出圆锥曲线每点处的曲率，并且曲率值不太大的话，那么可以使用相同曲率的圆的一系列类似Gupta-Sproull风格的查找表，不过代价也很大。

Hagen［HAGE88］给出了反走样直线、圆锥曲线和三次曲线的一般方法，该方法基于通常的与点到曲线距离相关的反走样值的利用，但确定附近像素的方法更巧妙些。如果想要使几个像素上的曲线变模糊，必须确定哪些像素靠近曲线。当用增量算法来计算曲线上的点时，要么沿基本水平的方向前进（例如，每一步选择东与东北向之间的像素），要么沿基本竖直方向前进（例如，每一步选择北与东北向之间的像素）。当曲线跟踪算法沿竖直方向前进时，我们可以沿水平方向步进以找到曲线附近的点，或反过来。但是，正像第3章所描述的，这一过程在宽曲线的象限变化处会留下“凹槽”。而且，如果用这一技术确定哪些像素需做反走样处理，在同样点处也会留下反走样裂缝。Hagen通过时而沿水平方向加宽，时而沿竖直方向加宽，而另一时候沿对角线方向加宽，来解决这个问题（见图19-44）。

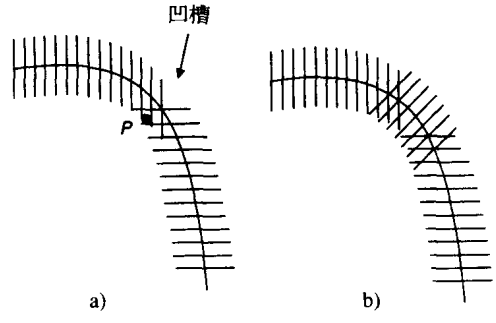


图19-44 通过确定哪些像素位于曲线附近选择何处需计算反走样值。如果总是沿水平或竖直方向移动来确定附近像素，会产生裂缝，如a)中所示。如果在某些时候沿对角线方向移动，这些裂缝就可以消除，如b)中所示

当然，这意味着曲线附近的一些像素点被赋两次亮度值。这两次赋值应几乎相等，从而使得重赋值不成问题（假定我们以**replace**模式画图）[⊖]。但对严重弯曲的曲线（见习题19.18），这会有一些困难。一个像素可能会与曲线的两个部分都很靠近，如图19-45所示，因而它被赋予的值应是来自两部分的和。Hagen注意到这个问题但没有给出解决办法。注意，简单地累加一个像素的值并不能解决问题：像素被赋值两次的原因可能是由于对曲线的两种不同的加宽（例如，被一个水平加宽和一个对角加宽同时遇到的像素，图19-44中的像素P就属于这种情形），也可能是由于靠近曲线的两个不同部分（例如，图19-45中的像素Q）。

⊖ 我们这里提到的所有算法最适宜于在空画布上以**replace**模式画图元。画多个图元时，一般要在各自的画布上画好每个图元，然后用MAX运算符或某种合成操作把它们复制到最终的画布上。

Hagen在论文的最后提出一个使用宽度扫描对此类曲线进行反走样处理的方法：在每一个点，沿曲线的法线方向画一条直线段，然后对法线上的每一个点（充分靠近）计算亮度值。前面提到过，这一思想已由Hatfield[HATF89]实现。他注意到，对隐式方程（比如由圆的$F(x, y) = x^2 + y^2 - R^2$）定义的曲线进行扫描转换过程中，当计算曲线每个点的残差时，需要计算函数的各种偏导数。这些反过来则用来给出曲线的法向量[⊖]。

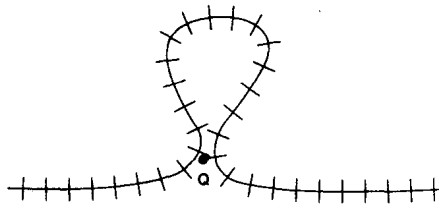


图19-45 在一条弯曲程度大的曲线内，有些点与两个不同的部分相近，它们理应比其他点更亮

对于三次曲线，假设它经过两个像素之间的某个位置，那么通过比较两像素的残差并进行线性插值以求出残差值为零的点，也能够得到这个位置的一种较好的逼近（见图19-46）。如果这点处的残差不为零，重复一次这个过程将可以产生零点的一个很好的逼近。（这一技术实际上也可以应用于更高次多项式定义的曲线）。

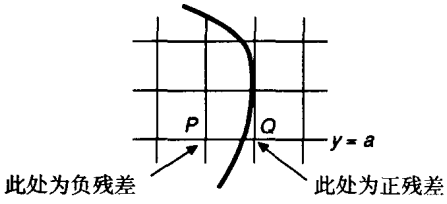


图19-46 曲线经过栅格点P和Q之间，所以在P和Q处的残差符号相反。如果P点处的残差是－10，Q点处的残差是5，就可以估计曲线通过从P到Q连线的三分之二的位置

采用偏差分还可以求出每点处的曲率半径的较好的近似值。尽管公式较复杂，但只包含隐函数F的一阶和二阶导数。曲率半径是曲线上某点处的最佳逼近圆的半径，所以可以认为此处曲线局部地就是这个圆。

因此，对扫描转换曲线上的每一个点，我们已知曲线的斜率、法线方向向量、到曲线的距离以及曲线的曲率半径。现在我们将不是沿水平、竖直或对角线方向步进以拓宽曲线，而是对曲线的法线做扫描转换，并计算每个像素到曲线的距离。然后用这个距离对亮度变化曲线进行索引，这条曲线刻画了像素远离半径为当前曲率半径的圆时亮度的下降情况。要精确地做到这一点，要求对每个可能的曲率半径计算（或通过微调提供）亮度变化曲线，这是不现实的。幸运的是，对于大于10的半径，Hatfield证明假设曲线在这点处为一直线可产生较好的效果。对于非整数曲率半径，查找相邻的（整数半径）亮度变化曲线并进行插值也能使结果平滑。

如果从反走样半平面的亮度变化曲线出发，那么通过稍许平移变化曲线并相减，我们可以计算细直线或曲线的适当的亮度变化曲线。这对应于以一个半平面减去另一个稍做偏移的半平面生成细直线的值。这一技术对偏移不超过1/2像素时效果良好（见图19-47）。对于比这更小的偏移，直线看起来是暗淡而不是细小。当然，用这种轮廓曲线描述的像素的总体亮度与曲线下面的面积成正比。为了获得相对而言很窄的曲线，需要用大于1的数去乘以光强变化曲线的值。Hatfield指出选择一个变化曲线以保持直线的亮度而不至出纰漏仍然只是一种技巧。

我们可以用类似的方式处理宽线，只需简单地增加一批偏移亮度变化曲线。对于很宽的曲线，对其外部进行反走样处理，而对内部只做区域填充是最恰当的。

971
~
973

注意，如果亮度变化曲线的宽度比曲率半径还大的话，沿着它们做反走样处理的各种直线可能会相交（见图19-48）。应把得到的亮度累加起来，而不是只是保存在输出像素图中。

⊖ 回想一下，我们在第3章中曾用梯度来确定法向量。

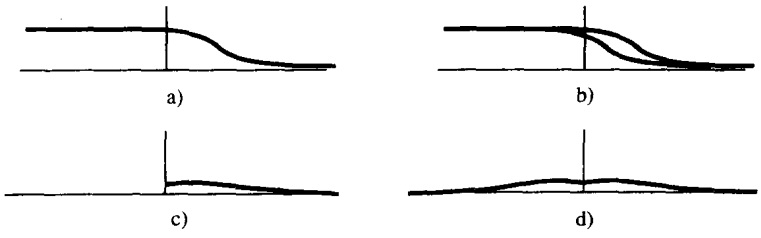


图19-47 a)显示了一个半平面的亮度变化曲线；b)中第二个副本的位置已做了少量偏移；c)显示两者差的右半部分；d)中给出的是一条窄线的整个亮度变化曲线

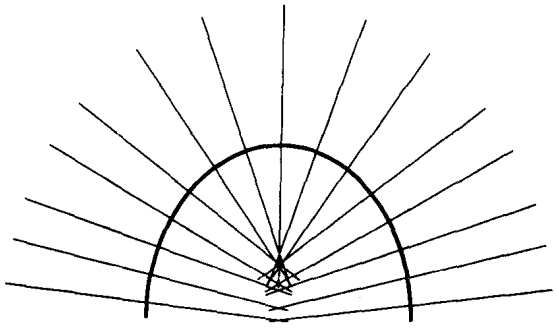


图19-48 当曲线的曲率半径很小，但亮度变化曲线很宽时，曲线的不同点可能对同一个输出像素产生影响

19.3.4 一般曲线的反走样

一般曲线的反走样方法较少。一种方法是把曲线当成一个非常短的直线段的序列，并使用多边形反走样方法[TURK82；CROW78]。另一种由Whitted［WHIT83］给出的方法实际是用一个反走样画刷来绘制线。画刷（一个像素矩阵）以很高的分辨率建立（它的每个方向上的密度是在其中进行画图的画布的16倍），然后进行滤波以消除高频成分（滤波器把频率减小到高分辨率采样频率的十六分之一；因为这个高分辨率画刷最终用来确定低分辨率位图的像素值，为避免走样，进行滤波处理是很必要的）。画刷拖动的路径也以高分辨率存储起来（每个方向上为分
974 辨率的16倍）。对路径上的每个点，高分辨率画刷中正好与低分辨率输出像素图中相对应的像素被复制。另外，给画刷及输出图像中的每个像素赋一个z值，并且用z缓存算法来控制像素复制。如果画刷是用任何非平整的轮廓描述（比如半球状），那么每次做少量移动时，它就不会重写自己。而且，用类似于17.6.1节中描述的α值（覆盖值）可以增强边界的外观。这一技术生成的结果令人印象深刻（参见彩图IV-2），但因此而付出的计算代价也很高，特别是内存，因为图像的每个像素都有R、G、B和z值，而画刷的每个像素则要R、G、B和α值。另一方面，画刷的准备尽管也较昂贵，但好在每个画刷只需一次，并且相关数据可保存起来供以后使用。特别地，可以生成各种宽度直线的画刷以便创建光滑曲线，尽管对太宽的曲线这一技术变得不再实用。

19.3.5 矩形、多边形和直线端点的反走样

Gupta和Sproull［GUPT81a］提出一种生成反走样直线端点及直线的方法。开始，正如对直线那样，可以明确地计算出一个大矩形角点附近像素的反走样值。但是，这种情形下，它依赖于两个数（到两个邻近边的距离）而不是仅仅到一条直线的距离。从图19-49可以看出矩形与用于像素的滤波器底部的重叠情况。但当矩形很细时，如图19-50所示，情况会怎样呢？像素Q的滤波器底部与矩形的三条边相交。可以用相减的办法求Q的反走样值：先计算滤波器底部与两个矩形的加权重叠，这两个矩形的差等于原来的小矩形，然后两个数相减就得到小矩形

的加权重叠。因此，不是为所有像素中心到四条边的距离都建立一张重叠表（那将是一张有四个索引值的表），我们可以用二值索引表两次，然后做减法。如果像素与所有四条边重叠，则需要做另一种加法和减法。

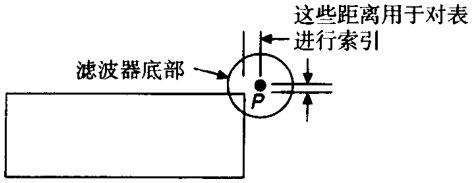


图19-49 通过计算P到矩形角点处两条边的距离确定P的亮度值。这两个距离用来对查找表进行索引

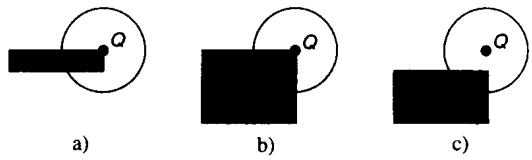


图19-50 细矩形（或宽直线）端点附近像素的亮度值可以通过相减来计算。这里，a)中像素Q的亮度可从b)减去c)的差得到

采用类似的方式，可以对圆端的矩形进行反走样处理，只要计算一个矩形的亮度和一个半圆盘的亮度即可。根据Hagen[HAGE88]的方法，斜面接合的宽线段可用矩形的方法来处理，多边形也类似。多边形角点附近点的亮度通过查矩形角点表来计算。如果角度与90°差别很大，结果不是特别好。Hagen增加45°和135°角的两个表作为锐角和钝角的代表，用较小的代价得到了很好的结果。

19.4 文字的特殊问题

迄今为止给出的算法只生成各种几何图元的位图及反走样版本，有关文本内容并没有论及。文字是一种非常特殊的实体，前面的技术一般地说来是不够的。第3章我们讨论过用字体高速缓存存储字符，然后它可直接复制到位图，但这种方法有它的局限性：对不同尺寸的字体需要不同的高速缓存，而字符间距又固定不变。另外，尽管粗体和斜体文字可从字体高速缓存创建，但这样得到的效果通常不能令人满意。

以字母“m”为例。字体设计者用几何笔划的方式给出了它的精确描述，那么我们如何建立它的位图表示呢？图19-51显示了字母“m”的笔划及其位图表示。可以看到，即使最精细的扫描转换，所得位图的中间一竖都比两边竖线细。

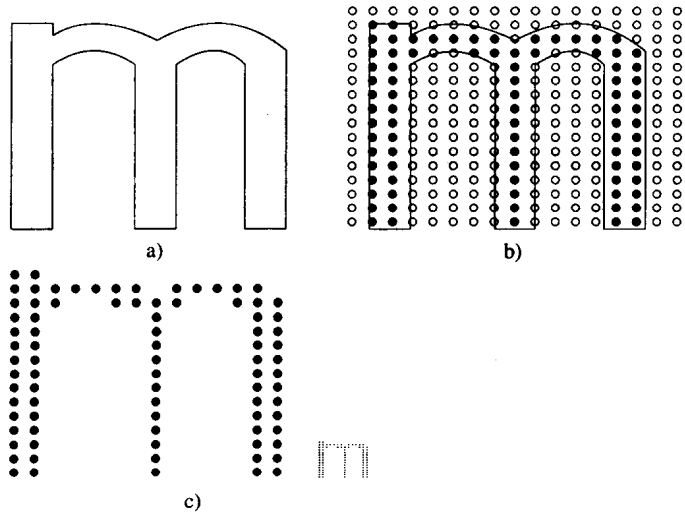


图19-51 由于中间一竖太细，字符的位图表示看起来很糟。a)字符轮廓，b)轮廓内像素置为黑色，c)完全尺寸和缩小后位图字符的外观

因此，字符不能逐划进行扫描转换。字符形状的几何特性是密切相关的——某些元素必须有相同的宽度，某些拓扑特征必须保持，而同一种字体中不同字符间的一些关系也应保留（例如，在某些字体中，字母“H”的竖线的宽度必须与字母“O”的孔的宽度相同）。即使设法创建了字符本身及字符之间都保持正确的几何关系的字体，仍有一些问题需要克服。比如，从字体设计图生成大写字母“O”时，应创建两个同心椭圆弧。当以某种尺寸的点对此进行扫描转换以生成像素表示时，可能会导致非常糟糕的结果。图19-52显示在栅格上的字母“O”及扫描转换后得到的位图。“O”的顶部只是稍微穿出一条扫描线之上，因此这条扫描线上只有一个像素被点亮。得到的结果有一个小突起，既不美观又难于阅读。位图字体生成软件应该避免这样的缺陷。为满足诸如此类几何和印刷上的考虑，很多商家已开发出基于规则的软件，但字体仍需手工调整，在较低分辨率时更是如此。前面提到的只是一些显而易见的问题。还有大量其他问题，解决这些问题则是数字印刷的特殊领域的任务；为获得进一步的资料，可参考[RUBE88]。

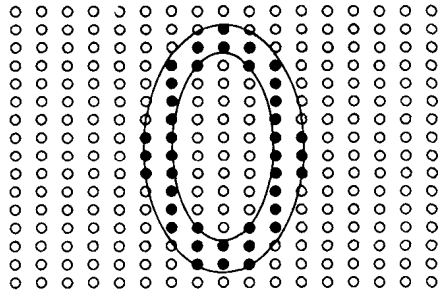


图19-52 扫描转换字母“O”会在字母的顶部产生一个小突起

在很多方面，反走样的文本比每像素一位的文本更简单。只要打算对字符作反走样处理，其位置就无关紧要（反走样是在连续而不是离散空间发生），所以当需要输出高质量字符时可以在子像素级指定字符位置。像小突起、孔洞及笔划宽度不均匀之类的缺点也自动消失。内存问题仍然存在。典型的字体通过表示字符轮廓的一系列样条曲线来描述。字体（字体的几个不同的子像素位置，称为相位）中所有可能字符外观的预处理需要大量的内存。Naiman和Fournier[NAIM87]估计，对一个典型屏幕像素密度，如果每像素占八位，那么做有128个字符，包括罗马、粗体、斜体和粗斜体四种字体，五种尺寸及在水平和垂直方向各八个相位的两套字体，需要50MB的存储空间。对于不要求产生高质量输出的用户来说，64种不同相位似乎已经够用。不过请不要忘记，我们谈到只是拥有5种尺寸的两种字体罢了。即使不考虑相位，只要把字体和尺寸都扩展到10种，同样很快达到50MB空间。

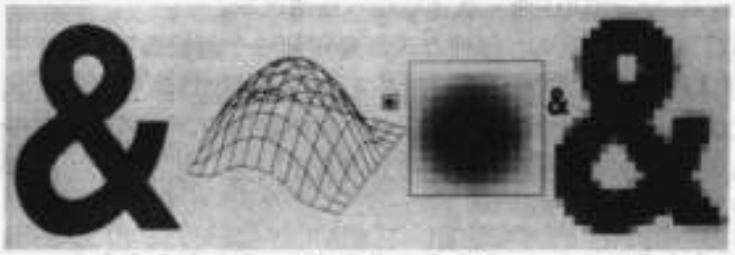
975
~
977

B样条曲线描述的另一种字符生成方式是对样条曲线进行缩放和平移变换，直到它们表示恰当位置的字符为止。[⊖] 然后在此位置计算字符的反走样位图。这种方法要求在每次使用字符时都重新做反走样计算，除非有巧妙的方法来加速计算过程，否则它在计算上是不切实际的。幸运的是，Naiman和Fournier提出了这样一种方法。他们把每个字符分解成若干小片，这很有点类似于下一节描述的形状数据结构：每个字符由多个矩形的并来表示。分解方法是在某个非常大的尺寸下对字符进行扫描转换并生成一个母版（master）。然后把母版分解成矩形，而所有其他较小尺寸的字符则通过对母版进行过滤得到。

在最基本的那一层，过滤字符通过用一个滤波器函数对母版字符（一个由0和1组成的$m \times m$方阵）进行卷积得到，而滤波器函数则表示成元素总和为1的$f \times f$方阵形式。卷积通过从母版字符方阵中选取采样点矩阵来实现。如果输出字符方阵为$g \times g$（这里g当然比m小），那么采样栅格也是$g \times g$，采样点的空间间隔是m/g。注意，在水平和垂直方向上采样栅格都可以有m/g个不同的位置（这些偏移就是字符的相位）。此后，把滤波器拷贝一份放到每个采样栅格点上，

⊖ 在不同的尺寸定义字体时，可能需要稍有不同的曲线——缩放变换并不总是足够的。

对此采样点按照如下方式计算出一个值：对每个滤波器矩阵的每个点，把滤波器的值乘以该点处母版方阵的值，然后求和。得到的$g \times g$采样方阵即是过滤后的字符（见图19-53）。



a) b) c) d)

图19-53 a) 被采样栅格覆盖的母版字符；b) 一个连续滤波器的图形；c) 滤波器的灰度表示，已放大；d) 过滤后的字符，也已放大。经Toronto大学的Avi Naiman许可

Naiman和Fournier还注意到在每个像素处计算过滤值所花代价也是不必要的。如17.4.3节那样，他们用一个总计面积表来表示滤波器，从而使任何矩形的过滤值计算变得很容易。因为母版字符已被分解成矩形块，可以简单地对母版字符的所有矩形块进行循环，求采样点处与滤波器盒重叠的部分，然后查找总计面积表来决定采样点处对亮度的贡献。这些亮度汇集了母版字符中的所有矩形块，最后结果就是像素的亮度。尽管此计算对每次采样都重做一次，平均意义下仍然比直接在每个点处计算采样值要快得多。开始其他工作之前，对矩形和滤波器盒进行范围检查，并利用相交的矩形表中采样点之间的相关性也可以极大地提高算法的性能。

经过反走样处理的文本看起来很漂亮，已用于IBM公司开发的YODA显示器［GUPT86］。彩图IV-3是这种文本的正常尺寸和放大尺寸的例子。

19.5 填充算法

有时，画好一系列图元之后，我们可能想在图元的内部填充颜色，或者可能希望在徒手画出的区域内涂上色彩。例如，通过创建栅格线然后用各种颜色填充栅格区域来得到马赛克（贴砖）图案效果要比直接在平面位置上均匀地排放涂有颜色的正方块要容易得多。请注意，使用前一种技术并没有涉及二维图元：画线之后所用到的正好构成背景的二维区域。因此，判定需要填充多大的区域归结为检查何时到达边界。完成这一工作的算法称为*填充算法*。这里我们将讨论*边界填充*（boundary fill）、*泛填充*（flood fill）和*浓淡填充*（tint fill）。最后一种是更为精巧的算法，属于*软填充*（soft fill）的一种。在这种算法中，区域的边界不由检测到另一种颜色的像素来决定，而是检测原来颜色消失为零的像素。因此，采用浓淡填充可把一个渐变为橙色然后到黄色的红色区域（在一张红色的画布上用反走样方法画一个黄色的圆时，圆内的情形正是如此）变为蓝色。橙色区域带部分红色，所以它们的红色分量被蓝色替代，结果是一个蓝色区域渐变为绿色直到黄色边界。

我们所讨论的算法中，赋予内部像素的颜色值称为新值（*newValue*）。按照Fishkin和Barsky［FISH84］的说法，每个算法可以被逻辑地分成四个部分：*传播方法*，它决定下一个要考虑的点；*起动过程*，用来对算法进行初始化；*内部过程*，判断像素是否位于区域内及是否应该填充；最后一个是*设置过程*，改变像素的颜色。

19.5.1 区域类型、连通性和填充

区域是一个由像素构成的集合，有两种基本类型。一种称为*四连通区域*，区域内的每两个

像素只能用通过上下左右四种运动得到的像素序列连在一起。与此相对应，另一种称为八连通区域，它里面的每两个像素可以用通过上、下、左、右、右上、左上、右下和左下八种运动得到的像素序列连在一起。一个四连通区域肯定是八连通区域。

978
~
979

一个区域可通过两种方式定义。每种方式都要用到一个起始像素P。内部点定义的区域是颜色值与P相同的点构成的最大连通区域。边界定义的区域则是颜色值不等于某个给定边界值的像素构成的最大连通区域。由于绝大数算法都是递归实现的，当遇到具有新颜色的像素时递归会终止。因此边界定义的区域内有新颜色像素时会出问题，因为可能有的递归分支过早返回（见图19-54）。

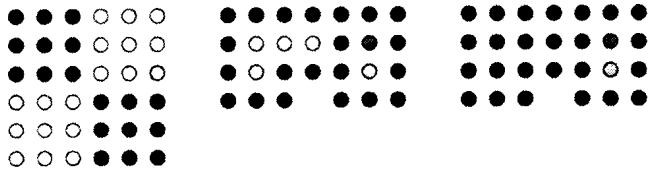


图19-54 a)中的黑色像素定义了一个八连通区域，但不是四连通区域，因为对角的两个正方形区域不是四连通的；b)中的白色像素形成内部点定义的区域的内部点，如果区域是由黑色像素边界定义的，则浅灰和深灰的像素也属于区域内部；如果我们试图用深灰色填充此区域，并从一个白色像素开始，则填充的结果可能如c)所示，由于与起始像素间有一个深灰像素，最右端的像素没有被填充到

填充内部点定义区域的算法称为*泛填充算法*，而填充边界定义区域的算法则称为*边界填充算法*。由于两者都是从区域内的一个像素开始，所以有时把它们都称为*种子填充算法*。

19.5.2 基本填充算法

最原始的传播方法是从起始点开始，沿四个或八个方向移动，并递归地利用此方法。对于这里给出的FloodFill和BoundaryFill两个过程，我们已经确定区域的内部，而像素设置例程只是简单地调用WritePixel。图19-55列出了四连通情形算法的代码，八连通情形递归调用次数是八个而不是四个。

```
void FloodFill4 (
        int x, int y,                   /* 区域中的起始点 */
        color oldValue,                 /* 定义内部点的值 */
        color newValue)                 /* 替换值，必须与oldValue不同 */
{
    if (ReadPixel (x,y) == oldValue) {
        WritePixel (x, y, newValue);
        FloodFill4 (x, y - 1, oldValue, newValue);
        FloodFill4 (x, y + 1, oldValue, newValue);
        FloodFill4 (x - 1, y, oldValue, newValue);
        FloodFill4 (x + 1, y, oldValue, newValue);
    }
}  /* FloodFill4 */

void BoundaryFill4 (
        int x, int y,                   /* 区域中的起始点 */
        color boundaryValue,            /* 定义边界的值 */
        color newValue)                 /* 替换值 */
```

图19-55 泛填充和边界填充算法

```
{
    color c = ReadPixel (x,y);
    if (c != boundaryValue &&      /* 还不在边界上 */
        c != newValue) {           /* 之前我们还没有到达此处 */
        WritePixel (x, y, newValue);
        BoundaryFill4 (x, y - 1, boundaryValue, newValue);
        三种其他情况;
    }
}  /* BoundaryFill4 */
```

图19-55（续）

这些过程虽然简单，但高度地递归，太多的递归层次不但耗时且内存受限时容易造成堆栈溢出。为此人们已提出一些效率更高的区域填充方法[LIEB78；SMIT79；PAVL81]。这些算法需要更多的逻辑分析，但除了一些退化情形外，堆栈的深度不再是问题。其中一个基本算法采用跨度的概念。跨度由值为*boundaryValue*的端点界定，且其内部不包含新的值*newValue*。跨度可以用一个循环重复进行填充。一个跨度由它最右的像素作为代表，对每个未被填充的部分，至少有一个跨度保存在堆栈中。

算法过程如下。首先填充包含起始点的像素构成的邻接水平跨度。然后从刚填充过的跨度的上一行开始从右到左检测以找到每一跨度的最右像素，并把这些像素的地址压入堆栈。对刚填充过的跨度的下一行做同样的工作。当一个跨度用这种方式处理完后，栈顶像素被作为新的起始点，栈变空时算法结束。图19-56显示了一个典型的算法运作的例子。图19-56a中包含起始点的跨度已被填充（起始点用空心圆表示），编有序号的像素的地址已入栈。序号指明像素在栈里面的顺序：1表示在栈底，并且最后一个被处理。图中只列出了区域填充的部分过程。读者可以对剩下的区域逐步完成填充的过程（见习题19.9和习题19.20）。

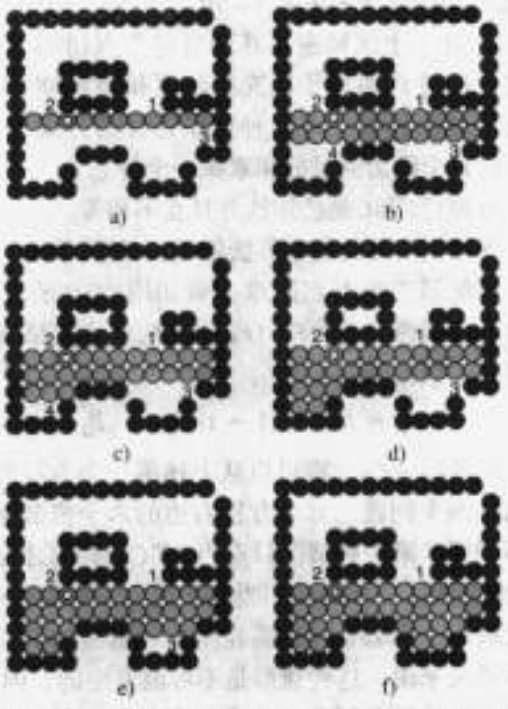


图19-56 递归填充算法的过程。a)具有一个填充跨度的区域，开始像素有一个空心；b)~f)随后跨度的填充

该算法可以通过避免邻近扫描线的重复检查得到改进［SMIT79］。在填充一条扫描线之后，对它的上一条线进行扫描以求取跨度，如果得到的所有跨度都落在当前扫描线的跨度内（称为当前扫描线的阴影），那么在处理上一条扫描线的过程中就不需要再处理它下面的扫描线。同样扫描线的跨度查找过程可以与像素填充过程合并，从而避免像素的多次读取操作。

另一种更好的方法是，通过稍大一点的内存开销可以加速当前直线的上下两条直线的整个扫描过程。填充当前跨度的像素之后，对上一条直线进行搜索，找到与当前跨度相关联的

像素时，把该种子像素与当前跨度的两个端点一起压入堆栈（因而当前跨度也叫父跨度）。然后从种子点开始调用填充过程。如果在填充过程之后，被填充的跨度没有超过父跨度的端点，则继续对父跨度的阴影进行扫描，直到找到另一个种子像素为止，并且这个种子像素被用来起动一个新的填充。对多数情况而言，当前跨度的上一个跨度都不会比当前跨度小，因此往往不需要再做扫描以获得新的种子点[LEVO82]。

980
~
982

19.5.3 软填充算法

软填充算法用于填充由于某种原因，主要是反走样，造成的边界模糊的区域。有好些作者对此进行了研究［LEVO78；SMIT79］，而Fishkin和Barsky则极大地推广了Smith的技术［FISH84］。假定区域最初是在某种背景色之上以另一种前景色进行绘制。如果像17.6.1节那样，需要填充的区域有一个相关联的α值，我们可以用它来检测点是否位于区域内并决定如何对它进行填充：在[LEVO78]中，将*newValue*与背景色按α的比例进行混合得到填充值。但是有许多区域即使无法获得α值，也可能要求重新填上颜色。按Fishkin和Barsky的做法，我们做以下三种假定：

1) 在颜色为C的背景上以前景颜色F对区域进行绘制。图像中的每个像素颜色是F与C的凸组合：$P = tF + (1 - t)C$。

2) 有一个区域遍历算法对每个点访问一次。改进后的种子填充算法可稍做调整来完成此项任务；另一种可供选择的方案是，给每个已访问过的像素赋一个标志。

3) 颜色F和C是已知的并且互不相等。

进行软填充的基本算法如图19-57所示。

```
void LinearSoftFill (
        regionType region,
        color F,        /* 前景颜色 */
        color C,        /* 背景颜色 */
        color N)        /* 新的前景颜色 */
{
    for（该区域中的每个像素）{
        color P = 该像素的颜色值;
        find t so that P = tF + (1 − t) C;
          将 P 替换成  tN + (1 − t) C;
    }
}  /* LinearSoftFill */
```

图19-57 基本软填充算法

算法第二步有点困难。假如用RGB三元组表示颜色，记为$F = (F_R, F_G, F_B)$，对C和P也引入类似的记号，则我们知道对某个t，下列方程成立（根据假设1）：

$$P_R = tF_R + (1 - t)C_R, \qquad P_G = tF_G + (1 - t)C_G, \qquad P_B = tF_B + (1 - t)C_B$$

如果$F_R \neq C_R$，则可以从上述第一个方程解出t。否则，可以使用第二或第三个方程。这样

983

就出现两个问题。如果方程右边的六个值都逐对相等，情况会怎样？如果按三个方程得到的t值不同呢？第一种情况只有当F和C相同时才会出现，而根据第三个假设，这种情形已被排除。（应注意到这个假设是相当自然的：如果给你一幅暴风雪中的北极熊的图片，并要求以棕色重画北极熊，你怎么知道熊在哪里开始雪在哪里结束呢？）。第二种情况则比较麻烦。从严格的数学意义来说，这种情形是不可能发生的，因为P是F和C的线性组合。不过，在整数空间（这正是颜色值的典型表示方式）存在舍入误差。最好的t值来自F和C分量相差最大的那个方程的解。这样就得到图19-58中的更健壮的算法。

这个算法的不足之处是它只允许对背景色为单一颜色的区域进行填充。也许我们想对一幅青蛙坐在红黑相间棋盘上的图片进行填充。Fishkin和Barsky［FISH84］使用下面介绍的一些线性代数的知识解决了这个问题。在n维线性空间中，任何足够一般的$n + 1$个点确定一个坐标系统。（这等于要求不存在$n - 1$维仿射子空间包含所有这些点。例如，在三维空间中的四个足够一般的点，就没有平面包含它们）。如果这些点记为$v_0, v_1, \cdots, v_n$，那么空间中的点p可惟一地写成下面的线性组合：

$$p = v_0 + t_1(v_1 - v_0) + t_2(v_2 - v_0) + \ldots + t_n(v_n - v_0)$$

其中t_i是实数。典型的是，通常以v_0为该空间的原点，$v_1, v_2, \cdots, v_n$为基向量。通过假定用几种颜色所画图像其像素颜色值为这几种颜色的线性组合，Fishkin和Barsky发现每个像素值位于颜色空间的一个仿射子空间中。在一种前景色对一种背景色的情形，这个子空间是一条直线——由RGB空间中前景色与背景色之间的所有颜色三元组构成。如果是一种前景色对两种背景色，则子空间是由三种颜色在RGB空间中的位置所决定的平面（除非三种颜色位于一条直线上，这种退化情形是所取点集不够一般的例子）。如果用一种前景色对三种背景色，那么子空间是整个RGB空间，除非四种颜色位于一个平面上（这是另一种退化的情形）。需要注意，上述分析只能到此为止：RGB空间中的任意五个点总是位于三维仿射空间中，因此构成一种退化情形。（如果颜色由n种谱样本表示，这里$n>3$，那么更复杂的情形可类似地处理）。

```
void LinearSoftFill (
        regionType region,
        color F,        /* 前景颜色 */
        color C,        /* 背景颜色 */
        color N)        /* 新的背景颜色 */
{
    int i;
    int d;

    /* 初始化部分 */
    求使 |F_i - C_i|over i = R, G, B最大的i;
    d = |F_i - C_i|;

    /* 内部测试 */
    for (each pixel) {
        color P = 该像素的颜色值;
        int t = (P_i - C_i) / d;
        if (t >(某个小值) {
            /* 设置像素颜色值 */
              将 P 替换为  tN + (1 - t) C;
        }
    }
}  /* LinearSoftFill */
```

图19-58 更健壮的软填充算法

因此我们只考虑一种前景色F对两种背景色C和D的情形。此时图像的每个像素值是如下的凸线性组合：

$$P = t * F + s * C + (1 - t - s) * D$$

要解决的问题是如何确定t和s的值，从而计算用新前景值N代替F后像素的颜色值：

$$P = t * N + s * C + (1 - t - s) * D$$

用R、G、B三种颜色分量代入上述第一个方程后得到有两个未知量的三个联立方程。与线性填充的情形一样，我们可以用其中任两个方程解出t和s。但是，如果向量$F - C$、$F - D$和$C - D$中的两个几乎平行时，由相应的两个方程得到的解会具有较大的舍入误差。做线性填充时另一个问题是线性系统可能无解。当F、C和D在颜色空间中共线时也会出现类似的问题。因此对一个有效的算法来说，颜色在RGB空间中的一般位置很重要（就是说，没有三种颜色位于颜色空间中的一条线上）。这一要求可从下面直观地看到。想像背景由红、粉红和白色的明暗色调组成，所有这些颜色值都是红色和白色的凸线性组合。现在如果用橙红色作为前景色填充图像，橙红色是粉红色加上少量的黄色。对于人来说，确定橙红色与粉红色的分界并不容易。算法做这样的区分也同样是件麻烦事。

我们以一个还没有文献提到过的推广来结束本节。如果用两种颜色对前景物体进行绘制，而背景由另两种颜色组成，可以用两种与上述不同的颜色来重新对前景物体上色。这样就可以把置于斑马背景图案上的红绿相间的棋盘的颜色改变为黄橙相间。这一机理与前述填充算法在四种颜色时的推广是等价的。假设两种前景色是E和F而背景色是C和D。解下面的方程：

$$P = rE + sF + tC + (1 - r - s - t)D$$

984 ~ 985

可以得到r、s和t的值（前提是E、F、C和D处于一般位置）。如果现在希望用M和N来分别代替E和F，只需简单地令

$$P = rM + sN + tC + (1 - r - s - t)D$$

下面是这一原理的一个运用。对红球在白光照射下的场景进行细腻的明暗绘制，所得球体的颜色将在红色和白色的组合之间平滑地变化。下面要把这幅图像合成到蓝绿相间的棋盘上。这样得到的结果应是绿色偏蓝的球体在蓝色偏绿的光照下的场景。我们可以用上面讨论过的方法来得到新的图像。注意，两次运用更基本的填充算法是无法得到预期结果的，因为一次替代后不再满足一般位置条件。

19.6 加速copyPixel

在所有扫描转换算法中，我们试图保持扫描线的相关性，因为在一条扫描线上同时复制多个像素（或在任何模式下写入像素）时，速度最快。在一位图形的情况下，也很容易一次复制整个字。处理字中个别位和整个字花费几乎一样。对于字构成的屏幕内存，逐个像素操作比逐个字慢n倍，假定字长为n位。（即使是8位的颜色，一个32位的处理器也能同时复制4个像素。）在第3章讨论这操作时，简单使用了SRGP_copyPixel，它的实现依赖于系统。一般地说，copyPixel 程序用到一位图像上时称为bitBlt（所谓位块传输）；这一节，我们接着Pike在MC68000微处理器上的快速bitBlt例程[PIKE84]，讨论bitBlt例程的优化。这里给出他的C代码。

我们想实现的bitBlt函数应支持对矩形的裁剪、任意写模式和纹理。纹理是bitBlt中的关键图案。窗口管理器能用纹理表示窗口未被激发。通过赋予窗口点画纹理，窗口管理器通知用户在使用窗口之前须首先激发。在bitBlt中做这种点画比在窗口中以点画图案重画所有图元快很多。所以，我们想定义一段程序如图19-59所示。

```
void bitBlt(
    bitmap source,          /* 源位图 */
    point pt,               /* 将被复制的区域的角点 */
    texture tex,
    bitmap destination,     /* 目标位图 */
    rectangle rect,         /* 目标区域的位置 */
    writeMode mode);
```

图19-59 bitBlt的过程声明

程序的被复制区域大小与*rect*一样，原点位于源位图的*pt*点。目标区域在destination（目标）位图上由*rect*确定。纹理是$w \times w$位列，w是机器字长。假定w为32，纹理实际上表示为一个32字的阵列。

程序的实现是直接的，但会遇到一些情况。首先要避免纹理全是1的情况。如果源矩形或目标矩形部分或全部位于相应位图之外，应避免涉及这些区域。两个位图可能相同，源矩形和目标矩形可能重叠，我们必须以不致造成破坏的顺序复制。最后，每种写模式都要求特别处理，因为其中一些很容易用简单的机器指令实现，而另一些要用多条指令，尤其对部分字进行操作时。

很幸运，C提供对处理器内存的直接存取，因为bitBlt涉及对单个位的操作，当然，越直接操作越能提高效率。在算法的第一个版本中，通过运用移位、位屏蔽、指针算术和指针比较，达到满意的速度——这对不提供内存直接存取的语言是不可能的。但是，即便这样也没快到能实用，BitBlt的最终版本必须用汇编语言实现。这显示了标准的设计折中：对时间关键系统的

内核，效率第一，而外围操作常常可少些效率多些易用性。

位图是基本的数据结构。为利用字操作，必须把位图排列成字阵列。另外我们还想创建大小不是32倍数的位图，因而，用三部分记录位图：一个指向字阵列的指针（见图19-60）；一个描述字阵列中位集合的矩形；一个整数宽度，它描述长方形一排字中有多少位。如果矩形的边与字列的边对齐，它就是以像素计算的长方形宽；否则，会大一些。图19-61给出基本bitBlt算法的伪代码。但是，实际的代码复杂得多。图19-62给出用C语言写的bitBlt代码。

986
~
987

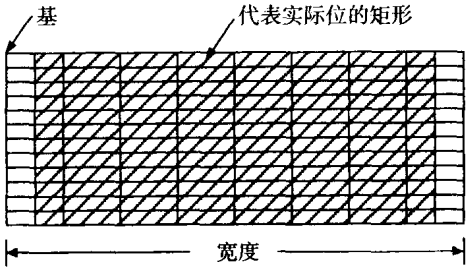


图19-60 位图是一个字的阵列，包含一个矩形，代表实际位图的二进制位

```
将矩形裁剪为源位图或目标位图；
找到以位为单位的矩形的宽度和高度；
if（两者都是负）
    从过程中返回
计算指向源矩形的第一位的指针p1；
计算指向目标矩形的第一位的指针p2;
if（在内存中p1在p2之前）{
    移动p1到源矩形中的低字；
    移动p2到目标矩形的低字；
    复制源矩形的行到目标矩形的行，自底向上；
}else
    复制源矩形的行到目标矩形的行，自顶向下；
```

图19-61 基本bitBlt算法的伪代码

```
typedef struct {                          /* 这不是第2章的rectangle类型 */
    point topLeft, bottomRight;
} rectangle;

typedef struct {
    char *base;                           /* 指向位图的存储区的第一个字的指针 */
    int width;
    rectangle rect;
} bitmap;

typedef struct {
    unsigned int bits:32;
} texture;

typedef struct {
    char *wordptr;
    int bit;
} bitPointer;

void bitBlt(
        bitmap map1,                      /* 源位图 */
        point point1;                     /* 将被复制的区域的角点 */
        texture tex;                      /* 在复制期间应用的纹理 */
        bitmap map2;                      /* 目标位图 */
        rectangle rect2;                  /* 目标区域的位置 */
        writeMode mode)
{
```

图19-62 bitBlt算法（略去了某些特殊情况）

```
int width;
int height;
bitPointer p1, p2;

/* 裁剪源矩形和目标矩形到它们的位图 */
clip x-values;                          /* 见后面的过程 */
clip y-values;
/* 以位为单位的区域的宽度和高度 */
width = rect2.bottomRight.x - rect2.topLeft.x;
height = rect2.bottomRight.y - rect2.topLeft.y;
if (width < 0 || height < 0)
    return;

p1.wordptr = map1.base;                 /* 源位图上的点 */
p1.bit = map1.rect.topLeft.x % 32;

/* 位图中的第1位是 */
/* 增加p1直到它指向第一个位图中指定的点 */
IncrementPointer (p1, point1.x - map1.rect.topLeft.x + map1.width *
                  (point1.y - map1.rect.topLeft.y));

/* 对p2也同样——它指向目标矩形的原点 */
p2.worldptr = map2.base;
p2.bit = map2.rect.topLeft.x % 32;
IncrementPointer (p2, rect2.topLeft.x - map2.rect.topLeft.x +
                  map2.width * (rect2.topLeft.y - map2.rect.topLeft.y));
if (p1 < p2) {
    /* 内存中点p1在p2之前；如果它们在同一位图中， */
    /* 源矩形的原点在目标矩形的原点的上边，或者， */
    IncrementPointer (p1, height * map1.width + width);
    /* 现在p1指向矩形的低位字 */
    IncrementPointer (p2, height * map1.width + width);
    /* 对p2也同样，但是对目标矩形 */
    point1.x += width;
    point1.y += height;
    /* 这个点现在正好超过矩形中的低位字 */
    while (height-- > 0) {
        /* 自底向下、自右向左将源矩形的行复制到目标矩形 */
        DecrementPointer (p1, map1.width);
        DecrementPointer (p2, map2.width);
        temp_y = point1.y % 32;         /* 用于索引纹理 */
        temp_x = point1.x % 32;
        /* 现在自右下向左上做实数bitBlt */
        RowBltNegative (p1, p2, width, BitRotate (tex[temp_y],temp_x),mode);
    }  /* while */
} else {      /* if p1 ≥ p2 */
    while (height-- > 0) {
        /* 自顶向下、自左向右将源矩形的行复制到目标矩形 */
        /* 自左上向右下做实数bitBlt */
        RowBltPositive(参数与前面相同)

            指针递增
        }  /* while */
```

图19-62（续）

```
    } /* else */
} /* bitBlt */

void ClipValues (bitmap *map1, bitmap *map2, point *point1, rectangle *rect2)
{
    if (*point1不在*map1内) {
        调整*point1在*map内
        以相同的量调整*rect2的原点
    }
    if (*rect2的原点不在*map2内) {
        调整*rect2的原点在*map2内
        以相同的量调整*point1
    }
    if (*rect2的对角点不在*map2内)
        调整*rect2的对角点在*map2内
    if (*map1中的相应矩形的对角点不在*map1内)
        调整矩形的对角点
} /* ClipValues */

void RowBltPositive(
        bitPtr p1, bitPtr p2,          /* 源和目标指针 */
        int n,                         /* 复制多少位 */
        char tword,                    /* 纹理字 */
        writeMode mode)                /* blt像素的模式 */
{
    /* 根据模式从位置p1到位置p2复制n位 */
    while (n-- > 0) {
        if (BitIsSet (tword, 32))      /* 如果纹理设问题复制…… */
            MoveBit (p1, p2, mode);    /* 然后复制该位 */
        IncrementPointer (p1);
        IncrementPointer (p2);
        RotateLeft (tword);            /* 旋转tword中的位至左 */
    } /* while */
} /* RowBltPositive */
```

图19-62（续）

Pike指出，C版本的bitBlt代码，在8-MHZ MC68000处理器上，水平卷动800 × 1024像素的位图，要花费大约8分钟。他提出几点改进。第一点是，通过构造和整个移动字，使用C和汇编语言中的字逻辑指令，减少对MoveBit的多次调用。这个变化提高速度16倍——同样的卷动只要30秒。

当在屏幕上bitBlt一大区域时，要调用MoveBit成百上千次。即使只移动字，也得调用这么多。迫切需要在这个内循环中运行尽可能少的代码。注意到，当目标和源位图有相同的偏移时，可避免大量的位移动。如果区域被复制到别处（典型的如屏幕），保持屏幕内、外位图相同的偏移能节省大量时间。

因为在屏幕上移动大区域是非常普遍的行为，加速这个过程非常有用。要做的就是检测这种特殊情况，间接地反复调用汇编指令移动32位字和递增指针，我们希望它非常快。

为有效地加速，必须考虑汇编语言。办法是考察要移动的区域和使用的模式，决定扫描是从左到右还是相反，从上到下或反之，位的旋转是否必要，等等。由此，可产生优化的汇编代码来执行实际的bitBlt运算。思想是汇编代码短而高效，能被调入内存快速执行。比如，前面

提到的屏幕滚动只需0.36秒——一个惊人的加速！用于滚动整个屏幕的汇编代码是一个简洁的杰作：它只包括8条指令（假定事先已设定各种寄存器）。

更复杂情况的代码会更精巧但同样高效。我们可这样设计bitBlt:用大量选择语句穷尽各种矩形、纹理、模式等的所有可能。实际上，整个程序能被写成这样形式的一个循环“对每一排，对每一字，执行以下操作：如果字是一个部分字，执行……；如果需要纹理，执行……；等等。”这种循环的缺点是所有情况都要检查，包括最简单的。所有带大量**if**和**switch**语句的汇编代码都不适合装入微小的指令内存。因为循环要执行许多次，将代码装入指令内存的代价远远超过实际bitBlt的代价。另一种方法，代码可写成大量嵌套**if**语句，其中每一可能情况都进行代码优化并作为决策树的一支。然后，一旦这种情况发生，那一小部分优化代码就被调入指令内存快速执行。这种方法的问题在于可能情况太多；Pike估计大约有2000种不同情况，平均每一情况约150个字节。这导致bitBlt代码达到1 MB，显然太大。因而，作为替代，在bitBlt程序中吸取执行循环的汇编代码。嵌套**if**语句中的每一分支在最终代码中仅执行自己的部分。当然，因为代码的吸取是在数据空间，而不是处理器的地址空间，必须通知指令内存一些代码不再有效，所以肯定要重新调入代码代替上次bitBlt执行时的代码。所有这些必须结合技巧以决定是区分情况还是直接处理（比如，bitBlt对于非常小的矩形——小于一整字）。

988 ~ 991

实现这样的系统也要求硬件支持。bitBlt中微代码实现的部分越多，其余部分越容易提高效率。

19.7 形状数据结构和形状代数

*形状数据结构*被发展起来以提高光栅操作（特别是裁剪）的效率[DONA88；GOSL89；STEI89]。它是欧几里得几何平面和光栅平面的结合。在这儿，形状是几何结构表示的区域的光栅近似。定义区域时，形状与位屏蔽比较，它的优点是双重的：形状是更小的数据结构，它的布尔操作很快。此外，形状的组织方式利用了扫描线相关性，因而形状的逐行处理很快。其他实现快速光栅操作的方法也发展起来，包括用行程长度编码[STEIN89]和四叉树系统[ATKI86]实现形状数据结构。

在给出形状的精确定义之前，让我们考虑一个例子。U形区域如图19-63a所示，能被分成几个矩形，如图19-63b所示。

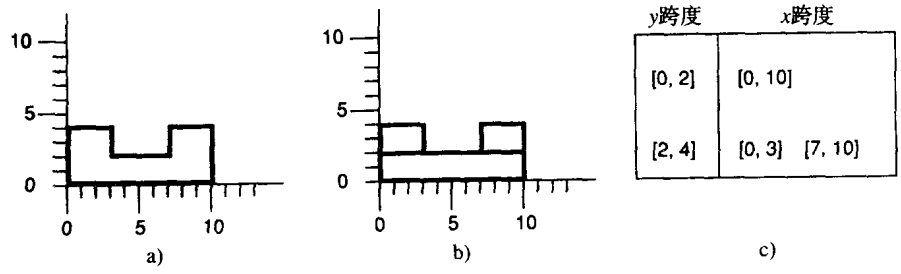


图19-63 a)一个区域，b)分割成矩形，c)该区域的形状数据结构

实际上，任何像素集合能分成平面上几个分立的矩形区域。在最极端的情况，例如，我们能用一个小正方形围住一个像素。形状数据结构把光栅平面上的区域描述为一系列矩形。更精确地说，一个形状由一系列y轴区间和一系列相应的x轴区间组成。每一对（y区间，x区间）代

表两个区间笛卡儿积形成的矩形。比如，图19-63a的区域能被分解成矩形[0,10] × [0,2],[0,3] × [2,4]和[7,10] × [2,4]，如图19-63b所示；它们能被储存为两组，对应于y区间[0,2]和[2,4]。第一组有x区间一个，第二组有两个，如图19-63c。然而，注意到虽然这种数据结构对矩形区域和如图19-63a中所示的简单区域非常有效，但对更一般的区域效率会降低。比如，一个实心圆，每一条水平扫描线一个矩形，结构就是一系列跨度。

如果我们要为扫描变换的图元创建形状，为画出图元的区域（如窗口）创建形状，我们发现裁剪后的图元的形状可用图元的形状的交表示。此外，为扫描转换的图元创建形状很容易，因为很多扫描转换算法基于一定的扫描线顺序。比如，第3章中的多边形扫描转换算法使用了活动边表，它扫描产生形状结构需要的水平跨度。相似的技巧应用于曲边区域。因为曲线可能是一个y值对应多个x值，根据y的属性把它们分割成具有对每个y值属性的段很重要，使得曲线段上只对应一个x值（如图19-64所示），然后用第3章中描述的活动边表这样的算法进行扫描转换。

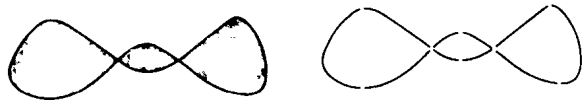


图19-64 在a)中所示区域的曲线轮廓线，在用于产生该区域的形状结构之前，必须分割成曲线段，如b)中所示

形状也用于产生连线或折线，避免了对同一像素重复**xor**操作。在这儿，整个形状的构造和绘图用**xor**模式一次完成。图19-65显示了直线钝角连接如何表示成形状的集合（见习题19.21）。

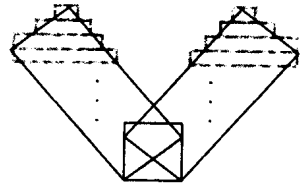


图19-65 用形状的合并做成两条线段的钝状连接

形状的一个主要优点是能用布尔操作进行组合（在15.10.3节，对直线上的区间进行布尔操作以实现光线跟踪中的CSG操作）。考虑如图19-66a中两个形状的相交。表中是形状结构，形状结构的交见图19-66b。怎样从原先的形状计算它们的交呢？

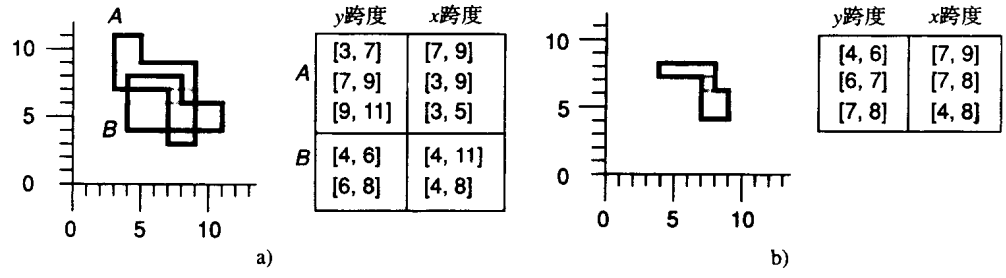


图19-66 两个形状的相交。a)两个形状和它们的相关数据结构，b)相交部分及其数据结构

注意到两个有重叠的矩形的交总是矩形。所以，我们能通过寻找有重叠的矩形计算两个形状的交。我们先检测形状y方向的重叠(若没有，则未相交)。然后，从每个形状的第一个y区间开始（y值最小的那个），比较y区间。如果重叠，比较x区间产生输出。否则，y值加1转到下一个y区间，重复。假定我们有一重叠函数，比较两y区间的重叠，返回指示重叠类型的代码，它的伪代码如图19-67所示。

```
void Intersect (shape shape1, shape shape2)
{
    int y11, y12;      /* y区间的端点在shape 1中 */
    int y21, y22;      /* y区间的端点在shape 2中 */
    int result;

    if ( 两个形状均不空 && 包围盒重叠 ) {
        y11 = Shape 1的最小y值;
        y12 = Shape 1中第一个y区间的另一端;
        same for y21, y22;

        while ( 在两个形状中仍有y区间 ) {
            result = Overlap (y11, y12, y21, y22);
            if ( 如果结果表明发生重叠 ) {
                y范围的输出重叠，由代码确定;
                如果有，输出重叠的x范围;
                如果没有x输出产生，删除刚输出的y重叠;
            }
            根据结果修改一个或两个y区间;
        }
    } else      /* if */
        什么都不做;
}  /* Intersect */
```

图19-67 求取两个形状相交的算法

x区间重叠的处理完全相同:如两x区间重叠，输出交，其中一个或两个区间都更新。所以，整个算法在于计算重叠以确定一结果代码，根据结果代码决定做什么。

如图19-68中的 x区间重叠的六种情况。第一种情况，如图19-68a所示，两线段分离，没有输出，更新过程包括取形状2的下一段。第二种情况一样，如图19-68b所示，不同的是取的下一段来自形状1。第三种情况，如图19-68c，形状1小值端与形状2大值端重叠。更新取形状2的下一个片段。剩下的情况相似。求和和差分的算法基于重叠结构的相同考虑（见习题19.22）。

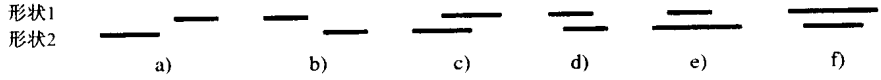


图19-68 两个区间覆盖的六种方式

形状代数可扩展为求交与填充结合的程序，用于在裁剪区域画图元。代替仅仅产生形状的x,y范围，我们输出矩形，将其画成实心的。

992
~
995

这种形状代数曾用于实现NeWS和X11/NeWS窗口系统。以下三种优化值得考虑。第一，在一些情况下，图元的形状表示是笨拙的（斜线是个好例子），并且，为计算图元与其他形状的交，我们发现对形状的每个矩形分别做裁剪更简单(Gosling建议这样处理线段[GOSL89])。第二，经过一些操作后，形状会变得破碎（做$(A-B)\cup B$），有必要在一些操作后聚合形状。第三，在窗口管理中，值得写一个特殊的求交和填充程序，在一个操作中，完成计算图元与窗口的交，选择应用于结果的图案，画出结果。所以，代替产生形状数据结构，求交程序产生的跨度直接输入绘图例程。

19.8 用bitBlt管理窗口

在一篇开创性的文章中，Pike[PIKE83]用bitBlt程序在位图屏幕上管理窗口。它的重要特点

是窗口的重叠和刷新与窗口内容无关，非窗口内存使用不大(大约一个屏幕的外部内存)，并且甚至在窗口部分或全部隐藏时也能绘制窗口。尽管重叠窗口在Smalltalk语言中已被引入，这篇文章首次展示了如何在支持bitBlt的机器上实现它们。这篇文章的思想现在应用于XWindows系统、Lisa操作系统、Macintosh操作系统和其他操作系统。

隐藏窗口的屏幕外管理在一些系统中还在使用，如我们在第10章讨论的(在X Windows系统中叫备份存储)，但代价巨大:当窗口数量增加时，备份存储的数量也在增加。需要考虑机器物理内存溢出的后果。如果窗口系统应用程序的编写者知道窗口系统提供的是“虚拟窗口”，那么即使窗口隐藏，应用程序也能写。当窗口移到屏幕上部时，窗口系统一定要刷新屏幕，因而不管屏幕其他部分怎样，窗口系统必须提供这个特性。因为应用程序窗口可能被一些点覆盖，窗口系统必须在窗口创建时分配足够的内存提供整个窗口的备份存储。如果同时运行的几个应用窗口每一个都这样，代价非常昂贵。

由于这个原因，一些窗口系统（如X Window系统）提供备份存储作为选项，也允许窗口不备份，它必须使用不同的更新策略，像第10章讨论的。当部分或全部隐藏的窗口重新出现时，窗口管理器通知应用程序哪些部分需用刷新，应用程序重新生成那些部分。典型地，它重画任何与新出现的区域相交的图元。这样一个系统的优点是应用程序可决定运行时需要多少内存(于是窗口系统不会通知应用程序，没有更多内存储存位图的屏幕外部分)，存储屏幕外部分的 996
时间在没必要时可省去。这样的代价是，每一个应用程序必须知道现在在画哪个窗口，而且应用程序的编写者不能假装认为窗口完全暴露。

然而，假定我们为每个窗口做备份存储，也必须以一种有效地方式实现各种窗口操作，如第10章讨论的。它们包括窗口的创建、删除、暴露、隐藏以及画入窗口。这节的剩余部分描述了Pike完成这些操作的方法。

表示窗口的数据结构包括一个矩形和一个字阵列（如19.6节的位图记录）以及三个附加部分：两个窗口指针，分别指向窗口前后；一个指针指向隐藏表列，表示窗口隐藏部分的位图的链表。前两个指针帮助决定窗口的可见或隐藏，同时那个表帮助决定窗口的哪部分可见或隐藏。窗口的顺序是不完全的，因而前后指针的选择也未完全决定。一旦选择不明确，它就是任意的。

隐藏一个窗口就是找到在它后面的所有窗口，依次放到它前面。这样做仅要求清除屏幕空间，唤起每个窗口的应用程序修补破损或做更多工作，如果窗口实际上在隐藏表保存了信息。比如，如果窗口程序要执行刷新任务，当bitBlt当前窗口的新隐藏部分到隐藏表时，隐藏涉及bitBlt新暴露窗口的隐藏部分到屏幕。幸运地是，窗口*A*遮盖窗口*B*的大小与窗口*B*遮盖窗口*A*的大小一样，没有必要分配新的内存。暴露一个窗口与此类似，移动一个窗口涉及暴露或隐藏新的部分，做bitBlt移动窗口的可见部分。当然，假设，正好有一块窗口*B*遮盖窗口*A*（反之亦然）大小的内存，它要求很好划分遮挡表：每个遮挡区域的边界长方形必须位于与之相交的窗口之内。

另一个要考虑的操作是画入窗口。在图19-69中，有两个重叠窗口。在底部窗口绘图（如图中的阴影区域)，图的一部分画到窗口的可见区域，另一部分画到隐藏的矩形中。这是个一般操作的例子:在每一个目标位图区域做一些操作。这样一个操作可以是清除一个矩形或画一条线。它能被精简到一个递归程序中，对目标区域和窗口的交集执行操作，然后在窗口遮挡表中的每一项调用自身。

注意到，窗口数据结构的设计包括一个关于矩形的选择：每个矩形用它的左边、底边和左下角定义，如第3章。所以，两个毗邻的矩形没有共同点。这个选择大大简化了许多操作，因为它有可能使遮挡表中的每个矩形完全位于与之相交的窗口中。如果矩形包括所有边，就不可能满足条件。

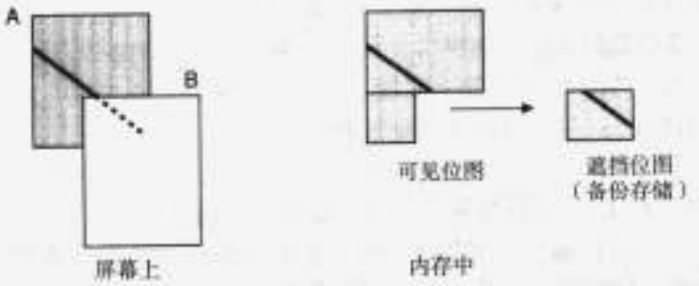


图19-69 向部分受到遮挡的显示平面绘图需要在可见部分和遮挡部分同时绘图。我们把要画的内容分割为两个部分，然后在适当的位图中画出适当部分

最初的窗口管理“层”模型被大大扩展。加入了一些特征，包括颜色。颜色加进来后，屏幕的图像复制变得复杂起来。每个像素用位的集合表示，而不是一位，传输像素的所有位要花很长时间。比如，假设红、绿、蓝各用4位显示。我们能想像这需要传输12个平面的位值给屏幕，并分别应用bitBlt操作。因为一次只处理一个平面，这个策略叫作*平面串行方法*。它的优点是很容易扩展为窗口管理软件位图版本。但有两个严重缺点:(在这个例子中)速度慢了12倍，简直是灾难性的表现；当大量平面传输时，窗口看起来很奇怪。比如，当所有红色平面处理过而绿、蓝平面未处理时，窗口看起来像蓝-绿色的旧材料与红色的新材料重叠在一起。这种效果如果持续时间长到能被看到，是很让人扫兴的。在颜色表系统中，结果更古怪:在平面的传输过程中，颜色索引会改变次序导致很糟糕的表现。另一可选方法是*平面并行方法*，所有像素图平面同时复制(可能用专用硬件)。因为显存和主存的组织有些不一样(虽然Pike的文章假设它们一样)，关键是，对给定的源和目标，用最有效的办法进行块传输。这项要求反过来要求块传输操作实现在可能的最低层——用汇编语言或机器语言，甚至硬件。

19.9 页面描述语言

*页面描述语言*基本上是一个图形软件包，用于隔离应用编写者和与机器有关的打印机细节，并且，在出版业中，帮助排版打印页面。页面描述语言不同于一般图形包：它只有输出，是2D图形包（虽然一些语言的3D扩展在发展中），对曲线和文本有广泛支持，它支持样本图像作为一级图元。更重要的是，它是代替子程序包的一种语言。因而，页面描述语言的解释器可装到打印机中；于是，短程序代替大量的像素数据送入打印机。此外，作为可交换格式，页面描述语言比子程序包更易用。一系列子程序调用用不同语言编写；传递调用顺序到另一设备需要翻译成新的语言。用页面描述语言编写的程序能传输到任何支持它的设备。因而，页面描述语言取消了7.11.3节讨论的元文件的概念。

997
|
998

最有名的页面描述语言是PostScript，这种语言的最初目的是描述位图页（主要生成于高分辨率打印机）。PostScript现在也被作为屏幕描述语言使用。页面描述语言很适合这项工作，特别是在客户-服务器窗口管理中，下载一个PostScript程序能减少服务器的网络拥挤和客户端的费用。如第10章讨论的，要在基于PostScript的窗口管理中激发一个对话框可下载一个显示对话框的PostScript程序，然后激活这个程序。再次启动对话框只须再次激活程序。

我们还有兴趣从PostScript和Interpress[ISO]产生国际或国家标准页面描述语言。这样一个语言能为出版业和计算机业提供标准格式。

在这一节中，我们将对PostScript的方方面面进行介绍，使你感受到页面描述语言是怎样工作的。页面描述语言的*成像模式*是在2D平面上的抽象行为的定义。在PostScript中，成像模式

基于在平面用不透明笔画图的概念。画笔是用户自定义宽度的笔和刷子，一些PostScript操作符控制画笔的位置。成像模式实际上是通过一个反映页面内容的大光栅内存来实现的；光栅内存的内容最后传输给打印机的输出页。

```
10.5  11.3  moveto
40    53.6  lineto
showpage
```

PostScript语言有三部分：语法、语义和绘制。语义和绘制间的差别很微妙。PostScript程序意思是“移动画笔，不画图，到页面位置（10.5,11.3），画线到(40,53.6)，显示结果。”这个句子是语义的一个例子。绘制是特殊图形设备的实际输出的产生。在激光打印机上，这个程序会在一页纸上打5000个小黑点；在位图屏幕上，会使87个像素变黑。所以，我们是在位图设备的驱动水平上调用PostScript绘制的。

PostScript的语法很简单。它是一个后缀解释器，操作数被推入堆栈，然后操作符对操作数（弹出堆栈）进行处理，把一些结果放回堆栈。所以，在前面例子中，操作数10.5和11.3被推进堆 999
栈，**moveto**操作符将它们弹出堆栈使用。数据类型支持数、数组、串、关联表（即字典）。（更精确地说，只有一种数据类——对象；每个对象都有类型、一些属性和一个值。类型可以是“整数”、“实数”、“操作符”等等。）关联表用以存储各种类型的定义，包括操作符定义。控制流构件包括条件、循环和过程。所以，典型的产生PostScript输出的应用程序是(1)产生一长串标准PostScript调用或(2)定义与自己需求密切相关的一个过程集（叫作序言），然后产生对这些过程的调用集。

PostScript也包括上下文的概念，它能存入堆栈，所以PostScript中的状态在操作前能被存储，随后恢复。定义自己过序的应用不必担心名字冲突，只要恢复到PostScript的初始状态。

PostScript的语义要复杂些。基本实体是图形实体和作用于其上的操作符。所有图形实体都被认为一样，于是，线、圆弧、三次曲线、点、一串文字和一个样本图像都能用同一操作符平移、旋转和缩放。各种操作符用于支持多个坐标系，于是对象能在一个坐标系中创建，并通过仿射变换到其他坐标系。一些操作符能检测PostScript的运行环境，这对制作与设备无关的图像很重要。因而，通过询问环境决定当前设备每英寸的像素数，我们能编写PostScript程序产生1 × 1英寸的正方形，不考虑使用设备的分辨率（除非它的像素间距有1.5英寸宽！）。

PostScript操作符分为六类[ADOB85b]：

1) 图形状态操作符。这些操作符能影响定义PostScript世界的当前属性的对象的集合，比如说像当前线宽、当前裁剪区域。

2) 坐标系操作符和变换。这些操作符用于改变那些定义了更多对象的坐标系。特别是，它们改变从PostScript坐标到输出设备坐标的映射。

3) 路径构造操作符。这些操作符用于定义和更新另一个称为当前路径的图形状态实体。它们开始一条路径，在路径上加入直线或弧，或结束路径（比如用一条直线段连接首尾）。当前路径是抽象实体——它不在页面上绘制除非激发了一些画图操作符。

4) 画图操作符。这些“绘制”操作符在光栅内存中产生数据，决定打印页上最终出现哪个点。所有画图操作符都针对当前路径。如果路径构造操作符定义了一个掩码，则画图操作符可看作在光栅内存的掩码所允许的位置。 1000

5) 字符和字体操作符。这类操作符用于指定、改变、选择字体。因为字体和其他图形实体一样，把文本放入当前“路径”的操作符是路径构造操作符；它们被作为特殊的一类仅仅因为字体的特殊性。

6) 设备-安装和输出操作符。安装操作符建立光栅内存与输出设备间的对应。输出设备控制从内存到设备的数据传输。

为展示PostScript模型的强大，我们给出使用几个操作符的例子。每个例子包括用“%”作注释分隔符的运行注释。

第一个例子，如图19-70所示，在页面上产生一些文本，如图19-71所示。**show**操作符是很特殊的操作符，它既定义要显示的对象（文本串）又把对象绘制到光栅内存。在所有例子中，页面边界都用方框表示，尽管画方框的指令未包括在例子中。在名称前面的斜线（/）把名称作为文字对象推入堆栈。

```
/Helvetica findfont      % 找出前面定位的Helvetica
                         % 字体对象。
188 scalefont            % 使其188倍大小——默认字体
                         % 大小为一个点。
setfont                  % 做这个当前字体（作为“图形状态”
                         % 的一部分存储）。
100 20 moveto            % 移到该页上的一个点。
67 rotate                % 坐标旋转67°。
(Hello) show             % 将“Hello”放在当前点(100, 20)
                         % 并在该页上绘制它。
```

图19-70 一个PostScript 程序

在图19-72中，我们建立一个矩形和它的标签，如图19-73所示，用虚线画出。我们用路径构造操作符建立“当前路径”。第一个路径构造操作符是**lineto**操作符，它从当前点到特定点画线。第二个是**charpath**操作符，它有两个参数，即文本串和布尔量；如果布尔量为**false**，文本串的轮廓用当前字体加入到当前路径，当布尔量为**true**，文本串的轮廓转换成一种特殊路径以便裁剪。（我们将在第三个例子中看到文本轮廓。）**stroke**操作符用于绘制当前路径和开始新路径。（当前路径也能用**fill**或**eofill**操作符绘制。它们都要求封闭路径。第一次填充遵守非零旋转原则；第二次填充根据奇偶填充原则。）

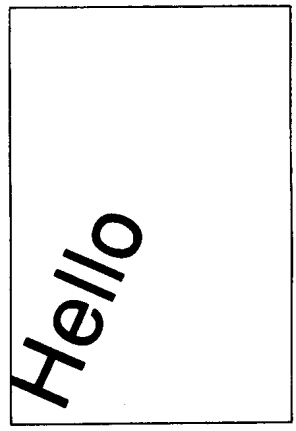


图19-71 第一个PostScript 程序的输出

```
/Helvetica findfont      % 找出前面定义的名为“Helvetica”
                         % 的对象。
120 scalefont            % 使它120倍大小——默认字体
                         % 大小为一个点。
setfont                  % 做这个当前字体（作为“图形状态”
                         % 的一部分存储）。
0 0 moveto               % 移到该页上的一个点。
newpath                  % 开始路径构造。
100 300 moveto           % 在位置(100, 300)开始。
500 300 lineto           % 加一条线到位置(500, 300)。
```

图19-72 一个较复杂的PostScript 程序

```
500 800 lineto                  % 继续加线段。
100 800 lineto
100 300 lineto                  % 我们已经构造了一个矩形。
100 50 moveto                   % 向下移动矩形。
(Rectangle) false charpath
                                % 增加文本“Rectangle”
                                % 到当前路径。
[9 3] 0 setdash                 % 设置虚线图案为9个亮3个灭。
stroke                          % 用这种虚线图案画出当前路径。
```

图19-72（续）

图19-74显示如何裁剪文本，结果如图19-75所示。裁剪文本需要使用裁剪路径，另一种PostScript图形状态一部分。初始时，裁剪路径包括所有东西。然后，我们就能定义简短的裁剪路径来创建裁剪过的对象。这里，我们绘制三个水平带，并且在两个不同裁剪区域显示：三次曲线，它用**closepath**操作符封闭，随后是一大片文本。因为**clip**操作符总是根据当前路径与当前裁剪区域的交减少裁剪区域，这对存储最初裁剪区域以便恢复它很重要。我们用**gsave**和**grestore**操作符完成整个图形状态的存储和恢复。

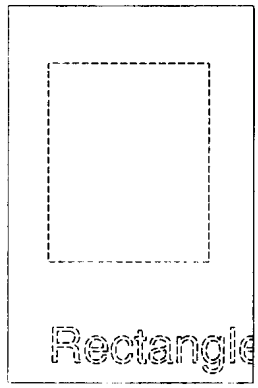


图19-73 第二个PostScript 程序的输出

```
/rect {                         % 定义一个新的操作符画
                                % 一个其宽和高都在堆栈中的矩形。
                                % 它通过w h rect调用。
  /h exch def                   % 定义h为高，第二个参数，
                                % 它在堆栈中。
  /w exch def                   % 定义w为来自堆栈的
                                % 第一个参数。
  w 0 rlineto                   % 通过以(w, 0)移动画笔画一条直线。
  0 h rlineto                   % 然后移动(0, h)画一条垂直直线。
  w neg 0 rlineto               % 把w压入堆栈中，求反，
                                % 把0压入堆栈中并移动( - w, 0)
                                % 画一条直线。
  0 h neg rlineto               % 对h也同样，封闭盒子。
  gsave                         % 保存图形状态，
  fill                          % 包括裁剪路径。填充当前路径，
                                % 减小裁剪区域为矩形。
  grestore                      % 恢复原始裁剪路径。
  newpath                       % 抛弃当前路径重新开始。
} def                           % 这结束了rect的定义。

/stripes {                      % 定义“stripes”操作符，
                                % 它在页面上画三个条，
  newpath                       % 它没有参数。
  100 300 moveto                % 移到屏幕上的点。
```

图19-74 一个复杂的PostScript 程序

```
  800 50 rect                    % 画矩形。
  100 200 moveto                 % 向下移动一点，再画矩形。
  800 50 rect
  100 100 moveto
  800 50 rect                    % 再画矩形。
} def                            % 这结束了stripes的定义。

0 0 moveto                       % 在区域的当前点开始。
.95 setgray                      % 设置灰度级非常浅。
stripes                          % 画示整个条。
.4 setgray                       % 设置灰度稍暗一些。
gsave                            % 保存当前图形状态。
                                 % (including the clip path).
newpath                          % 在50, 150开始一条新的路径。
50 150 moveto
100 250 300 275 250 175
curveto                          % 以控制点
                                 % (50, 150), (100, 250), (300, 275), (250, 175)
                                 % 画Bezier曲线。
closepath                        % 用直线封闭曲线。
clip                             % 新的裁剪区域
                                 % 是旧的裁剪区域
                                 % 与刚构建的路径的交
stripes                          % 画通过新裁剪区域的条状，
                                 % 比上一次稍暗一些。
grestore                         % 恢复原始裁剪路径。
.2 setgray                       % 颜色进一步变暗。
gsave
/Helvetica findfont
100 scalefont setfont            % 创建巨型Helvetica字体。
newpath                          % 开始一条新的路径。
200 80 moveto                    % 准备写一个字符。
(ABC) true charpath              % 创建一条从文本的轮廓线
                                 % 开始的路径。
closepath                        % 封闭此路径，
eoclip                           % 并使其成为新的裁剪路径，
                                 % 使用奇偶规则。
stripes                          % 画出经过它的条形。
grestore
```

图19-74（续）

过程的定义有点神奇，**def**操作符认为堆栈中有两个参数：一个名称和一个定义。当定义过程时，名称推入堆栈（前面的斜线以免解释），然后是过程体本身（只是一项——所有内容在括号内）。**def**操作符弹出这两个操作数，给过程体定义名称。

通常，当调用过程时，它的操作数还在堆栈中，则**def**的两个操作数顺序相反。为把这些操作数和局部变量绑定在一起，我们把变量名推入堆栈，调用**exch**操作符，交换堆栈的头两个元素，然后调用**def**。



图19-75 PostScript允许一般的裁剪，用文本的轮廓作为裁剪路径，如第三个PostScript程序的输出所示

这些例子很好地展示了PostScript模型的强大。PostScript本质上像汇编语言——功能强大但编程不方便。另一方面，定义过程很简单，使得PostScript容易使用。实际上，最好在序言形式中使用PostScript，各种图形状态属性被设定，各种过程被定义，随后是一系列对定义好的过程的调用。

在例子中使用的当前路径是一个常用对象。它能从几种方式得到：**stroke**操作符把当前路径当作直线和曲线的序列来绘制，它的粗细、图案取决于图形状态；**fill**操作符用当前填充图案填充当前路径（必须是封闭路径才有意义）；**eofill**操作符按照奇偶原则填充。在例子中，定义字符轮廓的曲线能作为路径的一部分。

PostScript也支持“图像”作为图元，它可以是每像素1位表示或每像素n位表示。当图像在光栅内存中绘制时，它像其他图元一样；光栅内存中图像覆盖的每一个像素可用“颜料”画上白、黑或任何颜色。因而，调用**image**操作符可在任何存在的图形上画图（它不会混合或组合成17.6节描述的形式）。另一种形式**imagemask**，适合于1位图像。这种掩码仅画图像的1处；0处光栅内存未变。因而，它以**transparent**模式传输图像很有效，除非图像在操作前会变形。实际上，一幅图像可以被缩放和旋转，并使用任何其他操作（包括用当前裁剪路径裁剪）。

最后，我们提出一个重要问题：PostScript解释器是怎样实现的？比如说，我们怎样画粗的三次曲线或旋转的图像？准确的解释没有写在《PostScript语言参考手册》[ADOB85b]中，因为它的实现对不同类的打印机（和显示）应该不同。还有，成像模式的一些方面应清晰给出。曲线被分割成小线段画出。这些线段粗细程度适中，绘制后产生粗曲线。因而，粗线如果很宽在它的两边有凹陷，多边形化的曲线是粗糙的（这也由用户定义）。注意到，在PostScript中，线宽是一几何属性而不是修饰属性，即它在一些坐标系中定义，以后进行变换。如果我们设定线宽和当前缩放变换系数为2，则所有直线会粗一倍。

1001
~
1005

PostScript图像很明显用仿射映射进行变换，使用的技巧类似于两重变换。在成像模式中定义一个图像，如一小正方形栅格，于是绘制变换后的图形就是绘制许多小平行四边形。任意路径裁剪的方法没有清晰表达，但好像一些应用中只使用了简单跨度算术：裁剪区域由一组水平跨度表示，每个图元按行扫描变换，仅仅跨度内那些像素被绘制。大量剩余的执行任务是记录保存；当前裁剪区域、当前路径、各种填充方式、光栅内存和当前变换矩阵都需要保存。

19.10 小结

我们讨论了一些现在使用的几何、光栅算法和数据结构，以及用它们实现各种光栅软件包，包括页面描述语言。尽管这些算法中的一些相当复杂，但有关这些主题的工作仍在进行。几何裁剪（它的3D扩充用于多面体构造实体几何）是一个活跃的研究主题，新的扫描变换算法，特别是结合反走样，还经常出现在文献中。

从这些算法中得到的经验是在，很多情况下简单更重要。截剪是一广泛使用的裁剪技巧，形状数据结构使许多操作简单，bitBlt的实现思想是使在内存中运行的代码简单短小。

习题

19.1 通过尽可能重复使用中间结果，改进Nicholl-Lee-Nicholl线裁剪算法。实现所有情况。

19.2 梁友栋-Barsky多边形裁剪算法会产生退化边（第3章的Sutherland-Hodgman算法也如此）。给出一个算法，把上述裁剪算法生成的边组成一个表，并“取消”任意相继的两条方向相反的边，递归使用这一过程。注意到这样的边对要么是竖直的要么是水平的，算法可

以变得更简单。解释原因。注意除了比较外不能用别的算术运算。

再给出一个算法，在必要时把得到的多边形分裂成多个输出多边形，使得没有两个输出多边形相交。这可以通过对输出多边形使用第一个算法，然后扫描所得结果，找出其中封闭部分。测试一下你的算法，把该算法用于几个这样的多边形：反复从一个顶点区域到相邻或相对的顶点区域。

19.3 不使用无穷大，把竖直和水平线当成特殊情况处理来重新实现梁友栋-Barsky多边形裁剪算法。

1006

19.4 写一个算法，把通过顶点表指定的平面多边形转换成Weiler多边形裁剪算法中描述的轮廓线所需的数据结构。推广你的算法，使之能处理由多条路径指定的带洞的多边形。

19.5 假定给定一个集合及其上的一个偏序（称为包含于）：对任意两元素A和B，要么A包含于B，B包含于A，或者A不包含于B且B也不包含于A。（这里的偏序与集合论中的element-of关系无关。）

设计一个算法构造一棵二叉树，其节点以集合中的元素标记，且具有如下两个性质：一个节点的左子节点总是包含于该节点中，而该点与右子节点则不存在包含关系。一个构造这种树的好算法可改善Weiler多边形算法的后处理步骤，在这一步需要确定轮廓线的被包含/重合结构。

19.6 以（Round(x_0),Round(y_0)）作为实际端点为(x_0, y_0)的直线段的整数起始点可以得到与文中不同的一种选择。哪种更好，为什么？

19.7 假定允许线段端点的分辨率为四分之一像素并考虑从(0, 0)到(40, 1)线段。考虑线段从$x = 15$到$x = 40$的部分。请说明下面两种方法会得到不同的结果：方法一是先裁剪，然后用裁剪后线段的端点（近似到整数的四分之一）计算直线方程，再扫描转换；方法二是先裁剪，然后用裁剪后线段的端点四舍五入取整作为起始点，再以原始直线方程做扫描转换。

19.8 按下面的方法完成19.2.6节给出的椭圆描述。设

$$\begin{pmatrix} x \\ y \end{pmatrix} = \cos(t) \begin{pmatrix} P_x \\ P_y \end{pmatrix} + \sin(t) \begin{pmatrix} Q_x \\ Q_y \end{pmatrix}$$

解此方程把$\cos(t)$和$\sin(t)$分别表示为x, y, P_x, P_y, Q_x, Q_y的表达式。利用恒等式$\cos^2(t) + \sin^2(t) = 1$建立一个关于$x, y, P_x, P_y, Q_x, Q_y$的方程并把此方程写成$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$的形式。从而导出系数$A, B, C, D, E, F$关于$P_x, P_y, Q_x$和$Q_y$的表达式，其中$D$和$E$的结果应为0。

19.9 给出描述双曲线和抛物线的方法及根据此描述计算圆锥曲线系数的方法。描述双曲线的一种方式是给定一个平行四边形。平行四边形的长对角线作为双曲线的轴，双曲线的两个顶点位于对角线的两端（顶点是指双曲线两个分支上相距最近的两点），其渐近线平行于平行四边形的边。一条抛物线则可用一个焦点（一点）、一条准线（一线）和抛物线上的一点来确定，但这不是很直观。你能想出更好的方法吗？

19.10 计算Van Aken圆锥曲线算法的增量表。这是画每个八分弧过程中沿平直或对角运动时判定变量的增量。证明，用于处理第一和第二八分圆的代码实际上可以用来画任一个奇数或偶数的八分圆。

19.11 证明，当半径和圆心均不是整数时，圆的三种误差度量方法导致相同的像素选择。如果假定所考虑的像素满足$x \geq 0$, $y \geq x$，则问题最为简单。（由对称性，所有其他情形都可由

这种情形导出）。半径的误差度量根据(x, y)和$(x, y-1)$两点关于$F(x, y) = x^2 + y^2 - r^2$的函数值选择其中一个函数值的绝对值较小的点。轴误差度量则用$y' = \sqrt{r^2 - x^2}$；同时计算$y - y'$和$(y-1) - y'$并择其中绝对值较小的点。中点误差度量也从(x, y)和$(x, y-1)$二者中择一，不过做何种择取决于$F(x, y-0.5)$是负还是正。做什么样的选择才能保证三种情形在临界情况下（三种临界情况分别是$F(x, y) = F(x, y-1)$，其中$y' = y - 0.5$且$F(x, y-0.5) = 0$）是一致的?或者没有一致的选择？至少说明，在临界情况下，存在某个像素使得三种误差度量都达到最小。 1007

19.12 采用任一种算法对半径为$\sqrt{7}$，圆心在原点的圆进行扫描转换。考察下面的问题。结果从美学角度看令人满意吗？把圆心平移到(1/2, 0)会发生什么情况？扫描转换结果中的尖锐角是否很令人失望？圆算法是否应该使曲率为常量而不是最小化那些距离？这个圆和其他几个半径平方为整数的圆一样特别的棘手。McIlroy[MCIL83]证明下一个带这种尖锐角的圆的半径是$r^2 = 7141$。

19.13 增加完成最后画八分弧的部分，完善Van Aken圆锥曲线算法。

19.14 通过增加八分弧跳跃检查，改进Van Aken圆锥曲线算法。DaSilva[DASI88]建议使用判定变量决定下一个要画的像素，然后看那个像素是否跳过了一个八分弧。如果是则选择另一个像素。

19.15 写一个一般的画线算法。你的算法应能处理端点坐标为具有固定分母p的有理数的线段。确认画共享一个端点的两条直线段时，光栅化后的结果在此端点处也有相同的像素。初始化判定变量可能是本算法最困难的部分。

19.16 通过考察从(0, 0)到(4, 2)和从(4, 2)到(0, 0)的两条直线段，说明第3章的中点算法由于画出方向的不同，可能会得到不同的像素。找到一种方法检查直线段端点的坐标并调整中点算法中的比较操作(从小于到小于或等于)使得不管从线段的哪一个端点开始算法都画出相同的像素。

19.17 创建一种画线算法画出由端点确定的直线段中位于两竖直线间的部分。也就是说，你的过程看起来像TrimmedLine (point *start*, point *end*; **int** *xmin*, **int** *xmax*)，而画*start*和*end*之间的线段中位于*xmin*, *xmax*之间的部分。先用截剪再用解析裁剪实现此算法。做解析裁剪时，记住从原始线段而不是裁剪后得到线段端点的舍入取整的结果导出判定变量和增量。

19.18 解释为什么难于用前向差分技术画严重弯曲的曲线。作为反走样的一种形式来解释这个问题。

19.19 通过一个完整的例子，跟踪递归填充算法加深对算法的理解。哪些是算法中较为精巧的情形？

19.20 实现递归填充算法。实现它，然后在一个常规的光标可存取的视频终端上执行它，用字符表示不同的像素值。实现边界填充和泛填充两种算法。

19.21 说明如何用形状代数构造两直线段间的斜面接口；也就是说，通过描述形状的并、交和差最终产生斜面接口。对于裁剪斜面接合，情况相同吗？

19.22 通过描述名为“Overlap”的函数的高级算法及其返回值，给出确定两形状的差和并的算法。

19.23 实现文中填充算法的改进。如果按先进先出而不是后进先出的方式处理堆栈会发生什么情况？你能够想到一种帧缓存体系结构以分析第一种方法是好还是坏吗？ 1008

19.24 考虑由方程$(x-100)^2 + y^2 = 10000.9801$和$(x+100)^2 + y^2 = 10000.9801$确定的两个填充

圆。证明：如果我们扫描转换这两个圆，两者都会生成像素(0, 2)，所以(0, 2)包含在两者的形状数据结构中。另一方面，请证明：这两个圆的交只包含$y = -1$到$y = 1$之间的点，因此它们不应都包含(0, 2)像素。尽管有这样明显的矛盾，形状代数是实现求交的一种好方法吗？解释在什么情况下可以接受形状求交作为几何图元实际求交的合理逼近。

19.25 在19.1节中，我们谈到判定一个给定点位于一条射线的哪一面。假设我们有一条从P到Q的射线和一个点R。

a. 证明从P到Q的射线绕P逆时针旋转90°得到的射线由$(-Q_y + P_y, Q_x - P_x)$给出。

b. 解释为什么仅当P到R的向量与(a)中给出的向量位于同一个半面时R位于P到Q射线的左侧。

c. 当且仅当它们的内积为正时两个向量位于同一个半平面。用这一结论证明当且仅当

$$(R_x - P_x)(-Q_y + P_y) + (R_y - P_y)(Q_x - P_x)$$

为正时，R位于P到Q射线的左侧。

19.26 实现19.3.2节中圆的较简单的反走样算法。你需要预先做些计算

a. 对足够大的R和充分小的s，证明：位于圆外到半径为R的圆距离为s的点(x,y)的残差值$F(x, y) = x^2 + y^2 - R^2$近似等于$2Rs$。（提示：称一个点从圆外到圆的距离为$s$是指点到圆心的距离为$R + s$。）

b. 计算半径为1的圆盘与半平面的加权重叠值表。该表有32项，对应于到-1和1像素之间点的距离（距离为1意指像素中心位于半平面内部一个单元）。

c. 现在可以按如下描述完成算法。对半径为R的圆附近的每个像素（使用类似于中点算法的方法确定这些像素），计算残差并除以$2R$得到距离。用这个值对重叠值表进行索引；再用距离减1对重叠值表索引，把两个索引值相减求得到覆盖值。

d. 对各种大小的圆试用你的算法，并对用于小圆时的性能进行评估。

19.27 假设给定平面上的一个圆，方程为$(x - h)^2 + (y - k)^2 - R^2 = 0$。描述如何检查一个点$(x, y)$是否位于圆内。再描述如何判定一个点是否位于到圆的距离为1的范围内。对椭圆$Ax^2 + 2Bxy + Cy^2 + Dx + Ey + F = 0$回答两个同样的问题。由于距离公式比较复杂，这种情形下第二个问题更困难些。假如你采用下面的方法来计算点(x, y)到椭圆的距离：以点(x, y)处的残差乘以$(AC - B^2)/F^2$并检查所得结果是否介于0.99和1.01之间。解释为什么这可以作为是否靠近椭圆的一个合理的度量。

19.28 a. 考虑x轴上介于$-1/2$和$1/2$的点集以及y轴上介于$-1/2$和$1/2$的点集，对每个这样的区间，我们都只包含两个端点中的一个。因为这是一个类似于“+”号的形状，我们把它称为形状P。证明：如果图元与P相交，那么采用已讨论过的中点算法将会准确地画出(0, 0)处的像素。（注意P只包含四个可能端点中的两个。判定到底包含哪两个端点归结为如下的计算：水平线贡献左或右像素时精确的中点交，类似地计算竖直中点交。确定你对P的端点的选择与你正在研究的算法相一致。）

1009

b. 类似地，采用中点算法也会精确地画出像素Q，如果图元与中心平移到Q后的形状P相交。以此证明半径为0.49，中心为（0.5, 0.5）的圆不会画出任何像素。

c. 除了形状P外人们还提出另一些形状，包括P的凸包和像素的正方形包围盒等。试评价一下这些选择。

19.29 用于绘制宽曲线的Posch-Fellner风格的算法可以稍做改进。不是考虑沿曲线拉伸圆而是拉伸一个*笔多边形*[HOBB89]。一个笔多边形由下面两个性质来刻画：相对的边互相平行，并且如果多边形被平移使得一条边的顶点位于原点，那么包含其对边的直线一定经过一个整数坐标点。例如，一个顶点为 $\pm(1, 1/2)$，$\pm(1/2, 1)$，$\pm(-1/2, 1)$和$\pm(-1, 1/2)$的八边形是一个笔多边形。使用计算Posch-Fellner算法(线宽为2)和笔多边形方法计算经过原点斜率为0.935的曲线上的点，对后者采用八边形笔。一般地，采用笔多边形画较厚的线会得到更为均匀的外观。 1010

第20章　高级建模技术

前面的各章主要集中于几何模型，包括对它们进行几何变换和绘制。如果世界完全是由简单的几何模型构成，那么以前讨论的几何模型已经足够了。但是许多自然界中的现象是不能用几何模型有效地表示出来的，至少其中很大的一部分是不能的。例如，雾是由水滴构成的，然而，若是使用一个包含每个水滴的模型，则是完全不可能的。而且，这种水滴模型也不能精确表现我们所感知的雾：我们见到的雾是在我们面前模糊的空气，而不是数以万计的水滴。我们视觉所感知的雾是基于雾怎样改变到达我们眼睛的光线，而不是基于单个水滴的位置和形状。因此，为了建立一个能有效地表现视觉所感知的雾，就需要有一个不同的模型。同样，树叶的形状可以用多边形来模拟，树干可以用样条管（spline tube）模拟，但是要把每一个树的枝叉、树梢和树叶准确地放到其对应的位置不仅繁琐，而且耗时巨大。

不仅仅是自然现象无法用几何模型表示，要清晰地描绘出布鲁克林大桥也同样具有挑战性。桥的很多细节是由铆钉、螺钉和螺母构成的，而它们又不是放在每一个支柱或者电缆的同一位置上，因此，是不能使用基本实体举例法的；实际上，它们放置的方式是由支柱或者电缆的位置所决定的（例如，两个电缆末端的粘接需要一个联结装置）。

这一章所介绍的高级建模技术试图超越几何建模，仅通过简单的模型来模拟复杂现象。一般地，这意味着用一个简单的参数模型来表示大量物体的集合，其模型的参数是易于调节的、直观的。因此，模型就可以通过参数列表生成。这种技术被称为*数据库扩展*[SMIT84]，这个术语精确地描述了我们模拟在高层次细节上相当一致的精致实体（例如，雾、桥、火和植物）的愿望。 1011

一些建模技术处在显式建模和数据库扩展之间的某个位置，例如[FORS88]中的层次样条、[BLIN82b]中的滴状物和[WYVI86]中的软物体。*层次样条*是在它们控制网格上有不同密度的曲面片。在建模物体特征少的区域，使用较粗的网格，而在那些具有很多细节的区域，用精细的网格分割在较粗网格中的一个矩形。因此，用适度的信息（控制点和子分层次）就描述了整个面的形状。*滴状物体*和*软物体*也是通过定位几个物体来控制的，然后再将两者“混合”在一起形成一个复杂物体。这几个所需要的物体（软质材料的球体或者滴状物体）必须放在那儿，但它们的相交部分被自动平滑，这使得用户不必费心去明确定义每个物体相交的细节。

这里介绍的一些技术应用很广泛，过程模型、分形模型、基于语法的模型和粒子系统已经被用来模拟各种各样的事物。这些技术的一部分是专用的，例如，海波浪模型。尽管如此，这里介绍的很多模型都给出了很漂亮的图片而不必对基本的科学知识非常了解。Gardner[GARD84]给出的基于纹理二次曲面的云模型仅是云的表现形式，而该模型不是基于大气科学的；以滴状物体建模的水不是根据水面张力和水分子动力学的物理性质。认识到采用这些方法建模和基于科学本质建模的区别是非常重要的。

对于每一种建模技术，必须要有新的绘制方法。在讨论一种新的建模技术的同时，我们也介绍一些为这种建模技术开发的新的绘制技术。例如，体绘制技术，这种在三维空间每一点都赋值的建模技术并不是新的。标量场在物理和数学上已经使用几百年了，但是只是最近才试图在二维图像上绘制这些场。

最后，本章介绍的很多方法是为动态模型开发的，而不是静态模型，不仅描述形状而且描

述生长和变化。通过这些模型产生的每一幅图像都是非常有趣的，但是当几幅静态的图像以动画的顺序被组织起来时，这些建模技术的潜力才被充分发挥出来。因此，这一章中所涵盖的主题包括物体建模和动画。

20.1 前述技术的扩展

在纵览模拟自然和人工物体的新技术之前，我们先讨论前面建模技术两种扩展：层次样条和基于噪声的模式映射。它们中的任一种都不能处理我们以前所不能建模的任一形状和特征，但是，每一种又都使建模大大简化。层次样条使得没必要在细节很少的区域中放置许多控制点（如果使用均匀的精细控制网格，这是必须的），而基于噪声的模式恰恰是要映射到通常物体上
1012 的令人感兴趣的模式。

20.1.1 采用样条的高级建模技术

根据第11章定义的张量积（tensor-product）样条曲面片，要获得更高层次的细节就必须给予更多的控制点。Oslo算法和它的派生算法[COHE80；BART87]可以应用到这些样条的控制网格上，以产生新的具有更多控制点但仍然定义同一曲面的控制网格。图20-1显示这种细化网格方法应用于样条曲线的情况。图20-1中的圆形黑点控制加粗线段的形状；如果移动其中某一点，粗线段的形状就会改变。但是当它的形状改变时需要多少个点来重画这个弧呢？对于该线段的外侧部分，我们可以使用（相对少些）空白顶点；对于该线段内部，我们使用圆黑点。这种细节定位方法是Forsey和Bartels[FORS88]提出的层次B样条方法的基本概念。

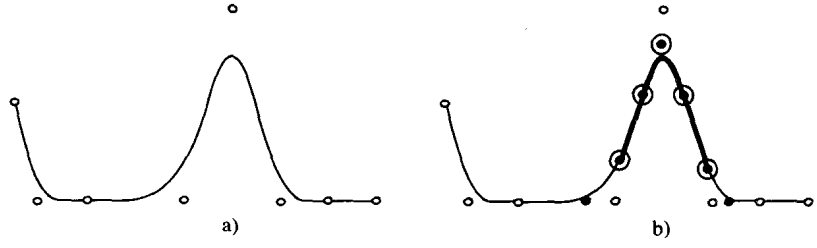


图20-1 在a)中的样条曲线是由显示的控制顶点所生成的。这些控制顶点可以被细化，如图b)所示。b)中的黑点是新的控制顶点；被圈住的黑点是对那段粗线有作用的点

这样将出现两个问题：维护层次样条模型的数据结构和在不影响小尺度样条的前提下对大尺度样条进行改变。这两个问题可以被同时解决。我们希望改变大尺度样条的同时，小尺度的样条也发生改变。在基于大尺度样条的坐标系中，我们通过对小尺度样条的（可调整）控制点的位置描述来进行这种改变。这同时也描述了样条的数据结构，即一个树形结构，在这个树形结构中每一个样条的控制顶点都是由基于其父节点的坐标系确定。每个控制顶点的初始位置定义了坐标系统的原点，大尺度样条法线方向和其坐标曲线切向方向的位移决定了坐标系的基本方向。因此，当大尺度样条发生改变时，原点和位移坐标系的基本向量也发生变化。

图20-2显示这个过程是怎样作用于样条曲线的。彩图IV-4a是这种技术所生成的令人印象深刻的结果。最终的物体是由不同的样条面片拼合生成的，因此，常规的能够用于处理样条的绘制技术（包括多边形绘制技术和光线跟踪技术）可以用来绘制这些物体。大约有500个控制节
1013 点用来定义龙头，其中的每一个节点都包含参数位置、重叠的层次和偏移等参数。通过定义控制顶点相对于骨架模型特定线段（而不是相对于父节点曲面）的位移，Forsey和Bartels将这种技术拓展到皮肤的自动生成，如彩图IV-4b所示。

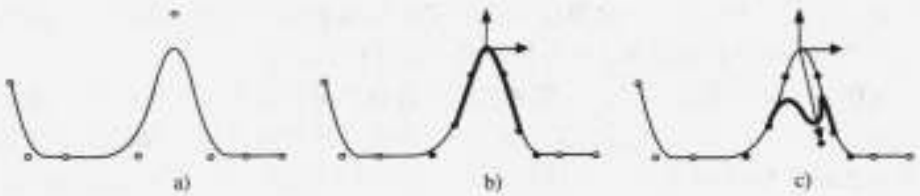


图20-2 a) 一个样条曲线和它的控制点；b) 在中间部分被细分的同一样条。在保持连续性的前提下，只有中间的控制点可以移动。中间控制点的位移坐标系如图所示；c) 沿着新坐标系的位移向量移动后的中间控制点

Sederberg和Parry提出了一种不同的改变样条模型的方法[SEDE86]，尽管它是基于三维空间的样条变形，但这种方法可以改变任意的模型。一个从三维空间到三维空间的函数可以被看成是一种对空间每一点赋予新位置的方法。假设我们有这样一个函数，它不移动大部分的点，仅仅变换空间某一部分的区域。（图20-3显示在平面的一个区域中模拟形变。）如果一个实体是通过在原始空间中的点的坐标描述的，那么将函数应用于这个实体上的每一点会生成一个新的实体，这样原实体就被该函数改变了形状，如图20-3所示的阴影部分的那样。

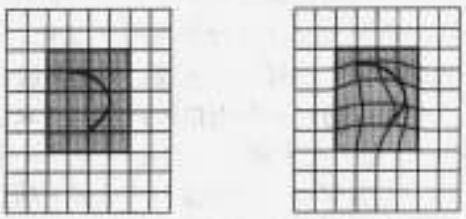


图20-3 由于平面的变形，导致矩形内部区域被扭曲，但是余下部分保持不变。因此在矩形内部绘制的图形也是被变形的

Sederberg 和Parry使用的从三维空间到三维空间的函数是基于Bernstein多项式的，它的形式是[⊖]

$$Q_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \qquad 0 \leqslant t \leqslant 1$$

这些多项式有性质：$\sum_{i=0}^{n} Q_{n,i}(t) = 1$（见习题20.1）并且每个$Q$总是在0和1之间。

如果我们建立一个由原点X和三个基本向量S、T和U的三维空间，通过考虑如下的所有点：

$$P_{ijk} = X + (i/n)S + (j/m)T + (k/p)U, 0 \leqslant i \leqslant n, 0 \leqslant j \leqslant m, 0 \leqslant k \leqslant p$$

我们可以构成一个$(n+1)\times(m+1)\times(p+1)$三维空间的网格，这是以原点$X$和$S$、$T$、$U$所确定平行六面体为基础的网状结构。任何满足$0 \leqslant c_{ijk} \leqslant 1$和$\sum c_{ijk} = 1$点的线性组合$\sum c_{ijk} P_{ijk}$存在于点$P_{ijk}$所定义的凸包中，也就是在平行六面体中。而且，在平行六面体中的点都可以表示为$P = X + sS + tT + uU$，且$0 \leqslant s, t, u \leqslant 1$。假设我们利用如下公式定义一个平行六面体：

$$\begin{aligned} &F(X + sS + tT + uU) \\ &= \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{p} \binom{n}{i}\binom{m}{j}\binom{p}{k} t^i (1-t)^{n-i} s^j (1-s)^{m-j} u^k (1-u)^{p-k} P_{ijk} \\ &= \sum_{i=0}^{n}\sum_{j=0}^{m}\sum_{k=0}^{p} Q_{n,i}(t)\, Q_{m,j}(s)\, Q_{p,k}(u)\, P_{ijk} \end{aligned}$$

由于Bernstein多项式的凸状特性，在边界对应边界的情况下，平行六面体将映射到自身。实际上，如果P_{ijk}如上面所定义，F使平行六面体中的每一个点映射到它自身。如果P_{ijk}被移动，映射也就不再具有惟一性。只有内部的P_{ijk}被移动时，映射才会在平行六面体所包围的实体当

⊖ $\binom{n}{i}$表示$n!/(i!(n-i)!)$。

中保持惟一性。因此，P_{ijk}对一个包围盒中任何实体提供了形状控制。彩图IV-5给出了一个调整自由形状的例子；彩图IV-6显示了一个用任意变形建模的锤子。

利用三元Bernstein多项式变换，计算多边形物体顶点是很简单的（我们仅仅将每个顶点表示为线形组合$X + sS + tT + uU$并在公式F中替换），因此这种方法适合于所有的多边形绘制。尽管可以使用由Barr[BARR84]提出的描述其他类变形的专门方法，但怎样光线跟踪这些变形的物体不是很清楚。

20.1.2 基于噪声的纹理映射

Peachey[PEAC85]和Perlin[PERL85]通过使用立体纹理扩展了传统的纹理映射。在传统的凹凸映射和模式映射中，纹理是从二维图像中抽取出来的，然后将它们映射到需要绘制的表面（见第14和16章）。对于面上每一个不同的点，在二维纹理上都有一个点与其对应，再将这个点周围的纹理值通过某种滤波器平均，结果纹理值就被赋于面上相应的点。

1014
~
1015

这种机制对立体纹理来说稍有变化，一个纹理值被赋于3D纹理空间中的一点。对于需要纹理化的物体上的每一点来说，都会与纹理空间中的某一点相对应；其点的纹理值也是相对应的。我们可以通过一个类似的物理例子来说明这种立体纹理映射。如果我们取一块大理石（纹理空间），那么在大理石上的每一点，不管是表面还是内部，都有某种颜色。因此，如果我们在大理石上雕刻出一个球面，在球面上的点也是被着色的。如果我们在大理石的不同部分雕刻球面，当然球面上的颜色也是不同的。

与这种机制相关的工作有两步：生成纹理和从物体到纹理空间映射（例如，物体上的点和纹理空间点的对应）。对于将物体在某个空间建模，并在绘制前将其变换到“世界坐标系”下的系统中，很容易进行到纹理空间的映射。在这种情况下，对纹理空间最自然的选择是建模空间。在绘制的过程中，一个在物体上的三维点通过使用逆模型变换给出在建模空间的对应点，它的坐标提供了立体纹理空间的索引（这种情形适用于在大理石上雕刻）。当绘制完成之后，改变物体的世界坐标系位置并不影响它的图案纹理。使用世界坐标作为立体纹理空间的索引能够提供有趣的特殊效果。如果大理石球面在动画过程中平移，纹理在其上滑过，就好像从新的大理石连续雕刻出来一样。在其他系统中，必须选择某一坐标系统（或者从世界空间到纹理空间的映射）使得物体上的每一点都有一个纹理值。

生成纹理是一个不同的事情。在讨论这个问题之前，让我们讨论一下纹理映射函数。当一个纹理图用于到反射表面的环境映射时，是没有潜在的立体纹理的。对于把标签放在盒子上使用纹理映射，或者凹凸映射被用来模拟像规则分布的天花板瓷砖之类的结构细节或者产生像擦得很亮的金属表面有向反射的表面特征等，情况都是一样的。但是当我们模拟像混凝土、树木或者大理石的纹理时，基本上物体的内部结构决定了物体的外观。在这些情况下，立体纹理是非常有用的。

作为一种中间情况也能够被立体纹理处理得非常好的是表面特征。例如装饰用的灰泥的纹理，在统计上它们是独立于表面位置的。对于这种情况，普通模式映射倾向于产生方向感，这是因为模式映射定义的坐标系和从映射空间到物体的变换的原因，这种变换往往使得图案在某个地方产生压缩或者伸长（例如，使用标准坐标映射到球面，在极点附近的图案是被压缩了的）。立体纹理是通过关联值来解决这个问题，这种关联值有效地独立于表面形状（见彩图IV-7c）。

生成一个立体纹理需要对某个体积中的每一点对应一个或者几个数。我们可以通过在三维网格（有时被称为三维图像）上每一点产生这些数，然后插值来给出中间值，或者仅仅通过在三维空间某个区域定义的一个或几个实值函数来确定它们。

1016

Perlin使用的大多数函数是基于噪声的函数。他定义的函数$Noise(x, y, z)$具有特殊性质：刚

体运动统计不变性和频域带宽有限性。第一个性质意味着任何统计特性，比如在某一区域上的平均值或者方差，与在其他位置和方向上的适当区域的测量值基本上是一样的。第二个性质说明对信号的傅里叶变换在某一窄频带范围之外是零（见14.10节）。实际上，这意味着函数没有突变，但也没有变化过缓的地方。一种表示有限带宽的方法是对于任意单位向量(a, b, c)和任意点(x_0, y_0, z_0)，当$f(t) = \sin(t)$或者$\cos(t)$，并且m不属于某个很小范围时，积分

$$\int_0^\infty \text{Noise}(x_0 + ta, y_0 + tb, z_0 + tc) f(mt)\, dt$$

等于零。本质上，这说明沿着(a,b,c)方向的参数线，噪声不具有以m为周期的特征。

这样的噪声函数可以采用多种方法生成，包括直接的傅里叶合成，但是Perlin有一个快速、易于实现的方法。对于每一个在整数网格上的点（例如，对某一点(x_0, y_0, z_0)，它的x_0, y_0和z_0都是整数）我们计算并存储四个伪随机[⊖]实数(a,b,c,d)。计算$d' = d - (ax_0 + by_0 + cz_0)$。注意到，如果我们将点$(x_0, y_0, z_0)$代入到公式$ax + by + cz + d'$，我们可以得到$d$值。现在，我们通过两条规则定义在任意点$(x, y, z)$处的噪声函数：如果$(x, y, z)$是整数网格上的点，那么$Noise(x, y, z) =$在该网格点的$d$值$= ax_0 + by_0 + cz_0 + d'$。对于任一不在整数网格上的点，$a$, b, c和d'的值通过周围的网格点进行插值（Perlin推荐三次插值——首先是x，然后是y，最后是z）以给出在点(x, y, z)处a, b, c和d'的值。这样，就可以计算$Noise\ (x, y, z)$的值：$Noise\ (x, y, z) = ax + by + cz + d'$。

由于系数a, b, c和d'是在整数网格上通过三次插值得到的，显然它们的值是连续的（实际上，它们都是具有良好的可微性的函数）。因此，$Noise\ (x, y, z)$的值也具有良好的特性且没有高频部分（也就是函数突变）。

通过改变颜色、法向量等，噪声函数可用来生成纹理[PERL85]。例如，通过将一个点的颜色设置为$(r, g, b) = Noise(x, y, z)*(1.0, 1.0, 1.0)$（假设，$Noise()$的值在0和1之间），而将一个随机的灰度值赋给这个点。一个点可以通过$(r, g, b) = (NoiseA\ (x, y, z), NoiseB\ (x, y, z), NoiseC\ (x, y, z))$被赋于一个随机颜色值，这里$NoiseA()$, $NoiseB()$和$NoiseC()$是$Noise()$函数的不同实例。一种对随机颜色赋值的可选方法是使用噪声函数的梯度：

$$Dnoise(x, y, z) = (dNoise/dx, dNoise/dy, dNoise/dz)$$

它在每一点产生一个三值的向量，这些值可以映射为颜色值。 1017

如果一个物体有足够大的区域，对它的整个的包围盒生成纹理可能是不实际的。相反，如同在第16章中那样，我们在一个有限的立方体（大约$256 \times 256 \times 256$）上生成纹理，并且使用点的坐标的低位对这个数组做索引（使用模运算使它们限制在0至255之间）。我们可以使用有限纹理数组生成另外一种噪声函数，即定义$Noise2(x, y, z) = Noise\ (2x, 2y, 2z)$。$Noise2$具有这样的性质，它生成的纹理大小是$Noise()$产生的大小的一半。通过产生这种$Noise()$函数的乘积，我们可以创建很多迷人的纹理，见习题20.2和习题20.3。Perlin已经拓展了立体纹理以允许做几何修改[PERL89]。一些结果的例子见彩图IV-8和彩图IV-9。

Peachey[PEAC85] 生成立体纹理的方法与此稍有不同。最有趣的一个例子就是他称为投影纹理的东西，虽然也可以叫“突出纹理”。在这种纹理中，空间上沿着特定平行线的纹理函数的值是常数。例如，这样一个纹理，它在每一个平行于z轴的平行线可能都是常数，然而在任何平行于(x, y)平面的横截面上，它看起来像通常的二维的纹理。其效果就像非透视投影的幻灯：当人在屏幕之前走动时，图像映射到人而不是屏幕。当把几个这样的纹理结合起来，就更有趣

⊖ 伪随机数生成在许多系统中是由$Random()$提供的，参见[KNUT69]。

了。如果沿着不同的直线，纹理是常数，结果就更能有效地模拟完全随机的纹理。彩图IV-10的纹理就是基于投射纹理的。

20.2 过程模型

过程模型描述能够与周围环境相互作用来改变自身的对象。因此，一个以所要求的加细分割精度产生球面的多边形表示的球面模型是过程的；实际模型是由精确度参数所确定的。一个需要从附近实体得到信息来决定它的坐标系的原点的模型，也是过程的。一个由它们的顶点确定的多边形集合却不是过程模型。

过程模型很长时间就一直在使用。它们最好的特征之一是节省空间：说“由120个多边形组成的球面”比显式地列出120个多边形容易得多。Magnenat-Thalman和Thalman[MAGN85]描述了一个桥的过程模型，在这个模型中，桥是由路、上层结构、桥墩和栏杆组成的，桥是通过给出这些组成部分沿着决定桥位置的方向的描述来定义的。每一个部分（路、桥墩等）都是由几个参数（路的长度、路与路间连接点的数目、桥墩的高度等）和过程指定的，然后从这些来产生模型。这和第12章的图元引用是类似的，但是不同之处在于物体的几何或者拓扑特性可能被参数所影响。另外，产生的模型也不必由实体的集合组成；例如，在夜间，为了表现桥，可能只需要一个点光源集合就可以了。在任何情况下，指定几个参数，就导致一个非常大的模型
1018 的产生。在桥的情况下，所生成的惟一的物体是各种不同种类的桥。可是在随后的过程模型中，例如粒子系统，一个单一种类的过程就可以支持完全不同种类的物体。

过程模型的一个非常重要的方面是它与环境相互作用的能力。Amburn、Grant和Whitted介绍了标准过程模型的两种扩展：一种是通信的方法，通过这个方法独立过程的行为可以互相影响；另一种是对子分概念的推广，包括表示法变化[AMBU86]。

物体间的通信可以控制由过程定义的物体的形状。作为例子，Amburn、Grant和Whitted用一条通过森林地带的路。该地带是由三角形随机分割生成的（见20.3节），森林中的树是使用基于文法模型生成的（见20.4节），路是通过一条沿着样条的线的突起生成的。在最高层，路必须沿着地形的几何形状。可是在较详细的层次上，地形加密，使得路面平滑。这些物体中的每一个都必须控制另一个。树的根基必须放在地形上，但是不能距路面太近。为了进行这种物体间的控制，每一个分割过程都会分几步进行，然后，检查它与其他物体的分割之间的关系。

这种物体之间的检查将是非常费时的；路需要用几百个矩形模拟，地形用几千个三角形模拟。检查它们之间的相交部分并且在每两个之间建立通信是非常麻烦的。因而，在构造路的过程中，建立路本身和路的每一对控制点以及路的每一段的包围盒。在整个三角形分割的过程中，维护类似的包围盒。只要每个子三角片的包围盒不再和路的包围盒相交，就可以阻断它们之间的通信。因此，在分割的最细层次，很少发生重叠。

这种分割也受表示法变化的影响。在分割过程中的某一点，当前的模型表示对于建模过程可能不再合适，当前执行的建模过程（或者在模型中的其他过程）可能要求某个过程建模中的物体改变它的表示。因此，一个最初用Bezier样条曲面片表示、经过递归分割的形状，可能在过程建模的某一点处就需要转而用随机分割，以便在某个特定的整个形状上形成一个皱褶。Amburn、Grant和 Whitted用一个临时文件记录这些表示法的变化：这些文件要么和单独的物体有关，要么和一类物体有关；比如，在临时文件中可以记录“第三层子分割中，从Bezier变为随机。在第五层，变成粒子系统表示。”当发生过程修改时，人类操作者也允许和物体进行交互。将来，我们希望这种交互不再必要，要创建的模型本身能够自己确定最适合的表示法。

在这一章中，余下的大部分模型在某种方式上是过程的。它们中的许多是通过较小的物体的反复分割和细化生成的。分割终止的层次是由建模者、模型或者绘制过程(取决于实现方式)决定的(例如，绘制过程可能要求不许产生子像素级的缺陷)。这些模型的作用体现在如何使建模者的努力事半功倍：在描述中很小的变化会导致形状发生剧烈的变化。(当然，在某些情况下这可能也是缺点，如果建模者不能控制结果中的微小变化。)

20.3 分形模型

分形近来引起很多人的注意[VOSS87；MAND82；PEIT86]。分形图像是非常引人入胜的，而且已经开发许多产生分形图形的方法。术语*分形*在计算机图形学领域中被推广到包括在Mandelbrot原始定义之外的对象。现在，它的意思是任何对象只要它有准确或者统计自相似的测度，我们就可以称其为分形，当然精确的数学定义是要求在任何分辨率下都有统计的自相似性。因此，只有通过无穷递归生成的分形才是真正的分形物体。另一方面，通过有限步骤生成的物体，到某一阶段，在视觉上察觉不出细节上的差异，这是理想状态的适当近似。我们所说的“自相似”可以利用一个例子即von Koch雪花很好地说明。开始时，一个直线段上有个凸包，如图20-4所示，我们对每一直线段都用与原始线段相似的线段替代。这个过程重复地进行：在图20-4b中每一段都用和完整形状相同的图替代。(用a)的图形替代和用b)的图形替代是一样的；如果使用a)替代，经过2^n步的结果和在当前图形的每一段用每一阶段的整个图形替代的第n步的结果是相同的。)如果这个过程进行无限多次，结果就是所说的“自相似”：整个物体和它自身的一部分相似(也就是，可以被旋转、平移和缩放等)。

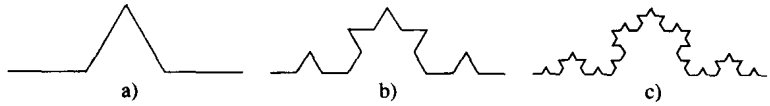


图20-4 von Koch雪花的构造：在a)中的每一段用整个图缩小3倍来替换。同样的过程应用于b)生成c)

不是精确自相似的物体仍然可以看成分形；也就是它可能实质上自相似。统计自相似的精确定义在这里是不必要的——我们只需要注意到缩放变换后仍然看起来“像”的物体仍然称为分形。

和自相似有关的概念是*分形维数*的概念。为了定义分形的维数，我们先看一下已知维数对象的一些特性。线段是一维；如果我们将线段分成N等份，每一部分和原始线段缩小一个比例因子$N = N^{1/1}$相似。正方形是二维的：如果我们把它分成N等份，每一部分和以因子$\sqrt{N} = N^{1/2}$缩小时的原始正方形相同。(例如，一个正方形被精确分为九个子正方形，每一个子正方形和原始正方形的1/3相似。)对于von Koch雪花又是什么情况呢？当它被分为四部分时(这些部分和在图20-4a中原始的四个线段有关)，每一部分和原始图像的1/3相似。我们可以说它的维数是d，且$4^{1/d} = 3$。d的值是log(4)/log(3)=1.26…。这就是分形维数的定义。

在这里值得提到的最著名的两个分形对象是：Julia-Fatou集和Mandelbrot集。这些对象产生于规则$x \to x^2 + c$的研究(也包括许多其他规则，但这是最简单并且是最著名的)。这里x是复数[⊖]，$x = a + bi$。如果一个复数的模小于1，反复自乘将使它趋向于零。如果模大于1，反复自

⊖ 如果你不熟悉复数，把i当成一个特殊的符号并且知道复数的加法和乘法的定义就足够了。如果$z = c + di$是第二个复数，那么$x + z$定义为$(a + c) + (b + d)i$，xz定义为$(ac - bd) + (ad + bc)i$。我们可以把复数表示成平面上的点，将点(a, b)对应复数$(a + bi)$。复数$a + bi$的模是实数$(a^2 + b^2)^{1/2}$，它给出复数大小的度量。

乘将使它越来越大。如果模等于1，经过反复自乘模仍为1。因此，一些复数反复自乘趋向于零，一些趋向于无穷大，还有一些既不趋于零也不趋于无穷大——这最后的一组数形成了趋于零的数和趋于无穷的数的边界。

假设我们对每一个复数x和不等于零的c反复使用映射$x \to x^2 + c$，例如$c = -0.12375 + 0.056805i$；一些复数将会吸引到无穷大，一些将会吸引到有限的数，有些数将既不会吸引到无穷大，也不会吸引到有限的数。画出这些点的集合，我们就能得到如图20-5a所示的Julia-Fatou集。

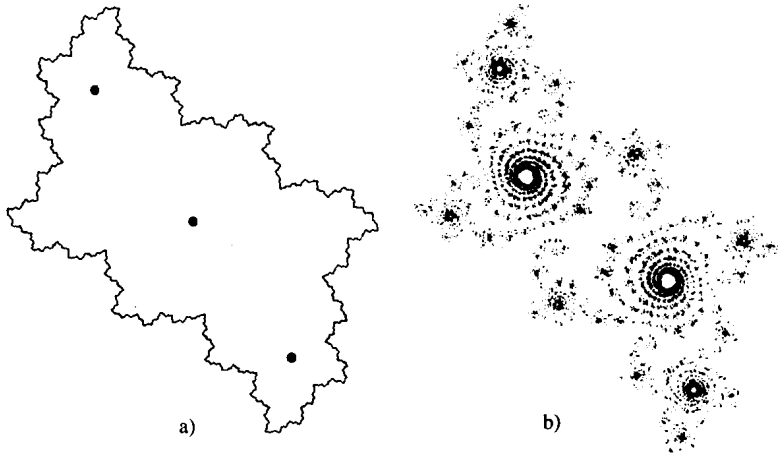


图20-5 a)当$c = -0.12375+0.056805i$ 时的Julia-Fatou集；b) 当$c = -0.012 + 0.74i$ 时的Julia-Fatou集

1021 应当注意图20-5b的区域不像图20-5a的那样保持了很好的连续性。在图20-5b中，一些点落向三个显示的黑点，一些趋向无穷大，还有一些与前两者相反。这最后的一类点是那些在图20-5b中被作为轮廓线画出的点。Julia-Fatou集的形状显然是由c的值决定的。如果我们计算所有可能c值的Julia-Fatou集，并且在Julia-Fatou集是连通的（也就是由一块组成，不是碎成不相连接的孤岛）时将点c着为黑色在该集不连通时c着为白色，我们就得到如图20-6所示的点集，称为Mandelbrot集。注意到Mandelbrot集是自相似的，即在集合的大圆盘的边界上，有几个小的集合，每一个和大的集合在比例上都是极为相似的。

幸运的是有一种非常容易逼近Mandelbrot集的方法：对每一个c值，取复数$0 = 0 + 0i$并且利用公式$x \to x^2 + c$有限次（比如1000次）。如果经过这么多次迭代值已经在定义的模小于100的圆盘之外，我们将c的颜色着为白色；否则为黑色。随着迭代次数和半径的增加，结果图像和Mandelbrot集就越接近。Peitgen和Richter [PEIT86]给出了Mandelbrot集和Julia-Fatou集的图像的生成方法的详细说明。

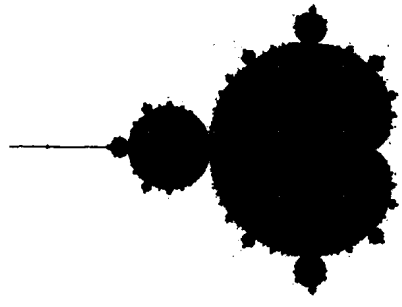


图20-6 Mandelbrot集。复平面上每一点c，如果对于过程$x \to x^2 + c$的Julia集是连通的，就被着为黑色

这些结果对于模拟自然形态是非常具有启发性的，因为很多自然现象都有惊人的自相似性。山有山峰、小山峰、岩石和碎石，它们所有的看起来都很相似；树有大枝、细枝和小枝，它们看起来也很相似；海岸线有海湾、水湾、河口、小河和灌渠，它们看起来也相似。因此，在某个尺

度上建立自相似模型似乎是一种建立自然现象的壮观模型的方式。在此，自相似失效的尺度不是特别重要，因为意在建模而不是数学。因此，当一个物体经过足够多的步骤递归生成，以致该物体所有进一步变化都发生在像素分辨率以下，那么递归过程就没必要继续进行下去。

Fournier、Fussell和Carpenter[FOUR82]开发了一种基于递归分割生成一类分形山的方法。这在一维情况下最容易解释。假设我们以一个在x轴上的线段开始，如图20-7a所示。如果我们 1022
现在把线段平分为二等份并且在y方向把中点移动一个距离，我们得到图20-7b。继续分割每一个线段，我们从(x_i, y_i)到(x_{i+1}, y_{i+1})的线段中点计算一个新值：$x_{new} = (x_i + x_{i+1})/2$，$y_{new} = (y_i + y_{i+1})/2 + P(x_{i+1} - x_i)R(x_{new})$，这里$P()$是一个根据线段长度确定扰动幅度的函数，$R()$是一个根据$x_{new}$选择的在0至1之间的随机数[⊖]（见图20-7c）。如果$P(s) = s$，那么第一个点的位移不能超过1，接着的两个点的位移不能超过1/2（它们最大高度是1/2），依此类推。因此，所有的结果点都在单位正方形中。对于$P(s) = s^a$，结果的形状依赖于a的值；较小的a值将产生大的扰动，反之亦然。当然，其他函数也可以使用，例如$P(s) = 2^{-s}$。

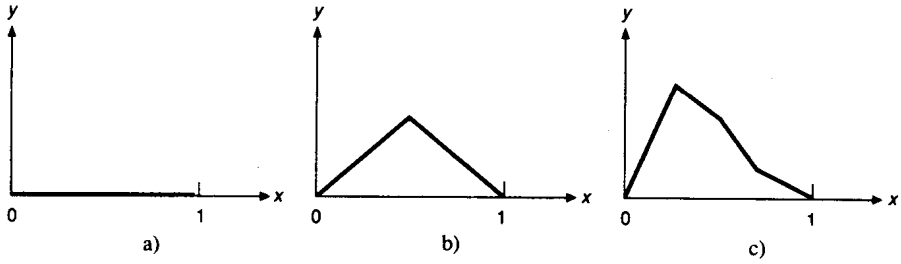


图20-7 a)在x轴上的直线段；b)直线的中点在y方向已经以一个随机量平移；c)进一步迭代的结果

Fournier、Fussell和Carpenter以如下方式使用这个过程来改变2D形状。他们从一个三角形开始，标记每条边的中点，并且连接三个中点，如图20-8a所示。每一个中点的y的坐标用我们以前描述的方式修改，导致如图20-8b所示的四个三角形。当迭代下去时，这个过程就会产生很真实的山，如彩图IV-11所示（尽管从顶上看，会看到一个非常规则的多边形结构）。

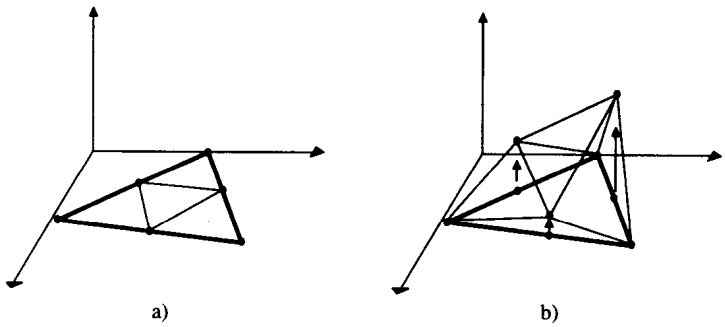


图20-8 a)一个三角形分割成四个较小的三角形，原始三角形的中点在y方向被扰动之后生成b)中的形状

注意到我们可以从具有一定形状的三角形的排列开始，然后应用这个过程生成进一步的细节。这种技术在某些建模应用时特别重要，比如在一个场景的物体布局中在较低层次看似是随

⊖ $R()$实际上是一个随机变量，一个取实数值并产生0和1之间随机分布数的函数。如果由伪随机数发生器产生，就有一个优点，分形可以重复：我们可以通过提供同样的种子点给伪随机数发生器来在此产生它们。

机的，而在较高的层次上则是有规律的：在一个装饰花园中的植物可以采用随机机制产生，但是它们的排列必须严格。另一方面，初始三角形排列的高层结构在迭代分割过程不变的话，对于某些应用是不合适的（特别是，如此生成的分形没有统计上的自相似，如在基于布朗运动的分形中显示的那样[MAND82]）。再有，由于任一个顶点的位置只被调整一次且以后就不变了，沿着原始三角形之间边沿的表面容易产生褶皱，这看起来很不自然。

Voss[VOSS85]描述了这个算法的修改版本，在模型的第$n+1$步生成的过程中对第n步的所有顶点和边的中点附加一个随机的扰动。这种方法消除了原始分割方法的大部分缺陷，但是缺少原算法的控制。Voss也讨论了生成在缩放变换下具有更大的统计不变性的模型的方法，以及除了与分形的原始定义协调外，还具有其他数学性质的方法[VOSS85]。特别是Weierstrass-Mandelbrot随机分形函数为生成单变量的分形函数给出一个计算上可跟踪的机制，并且无疑可以扩展到两个或更多变量。

Mandelbrot已经对中点位移算法进行了另一个改进[PEIT88]。他首先注意到在原来的中点位移算法中位移是对称的，因此当一个这种类型的分形山被倒置，它有同样的统计特性。真实的山从倒置山谷来看是非常不同的，Mandelbrot通过从非对称分布(例如二项式分布)选择位移来模拟这种不对称。他还通过选择不同子分方法减轻中点模型的一些褶皱。他是从一个六边形网格开始，而不是三角形网格。注意到高度值只需要和网格上顶点相关，他在分割的过程中改变网格的拓扑结构，以便六边形最初的边在分割之后不再是边。因此，他用三个小的六边形替代原来的六边形，如图20-9所示。中间的顶点的高度像三角形算法那样算出，即原始六边形与相邻顶点的平均值加上一个位移。其他的六个新顶点的高度值是六边形顶点的平均值计算出来的。Mandelbrot说不同的加权方法得到实质上不同的结果。这种分割的主要特征是原始六边形的可能会生成褶皱的边被分割成多个边，这样就使得生成的褶皱很不明显。由此生成的分形是相当吸引人的。

1023
↓
1024

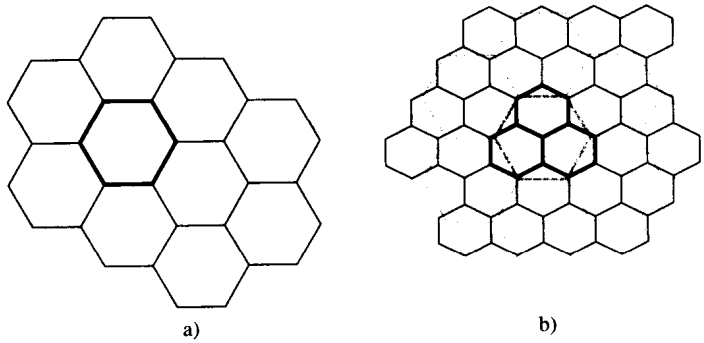


图20-9 a)最初六边形网格，其中的一个六边形用粗线画出。b)被分割的六边形网格，被标出的初始六边形的后代用粗线画出

要进一步了解分形算法，请看[VOSS85；PEIT86]。

用其他迭代算法也能生成大量有趣的图像。下面几节描述的基于文法的模型和粒子系统就有如此功能。这些模型在较深层递归的变化表明对自然物体缺少自相似的模型。树在特定层次上可能是自相似的（树枝和嫩枝看起来很像），但是树叶就不真的像树。

绘制分形会是非常困难的。如果采用z缓存方式绘制一个分形，那么由于涉及大量多边形，显示起来将很费时。在扫描线绘制算法中，分类所有的多边形代价高昂，所以仅考虑与扫描线相交的多边形。但是由于必须判断每一个光线和数以百万计的多边形的相交情况，所以用光线跟踪来实现也是极困难的。Kajiya[KAJI83]对[FOUR82]中提出的分形给出了一种光线跟踪算法，

Bouville[BOUV85]采用了更好的物体包围体而改进了该算法。

Kajiya指出，如果我们以三角形开始并且在三角形内垂直地移动点，如[FOUR82]中所描述的，那么结果物体将被一个无限的三棱柱包围，它的截面是原始的三角形。如果三角形的点移动得足够小，那么它们的和是有限的，并且基于三角形的分形的形状被包围在被截断的三棱柱中（像酪饼的一片；见图20-10）。因此，我们可以光线跟踪一个分形山，首先检测光线是否和每一个初始三角形的“酪饼切片”相交；如果不相交，不必进行进一步的检测它的子分割部分。对要进一步分割的三角形创建附加“酪饼切片”，我们可以进一步减少相交测试，尽管为分形中的每一个单个小三角形建一个酪饼切片需要大量的空间。 1025

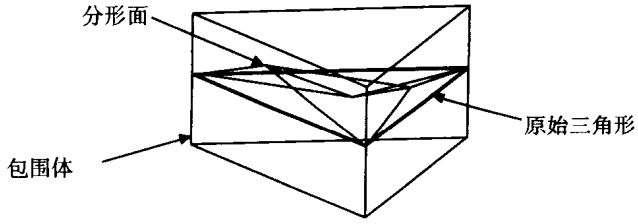


图20-10 一个包围三角形分形扰动的“酪饼切片”

这个方法有两个缺点：检测一条光线是否和“酪饼切片”相交需要和几个平面计算交点(也就是解几个代数方程)，并且酪饼切片不是紧包围体——许多光线和酪饼切片相交但和分形不相交。Bouville注意到：当一个三角形被子分并且内部的点被移动时，原始的点保持不动[BOUV85]。因此他提出定义一个包围这个子分三角形的椭球以便原来的三个顶点在椭球的一个大圆上，所移动的内部顶点都在这个椭球内。实际上，只要移动的顶点在这个椭球内的可能性很大，那么结果是非常有趣的（可能性的大小需要艺术的而不是科学的判断）。如果椭球足够大能够包含所有可能的移动顶点，它也许是一个很差的包围区域，光线也许碰到椭球但是没有和包围在椭球内得分形物体相交。注意到检测光线和椭球的相交是很容易的：它等价于解一个二次方程。这使 Bouville方法要比“酪饼切片”法快得多。而且，椭球比酪饼切片包括更少的无关区域，因此将有更少的递归层次。

还有一种分形建模技术值得提到，那就是第17章提到的迭代函数系统（IFS）。在那里描述的迭代函数系统不同于这一章所有其他形式的建模技术，就在于IFS为图像建立模型而不是图像中的物体。也就是说，一个收缩仿射图的集合（赋给相应的概率）和一个着色算法(如17.7.2节所述）就提供了一个像素图的紧凑描述。例如，在彩图IV-1所显示的场景中，仅仅改变一个仿射图就会使图像发生很大的扭曲，在每一个树上剪掉一个大枝（并且从每一个大枝剪掉一个细枝，从每一个细枝剪掉一个小枝）。它也会在不想要的地方生出叉。

迭代函数系统可以用来产生非常复杂的图像，由于这类图像对模式映射经常是理想的，我们期待看到迭代函数系统成为造型工具箱的一个标准部件。

对迭代函数系统进行仔细的研究表明：这种技术实际上不依赖于维数，因此也可以创建三维物体的迭代函数模型。在某种意义下，下面讨论的基于文法的模型非常相似：一个模型的新的部分的产生是通过把旧部分变换成初始物体的部分或全部的较小尺寸的副本实现的。 1026

20.4 基于文法的模型

Smith[SMIT84]介绍了一个描述某些植物结构的建模方法，这种方法最初是由Lindenmayer[LIND68]开发的，它通过使用并行图文法语言（L文法）来描述，Smith称它为graftals。这种语

言是通过一个产生式集合组成的语法来描述的，所有的产生式都至少用一次。Lindenmayer将语言扩展后包括了括号，因此语言的字母包括两个特殊的符号“[”和“]”。一个典型的例子是字母表{A，B，[，]}和以下两个产生式规则组成的文法：

1) A→AA

2) B→A[B]AA[B]

从公理A开始，前几代产生的是A，AA，AAAA，等等；从公理B开始，前几代是

0) B

1) A[B]AA[B]

2) AA[A[B]AA[B]]AAAA[A[B]AA[B]]

等等。如果我们说语言中的一个词代表图结构中一系列段，括号部分代表分支从哪开始，则由此生成的三层图如图20-11所示。

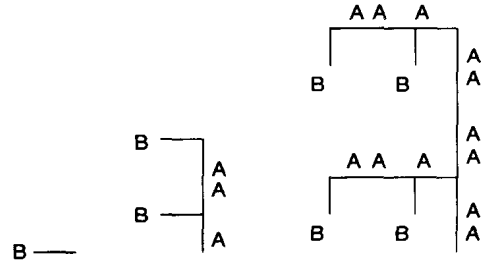


图20-11 语言的头三个词的树表示。所有的分支画在当前主轴的左侧

这个图的集合有一个很好的分支结构，但是更平衡一点的树会更有吸引力。如果我们将上述语言加入符号“(”和“)”，并改变第二个产生式为A[B]AA(B)，那么第二代为：

2) AA[A[B]AA(B)]AAAA(A[B]AA(B))

如果我们假设方括号代表左分支，圆括号代表右分支，那么相关的图如图20-12所示。在这样一种语言中，进行到后面，我们可以

1027 获得代表非常复杂模式的图结构。这些图结构有某种形式的自相似性，因为通过第n代的单词描述的模式(在此情况下，重复地)被包含到第$n+1$代中。

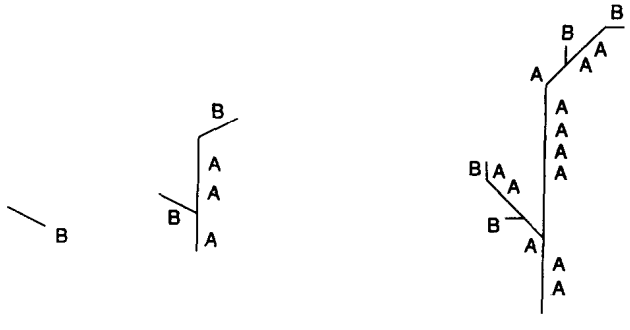


图20-12 两侧平衡的语言的头三个词的树表示。当前进到下一代时，我们已经使树的新的一段变短

从这样的词汇中生成一个物体是从生成词本身的过程中分离出来的。这里，树的各个部分以连续变短的长度来绘制，每一个分支的角度都是45°，分支向左或向右生长。对于不同深度的分支选择不同的角度，对不同的段选择不同宽度的线(或者甚至不同直径的圆柱)给出不同结果；在树的每一个终节点绘制“花”或者“叶”会进一步增强图的效果。由于文法本身没有内在的几何内容，因此，使用基于文法的模型既需要语言的文法解释又需要语言的几何描述。

有好几个研究者对语言的增强和词汇的解释(也就是从词汇产生图)进行了研究[REFF88；PRUS88]。文法已经被丰富到允许我们跟踪在一个词汇中字母的年龄，使得老的和年轻的字母变换时是不同的（这种年龄的记录可由如下形式规则完成：A→B，B→C，C→D，…，Q→QG[Q]，以至于直到植物已经“老化”，感兴趣的过渡才发生）。许多工作都集中于构造文法，

这些文法准确地表达植物发育的生物学。

然而，在某些情况下，一个文法作为植物的描述很笨拙：很多附加特征要加到文法或者文法词汇的解释中。在Reffye的模型[REFF88]中，植物生长的模拟是由一个小参数集来控制的，这些参数由生物术语来描述，并且体现在算法之中。该文法的产生式是按概率规则而不是确定性规则来使用的。

在下述模型中，我们像从前一样以一个简单的茎开始。在这个茎的顶端是蓓蕾，它可以经历几种变化：它可以死亡，可以开花然后死亡，可以休眠一段时间，或者成为一个内部节点——在两个蓓蕾之间的一段。一个内部节点的生成过程分为三步：一个原始的蓓蕾可以生成一个或者多个叶腋蓓蕾（在两个内部节点交叉处的某一侧的蓓蕾），这个过程称为分支；加入内部节点；新的内部节点的端点变为顶点蓓蕾（一个处在一系列内部节点端点处的蓓蕾）。图20-13显示从蓓蕾到内部节点变化的例子。

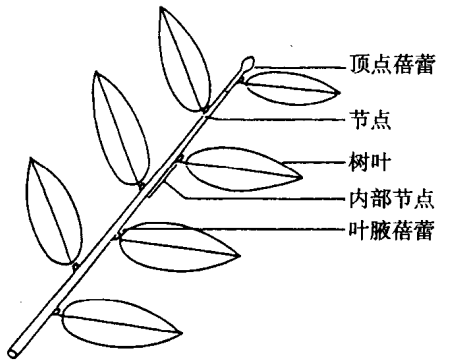


图20-13 在一个植物的一段的顶点的蓓蕾可以成为内部的节点；之后，它产生一个新的蓓蕾（叶腋蓓蕾）、新的一段（内部节点）和一个在顶端的新蓓蕾（顶点蓓蕾）

在结果物体中的每一个蓓蕾都经历相似的变化。我们把树的原始部分称为具有层次1，我们可以采用演绎的办法定义其他内部节点的层次：由层次i的内部节点的顶点蓓蕾生成的内部节点也具有层次i；从层次i内部节点的叶腋蓓蕾生成的内部节点具有层次$(i+1)$。这样，整个树的主干是层次1，大枝为层次2，在大枝上的细枝为层次3，依此类推。图20-14显示了一个更复杂的植物和这个植物不同内部节点的层次。

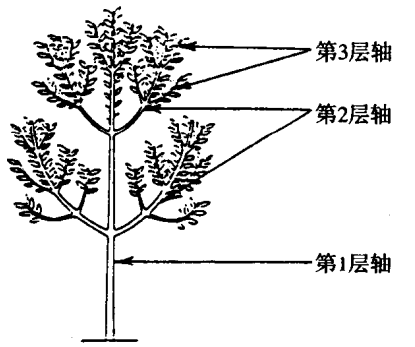


图20-14 一个更复杂的植物(参见图20-13)，在不同的内部节点标有层次标记

到此为止的讨论描述了植物的拓扑，而根本没有描述植物的形状——分叉是指向上、向下或指向一侧等，这些信息还没有被记录。在层次i的内部节点放置叶腋节点将有不同方式（见图20-15a），在从层次i叶腋节点生出层次$i+1$的内部节点(如果有的话)的角度也决定植物的形状（见图20-15b）。在树的生长过程中也有某些异常，层次$i+1$的内部节点集合的行为是不标准的，而是与某个较低层次的节点集合相似（称之为迭代），而这种情况也必须考虑。

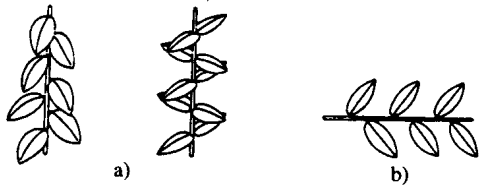


图20-15 a)叶子的两种不同的排列：螺旋的和对分的。b)不同分支角度的效果

最后，将这种描述转换为一个实际树的图像需要一个模型，以描绘不同部分的形状：例如，层次1的内部节点可能是大的锥形圆柱，层次7的内部节点可能是细小的绿线。惟一的要求是在每一个叶腋节点上有叶子（尽管叶子有时会落下）。

为了模拟这种模型中植物的生长，我们需要以下的生物学信息：模型的当前年龄，每一个

内部节点层次的生长速率，在每一个内部节点开始的叶腋蓓蕾数目（作为内部节点层次的函数），以及作为年龄、维数和层次函数的死亡、间歇、分支和迭代的概率。我们也需要一定的几何信息：每个内部节点的形状（作为一个层次和年龄的函数），每一个层次和年龄的分支角度，以及每个轴的取向（层次i的内部节点序列是一个直线还是朝向水平或垂直的曲线）。为了绘制一个植物的图像，我们还需要更多信息：每个要绘制的实体的颜色和纹理——不同层次的内部节点、不同年龄的树叶和不同年龄的花朵。

模拟生长过程的伪代码如图20-16所示。

```
for（每一个时钟周期）{
    for（仍然存活的每个蓓蕾）{
        根据层次、边等确定将要发生的情况；
        if（蓓蕾并没有死）
            if（蓓蕾并没有休眠）{
                以其位置、方向等几何信息
                    创建一个内部节点；
                创建顶点蓓蕾；
                for（旧的蓓蕾位置的每个可能的蓓蕾）
                    if（分枝）
                        创建叶腋蓓蕾；
            }  /* if */
    }  /* for */
}  /* for */
```

图20-16 植物生长算法的伪代码，摘自[REFF88]

我们可以将这个完整的讨论放到这样一种文法模型中去，该模型通过对不同年龄的顶点蓓蕾和叶腋蓓蕾赋予字母表中不同的字母，并将语言的产生式和概率相联系。可是因为产生式的应用和在伪代码中处理是等同的，这样的改变是否有特别价值还不清楚。

改变不同的概率和角度的值能产生各种各样的极具有说服力的树模型，其中的几个如彩图IV-12和彩图IV-13所示。这些参数的正确选择取决于植物学知识或者建模者的艺术眼光；若用了错误的值，我们也能够生成植物，但它和真实植物一点也不像。

这些植物模型是基于文法的建模技术最壮观的例子。在其他应用领域也可以使用这种方法，包括建筑[STIN78]。在任何被建模的对象表现足够的规则性的领域都有机会发展出一种基于文法的模型。

20.5 粒子系统

粒子系统对于某些物体建模来说是一种有趣的方法，随着时间的演变，这些物体的行为不易通过物体的表面进行描述（也就是物体的拓扑会发生改变）[REEV83；REEV85]。一个*粒子系统*是通过粒子的集合来定义的，这个集合随着时间演变，这个演变过程是由应用在其上的概率规则决定的：它们或许产生新粒子，或许根据它们的年龄获得新的属性，或许死亡（从物体上消失）。它们也许根据确定性的或者随机性的运动规律移动。粒子系统已用于建立火、雾、烟、焰火、树和草的建模。

在图形学建模技术中，粒子作为基本的实体已经使用了许多年，特别在以前的视频游戏中，它们表示子弹或者爆炸的战舰。然而这种粒子是确定性的并且每一个粒子需单独放置。大规模粒子集合的效果以前也被用过，主要是模拟在雾中或其他漫射介质中光的反射和透射

[BLIN82a；NISH87；RUSH87]。粒子系统的精髓在于粒子的位置是自动产生的，它们的演化过程是被自动控制的，并且单个粒子直接影响最终的图像。

在Reeves关于粒子系统的第一篇文章中[REEV83]，他描述了粒子系统在火、爆炸和焰火建模中的应用。Reeves和Blau着手用它们来模拟森林中的草和树[REEV85]。在这个意义下，粒子系统看起来特别像前一节描述的随机的基于文法的建模技术。例如，作为粒子系统建模的树，在这个树上，每一分支都是一个粒子，每一个粒子都是沿着主干的长度随机放置的；每一个分支根据某个概率分叉或者伸展。分支的角度或不同的段是根据分布来选择的，这也适合于分支的长度（根据分支在树的位置）。在这个系统中的粒子像在基于文法方法中字母集中的字母，粒子的诞生、死亡和变换的规则相当于文法中的产生式。

1028
~
1031

在[REEV83]中对火的建模采用了不同的方法。这里，粒子有个树结构（粒子有子粒子），但是树结构不被吸收到结果图像中去。在彩图IV-14模拟“起源”的效果时使用了两级粒子系统。第一级是生成一个粒子的集合，这些粒子在半径变化的圆上，这些圆是以行星表面的一个单个点为中心的；粒子是分布在圆上的通过概率分布函数选择的随机位置上。这些粒子的每一个都被作为不同类型新的粒子系统的开始位置（一个爆炸粒子系统）。

在起源效果中，一个爆炸的粒子系统被用来模拟行星表面上的一个区域小的火花喷发（这类系统同样可以模拟焰火和类似的现象）。系统中的粒子是在行星表面的一个圆盘上生成的，且拥有一个初始的运动方向，它的方向是从表面向上的，但是也可能有一些水平分量（见图20-17）。在随后的时间中，粒子的位置是通过将速度向量加到当前位置来计算的；速度向量也可能由加速度向量（这可能包括重力）更新。在圆盘上粒子的放置、产生它们的速率、初始速度和粒子的寿命都是随机选择的。在每一这样的选择中，属性值是通过如下形式的规则来选择的：

$$property = centralValueForProperty + Random() * VarianceOfProperty$$

因此中心值和属性的方差也必须确定。

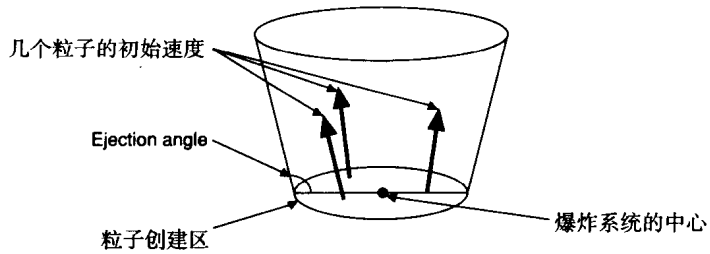


图20-17 对爆炸建模的粒子系统的初始阶段

粒子系统的颜色初始值被设置为红色，带有绿色和一点点蓝色，并且随着时间的流逝颜色退去，而红色持续的时间比绿色或蓝色要长，这样就可以模拟白热物质的冷却。

绘制粒子系统则完全是另外一回事。因为计算每根光线和数以百万计粒子的每一个包围盒的交是非常费时的，所以光线跟踪粒子系统是不可能的。为了绘制“起源”序列中的火，Reeves直接将每一个粒子作为一个小的光点并且计算这个光源对最终图像的贡献。由于粒子是运动的，他实际计算一个短的线段代表在帧的绘制中粒子的路径，然后将短线段绘制（经过反混淆）成为一个最终的像素图[⊖]。每个像素值通过累加每个粒子的值来计算，因此一些受许多粒子影响的像素不能超过红、绿和蓝的最大值（特别是红，由于它是主要的粒子颜色）。实际

⊖ 这构成运动模糊，见第14章和21章的讨论。

上处于其他粒子背后的粒子仍然对图像做出贡献，因此不存在粒子的遮挡。为模拟行星上的熊熊大火，采用了两个技巧。第一，先绘制所有在行星隐藏面的粒子，然后绘制行星，最后绘制在前面的粒子。按照后面的粒子-行星-前面的粒子的顺序组合起来，就可以阻止后面的粒子通过行星表面显示出来（也就是绘制的粒子图像，没有存储任何z信息）。粒子系统只对屏幕图像贡献出光，然而实际的火光也照亮行星的附近部分。Reeves是通过在行星表面附近放置一个高密度锥形光源来模拟的，而不是从粒子计算直接的光照。

在为电影《André and Wally B.》设计的森林场景中[REEV85；LUCA84]，由于粒子系统不再是光源，而是树和草，并且它们成为光的反射器，所以需要和以前不同的绘制技术。为了绘制粒子系统必须开发特别的绘制技术；一些树遮挡另外一些树，这些树的不同部分处在阴影中，有的草在树的阴影中等。绘制的办法包括两个方面的：为阴影开发概率模型和使用改今后的z缓冲技术计算遮挡。沿着从光源到粒子的光线计算树上粒子进入树的深度，从而处理树（树叶或者茎）上的粒子的浓淡（见图20-18）。这个深度被用来计算一个漫射分量减弱的指数：$D = e^{-kd}$，这里D漫射分量，k是一个常数，d是粒子的深度。具有非常小的d值的粒子随机地计算镜面高光；如果d非常小并且光线的方向和分支的方向几乎垂直，那么一个镜面高光可能被加入。最后，环境光(它在树的内部很小，在边沿较大)通过设置$A = \max(e^{-js}, A_{min})$计算，其中$j$是常数，$s$是从粒子（在任何方向）到树的边沿的距离，且$A_{min}$是环境光的下界（即使树的最深部分也有微光照到）。如果树在其他树的阴影中，高光和漫射分量不应该被计入。这通过确定从近处的树到所考虑的树所决定的平面来实现；如图20-19所示，该平面包含附近树的顶端和光源，并且在它的法线方向有最大的y分量。这个平面上的粒子使用三个分量来照亮，然而对于那些在该面下面的粒子，随着到平面的距离增加，它们漫射分量和高光分量（在概率的意义上）就越来越小。

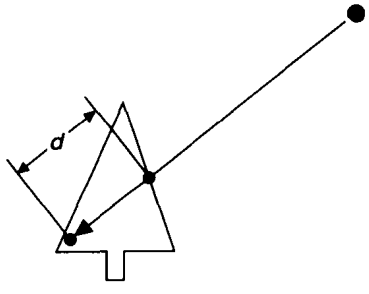


图20-18 沿着光源到粒子的光线，树上的每一点都存在有一个深度。这个深度值决定这个粒子被照到的可能性

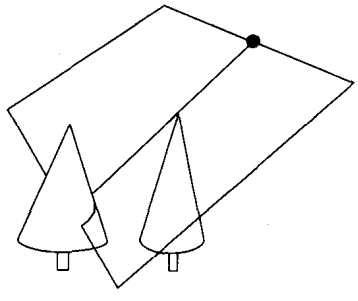


图20-19 由相邻树确定的平面，它决定树的阴影

即使使用这种简化的光照计算，仍然要决定可见面。在场景中的这些树按照从后到前的顺序分类，也按照这种顺序绘制。在绘制树时使用一个桶分类类型的z缓存。每一个树的深度范围被分为许许多多桶，每一个被生成的粒子根据其在树上的深度插入相应的桶。当所有的粒子被生成之后，使用从后到前的顺序绘制。每个粒子画成一个小圆圈或者一个小线段（经过反走样）。当每一个树绘制之后，树的信息被抛弃。这个排序的结果是一棵邻近树的分支可能覆盖稍微远些的树的分支，即使第二个分支在第一个分支之前，因为第一个分支作为一棵完全在第二个分支之后绘制的树的一部分。在有足够复杂的场景中，这种排序错误似乎不是一个问题。

这个在绘制场景上的困难凸显了一般粒子系统的缺陷：建模者获得了相当的能力，但可能要为每一个应用开发特殊用途的绘制技术。

20.6 体绘制

体绘制是以二维图像来显示一个实体区域内部的特征。在一个典型的例子中，实体是一个被加热的机械加工的零件，且实体内部每一点的温度都通过物理或者数学方法计算出来。可视化地显示这些温度是有意义的。严格地说，这不是一个建模问题，因为要显示的物体的形状和特征先前都是可用的。但是将这些数据转换成为像素图中的信息是一种形式的建模，也就是建立从三维到二维变换的模型。在另外一个例子中，通过CT可以计算出在三维网格点上人或动物的组织密度。这个信息的显示应该指示出不同组织的边界（通过密度变化指定）。定义这个边界的表面必须从采样数据中获得以便能够绘制这个实体。 1032 - 1034

与体中每一点相关联的数称为在那点的值，所有这些值的集合称为在这个体上的标量场。在一个体空间具有给定值的所有点的集合被称为层次曲面（如果标量场是足够连续的，这个点的集合确实形成一个曲面）。体绘制是一个显示标量场的过程。重要的是认识到被显示的数据可能不是理想的。如果数据是在一个规则网格上采样，那么它们所代表的标量场所包含的频率可能高于奈奎斯特采样频率（见第14章）。例如，在断层图像中，从肌肉到骨头的过渡是非常突然的，因此包含很高的频率，但是采样频率很可能太低而不能精确地代表这种变化。同样，描述物体内部的数据或许以某种不规则模式聚集，例如通过核心采样获得的地理数据，在这种情况下它不可能均匀地采样。

已经提出了几种不同的体绘制方法。它们可以分为两大类：计算层次曲面的一类和显示沿着光线的密度积分的一类。通过将密度赋予特定的层次曲面，然后采用光线跟踪的办法显示这个结果，可将两类方法结合到一块（这和创建一个不同的体去显示是等同的）。如果可以用动画，第三类绘制方法是可能的：数据的一系列的二维切片被连续计算和显示，使用颜色和高光去显示切片上每一点的标量值。如果可以交互控制层次和切片方向，这种方法可以给出标量场内部结构的精彩景像。

尽管如此，有时用集合方法观察数据要比切片有用，一种方法（虽然不是第一个）是移动立方体（marching-cubes）算法。在这个算法中，将标量场的值赋予三维空间的网格点。通过决定层次曲面和网格边的交点可以逼近一个特定的层次曲面[⊖]。考虑相邻几对相邻网格点，它们的场值包含期望值（即一个顶点的值大于期望值，另外一个顶点的值小于）。层次曲面和边相交的位置通过线性插值来估计。

现在，在网格上的每一个立方体有些边以相交点标记。根据在边上相交点的排列可以分为256种情况（每一个立方体的8个顶点之一或者大于或者小于目标值，给出$2^8 = 256$ 种可能的排列）。对于每一种情况，选择一个在立方体内部填入层次曲面的方法，图20-20显示了两种情况。 1035

所有定义的曲面片的集合构成一个曲面，可以对这个曲面（在每一个子多边形）赋予一个法向量，为在下面的明暗处理做准备。对于立方体上的每一个顶点，对标量场的梯度都进行一个数值估计。插值这些值去估计子面片上一些点的梯度向量。由于一

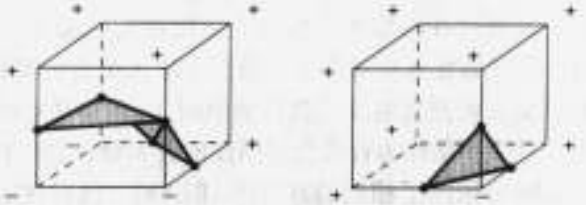


图20-20 两个可能的层次曲面和整数网格中立方体相交的排列，对每一个情况选择一种填充曲面的方式

⊖ 网格是空间点和线的阵列。网格点在x，y，z三个方向等距分布，由平行于坐标轴的线段连接。由整数坐标点和平行于轴的线段连接构成的集合叫整数网格。

个标量场的梯度总是沿着层次曲面法向量的方向，这个插值得到的值为法向量提供一个很好的近似。（零的特殊情况必须单独处理。）

生成的层次曲面可以使用通常的方法绘制。这种策略在医学图像处理中显示不同组织边界形状是非常有用的。令人遗憾的是，一次它只能计算一个形状，不同层次的相对位置很难看到。

Upson和Keeler[UPSO88]也假设标量场在采样点之间是线性变化，他们提供两种显示方法。在这两种方法中，用户首先创建四个函数R，G，B和O，在这里O是不透明度（opacity）。这些函数的变量是标量场的值，因此假设标量场的值已经规格化到0和1之间。R，G，B和O函数的选择对结果图像的影响非常大。如果被选择的函数在标量场的特定值处有窄的尖峰，有这些值的层次曲面就被高光所照。如果被选择的函数在场值上变化比较平缓，那么颜色可以用来显示场的值（见图20-21）。因此，事实上，我们获得了多色的伪彩图。

对采样点网格上的每一个立方体的标量场插值是在每一变量上的线性方程，因此在三维空间是三线性插值（也就是具有$S(x, y, z) = A + Bx + Cy + Dz + Exy + Fxz + Gyz + Hxyz$的形式）。如果我们以$(x, y, z) = (a, b, c) + t(u, v, w)$形式参数化光线，在光线上点的$S$值是$t$的三次函数。

快速计算三次方程的能力形成Upson和Keeler第一个绘制方法的基础，这个方法是基于[KAJI84]开发的对体数据绘制的光线跟踪机制。对于每一条从眼睛出发穿过图像像素的光线，当它穿过体数据时，R，G，B和O值被累加。当不透明度增加到1或者光线穿出体空间，不论何者先发生，累加过程停止。实际上，不仅仅是累加：在一个像素体中的标量场、明暗处理函数、不透明度和深度提示等在过程中的每一步都被计算以便精确积分三次插值。

1036

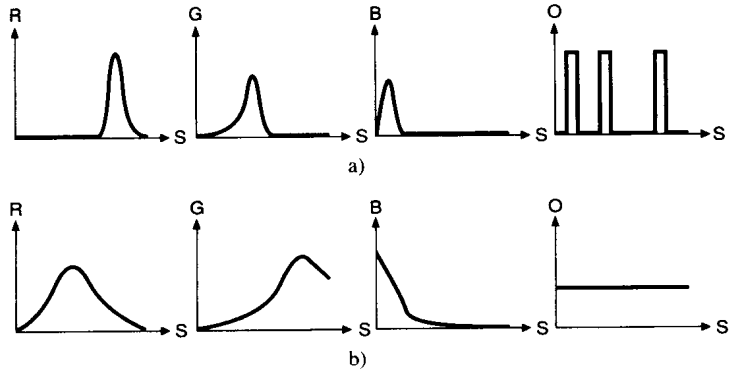


图20-21 对R，G，B和O函数形状的两种不同选择。在图a)中标量场的某个层次曲面由红、绿和蓝的高光绘出，在图b)中颜色作为标量场的函数逐渐变化

Upson和Keeler的第二种体绘制方法采用同样基本的沿光线积分的想法，但是累加像素值的方式是通过在网格中立方体上从前往后处理（对于任何特定的视方向这是很容易确定的）。两位作者在确保计算效率方面做出了巨大努力，对积分采用自适应的积分法，并且在一个点处决不重复求解方程组（当执行插值时）。值得注意的是，如同他们所说，这种方法是作为一种分析工具而不是作为合成真实图像的技术设计的[UPSO88, p.64]。

Sabella采用了相似的方法[SABE88]。他对在需要绘制的空间的每一点赋予一个密度发射器，用来模拟来自于半透明体的光。这种模拟只是对光在这种介质中的部分效果进行模拟，也就是说，在介质中较深的部分被靠近前面的部分遮挡。Sabella故意忽略了阴影和由于不同波长的光的散射的不同而引起的颜色的变化，认定上述这些从密度变化的角度来看实际上微不足道。密度发射器可被想像为微粒子，它既发射光又散射光。在空间一个小区域，这些微粒的密度是

由相应位置附近的标量场的值给出的。沿着任何光线到达眼睛的光通过下述结果计算：累积沿着光线的所有发射器的辐射值，然后根据概率，削弱从每一个发射器发出的光，即它在传播到眼睛之前是散射的。Sabella计算四个数：M，沿着光线的标量场的峰值；D，到峰值时的距离；I，刚刚提到的削弱强度；C，沿着光线的密度发射器的重心。通过将这些数字不同的组合映射到不同的颜色空间（例如，当采用HSV模型时，他映射M为颜色，D为饱和度，I为亮度），他可以高亮度显示标量场的不同特征。通过给粒子辐射的方向性，他进一步给出粒子光照的效果。每个粒子的辐射根据朗伯光照模型来减弱：几个光源置于要绘制的体的周围，一个在位置（x, y, z）处的粒子的辐射是由下面所计算的值决定的，首先将该点标量场的梯度和光照的方向 1037
的点积进行累加，然后将上述累加结果乘以那点的密度。这样所得到的结果使高密度的曲面看起来更像反射面，这种效果可以帮助眼睛消除所表示的信息的岐义性。

进一步提取表面的方法是由Pixar 公司的Drebin、Carpenter和Hanrahan 提出的[DREB88]。这些研究者针对要绘制的标量场做出几个重要的假设：表达数据场的体数组假设大约以奈奎斯特频率采样（或者这个场在采样之前已经滤波以保证这一点）；数据场是由一种或几种物质（例如，骨、脂肪和软组织）的标量场的组合体或者体空间有几个标量场相关联，例如在某种材料中的应力和应变。对于多种材料的标量场，假设在每一点（至少统计上）可由标量值区分，或者假设关于每一个体元的材料组合的信息是额外提供的。

给定了这些信息之后，他们在采样点阵列上创建了几个新的标量场：*各种材料的百分比体*（他们在此使用术语体是指体的一个标量场）。在材料的百分比体中一个网格点上的值是一种材料在包围这个点的体素中的百分比。如果在原始的数据中指定了多个场，计算这些物质的百分比是非常简单的。如果仅仅给出一个场，不同材料的百分比值可能必须由Bayesian分析来估计。

在计算多种材料的百分比体之后，上述作者把颜色和不透明度与每种材料联系起来。通过对每一种材料的百分比体的颜色和不透明度线性组合来形成合成的颜色和不透明度的值。（这里的不透明度和在17.6中描述的α通道的概念一样，线性组合和在那描述的线性组合是一样的。特别提到的是在组合之前需要将颜色和不透明度的值相乘。）他们进一步允许和*不光滑体*进行组合，这是些在体空间上场值介于0和1之间的标量场。通过使用颜色和不透明度与这些不光滑体相乘，可以获得原始的体数据的切片或者一部分。在0和1之间进行一个平滑过渡使得在不平滑的边界保持数据的连续性。

在这用的光照模型和其他两个算法中的模型是相似的。一定量的光进入每一个体素（从给定的体素后面的体素出来的光），不同量的光从这个体素穿出。光的改变可受体素中材料的透明度的影响，或者说是受体素中包含的曲面和粒子散射器等的影响，体素既可以衰减入射的光又可以反射从外部光源来的光。这些效果可以采用以下两个手段来模拟：(1)要求穿过有颜色的透明体素的光有这样的颜色——体素的颜色加上入射光的颜色乘以$(1-\alpha)$（这是一个17.6节中的Feibush-Levoy-Cook组合模型的**over**操作）；(2)确定曲面和它们的反射和透射特性。

曲面的判定不如以前所描述的方法那么准确。每个体素被赋予一个密度，它是各部分材料密度的加权平均（权值是材料所占的百分比）。“曲面”就是那些组合密度变化大的地方。一个曲面 1038
的*强度*是密度梯度的*幅度*，在明暗计算时曲面的法向是梯度的方向向量。为了计算表面的明暗度，把所有体素分为在曲面的前面、曲面上和曲面的后面。离开体素的光的亮度(I')和进入的光的亮度(I)相关，根据的规则是$I'=(C_{\text{front}}\ \textbf{over}\ (C_{\text{surface}}\ \textbf{over}\ (C_{\text{back}}\ \textbf{over}\ I)))$。因为**over**算子是关联的，所以和体素相关的三个项可以提前计算和混合。曲面颜色是通过Cook-Torrance风格的模型计算以给出镜面和漫射分量。这些颜色值是用曲面的强度加权以致于在各向同性物体中没有任何反射光。

曲面前后的颜色的计算是通过从光线穿过的材料来估计的，并且使用这些材料的颜色。

结果非常令人满意。彩图I-1显示将这个办法应用于一个小孩头部CT扫描数据的结果。可是处理过程是昂贵的。生成图像的过程产生了多重体空间（也就是多重标量场），并且存储量要求也是非常大的。关于数据场是以等于或高于奈奎斯特频率采样的假设并非在所有的情况下都是实际的：有时数据是给定的，我们希望看到结果，即使有走样。最后，关于数据是来自于匀质材料混合的假设也不总是有效的，因此，上述方法的应用是受到限制的。

20.7 基于物理的建模

许多物体的行为和形状是由物体宏观的物理特性所决定的（这和生物系统对比是非常明显的，生物系统的行为可能是由系统的化学和微观物理特性决定的）。例如，一块布怎样覆盖在物体上是由表面的摩擦力、织物以及物体的力产生的内部应力和应变所决定的。悬挂在两个柱子之间的链子表现出的弧形是由重力和相邻的链子为保持不分离的拉力决定的。*基于物理的建模*技术就是使用这些物理特性去决定物体的形状（甚至在某些情况下，决定它们的运动）。关于这个主题的当前的工作收集在[BARR89]中。

大部分采用这种方法建模的技术使用的数学知识已经大大的超过了本书的范围，但是我们可以给出这些技术的一般概念。正是在这类建模技术中，图形学和其他科学的区分是非常模糊的。对一块薄布落到一个障碍物上产生撕裂的计算纯粹属于实体力学中的问题。可是这样的计算可能不会做，除非能够以某种方式显示其结果，所以物理研究的动机是由对结果的可视化的能力（或者欲望）提供的。同时，生成更真实感图形模型的希望给基于物理建模过程的研究提供了动力。在本节，我们讨论几个更为引人入胜的例子。下一节描述自然现象的模型。为了生成吸引人的结果，这些模型并不直接基于多少科学原理并且有一些（或很多）折衷。在科学
1039 基础和现成方法之间存在一个连续的变化，它们的分界线根本不明显。

20.7.1 基于约束的建模

当我们从基本对象用布尔操作建造一个对象时，我们发现能够说“我想把这个球放在这个立方体的上表面使它们只有一个点接触”这样的话是非常方便的。即使在允许用户使用眼睛定位物体的交互程序中，使两个物体在一个点[⊖]处接触可能也是非常困难的。像这样的规则被称为*约束*。基于约束的建模系统允许用户指定一个模型每一部分所要满足的约束的集合。一个模型可能是*欠约束的*，在此情况下存在建模者可以调节的附加的自由度（例如球面和立方体接触的点的位置）；一个模型也可能是*过约束的*，在此情况下某些约束也许不能够被满足（如果球的上面和下面都需要位于立方体的上方时，就会发生）。在基于约束的建模技术中，必须给出每一个约束的优先级，以便重要的约束首先得到满足。

指定约束是非常复杂的。某些约束可以以数学等式（例如，两个物体被限制在特定的点接触）或不等式（例如，一个物体被限制在另外一个物体内部）方式给出。其他约束更难以指定。例如，限制一个物体的运动以便满足物理学的运动规律需要指定一组微分方程的集合。而这类约束系统恰恰是基于物理建模的中心。

最早的基于约束的建模技术是Sutherland在Sketchpad系统中提出的，在第21章将详细陈述。自从那时开始，研究人员已经开发了许多基于约束的建模系统，包括基于约束的人体骨骼模型[ZELT82；KORE82；BADL87]。在这个模型中，骨骼的连接和关节处的角运动的限制是由约束控制的。此外，还包括[BARR88]提出的*动态约束系统*，[WITK87；WILH87]的*能量约束系*

⊖ 键入数字也不是好方法，可能需要建模系统在此之前求解一个方程组。

统。这些系统可以分为两大类：一类可以指定一般的约束，另外一类给定特定的约束集合。例如，在骨骼建模中，在两块骨骼上对应点相互接触的点对点约束是一般约束；在一个关节处两块骨骼间相交的角度被限制在一个特定的范围的约束也是一般约束。但是关于指定两个物体质心距离满足最小的约束不是很可能发生的。专用的约束系统可能允许特殊类型约束的分析结果，然而通用的约束系统更可能使用数值方法。

例如，在上面提到的能量约束系统中，约束被表示为函数，这些函数在任何地方都是非负值，只有当约束满足时才精确为零（它们是在被建模物体所有可能状态的集合上的函数）。这些函数累加起来形成一个函数E。一个约束问题的解是这样的一个状态，在这个状态下E为零。由于零是E的最小值（因为它的所有分量都是非负），我们可以从任意的初始状态开始，通过改变状态以缩小E的值求得满足E最小的这些状态。采用数值方法找到这个最小值。在这个寻找最小值的过程中，E有可能陷入局部最小值，但是如果没有陷入局部最小值，我们就可以它逐渐达到全局最小值。这样一个全局最小值或者是零，在此情况下所有的约束都满足；或者非零，此时某些约束可能不满足。通过改变函数E中的单个约束的相关系数，我们可以强调一些约束比其他约束重要。在系统陷入局部最小值的情况时，建模系统应该从一个新的初始状态开始，或者，在一个理想的基于约束的建模系统中，应该给状态一个“推动”使之从局部最小值移出并且朝向全局最小值移动。当状态集合向函数E的最小值移动时产生的状态序列可能是非常有意义的动画，尽管初始的意图仅仅是模拟一个状态集合使之满足尽可能多的约束。事实上，事件序列动画在决定被使用函数最小值算法的特征时可能是非常有用的。 1040

在20.9节和第21章将进一步描述基于约束建模的例子。

20.7.2 布面和柔软表面的建模

近年来，已经开发了几个为布面和其他类似表面的建模方法[WEIL86；WEIL87；TERZ88]。Weil假设布面是矩形的丝织物，每根线是没有弹性的。在丝织物表面上一个点的经线和纬线提供了一个坐标系统，在这个系统中描述这个布面内部的事件，而每一个点都有一个三维的坐标位置。Weil模型的第一个假设是在三维空间的特定位置挂住布面特定的点将其悬挂；因此这块布面的初始位置可以由一些有限数目的点确定。在任意两点（内部坐标系统中）之间的线假设映射为悬链曲线（它是一个链子悬挂时的形状）。这决定一块布面上几条线的位置。注意到在一个有两条线穿过的点，相交点的位置是超定的。在任何这样的情况下，Weil忽略下面的线。在表面上悬挂点之间的线决定在布面上的区域，每一个区域都用很多曲线来填充的。布面的（至少初始的）形状现在确定了。目前为止，布面的结构已经被忽略：构造布面的丝线或许被拉伸了，而它们是被假设为没有弹性的。Weil进一步采用松弛过程递归地移动这些点以缓解丝线中的张力（通过计算每一点和它的邻居之间的方向向量）。这些向量与它们自己的长度相乘然后平均来为计算每个点的位移（相乘确保大的误差有大的结果）。这个过程递归地进行直到得到的曲面满足了约束为止。一个相似的方法用来模拟布面的刚度。彩图IV-15显示了这个模型和[WEIL87]中描述的改进模型的结果。

Terzopoulos和Fleischer[TERZ88]采用了一个更高级的方法，并且它能模拟比布面更一般的介质。他们假设一种材料以网格方式排列（可能是三维，但布面是二维），并且在网格上的相邻点是由弹簧、减震器（它可能是震动吸收器）和塑性滑动单元构成的单元所连接的。弹簧受外力时所发生的形变和力的大小成正比，力消失时，形变也消失。减震器对力的反映是它的变形是以一定速率正比于力的大小。因此，一个常力引起减震器的延伸，直到力消失的那一刻，减震器才停止变形。一个塑性滑动单元对力的反应是直到力达到一定大小时，它才发生形变， 1041

接着发生自由滑动；这种单元最好和其他单元综合使用，例如弹簧单元。以图20-22所示的放置一个塑性滑动单元和两个弹簧创建了一个单元，该单元逐渐延伸（包括弹簧的延伸、塑性单元的静止）直到在塑性单元上的力达到它的阈值。在这一个时刻，塑性单元开始滑动并且和它相连的弹簧暂时收缩，直到其他弹簧承担了一些负载为止。因此，达到这一刻时，系统好像只有一个弹簧组成。一旦逐渐减小所作用的力，下面的弹簧承担一些（压力）负载，直到力充分小到能引起塑性滑动单元再次滑动为止。

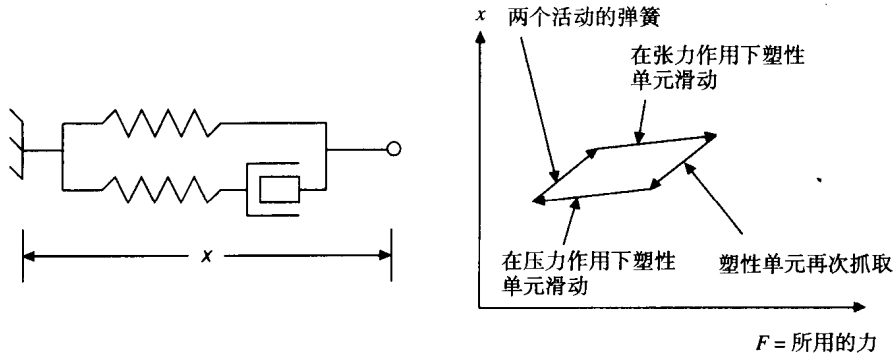


图20-22 一个塑性单元和一个弹簧连接，然后并联上另外一个弹簧，这样创建了一个新的单元，它对变形力的反映直到力达到一个阈值。接着滑动单元滑动，并且保持变形，直到一个足够大的力使这个单元恢复到原始状态

这样的单元组成的网格受到物理规律的影响（全局是由刚体动力学模拟，而局部是受到这个内部坐标系统中材料的单元结构有关的应力和应变的影响）发生形变和延伸。特别地，如果在布面上的丝线是由塑性滑动单元来模拟的，那么在张力达到某个值时，这些单元将会滑动（也就是丝线将会破裂），这样将会产生一个撕裂的结果。彩图IV-16显示这种结果的一个例子。

20.7.3 实体建模

在前一节讨论的Terzopoulos和Fleischer模型也可以用于描述线性或者立体组合的实体，它们是由不同的弹性或粘性单元连接的点的集合。通过将基于这种结构的固体力学和动力学约束组合在一起，Platt和Barr[PLAT88]在模拟可变形实体（例如，油灰和凝胶）方面做了类似的工作。他们的工作的精髓是建立一个大的微分方程的集合，这些方程决定任一时刻的粒子组合（或者
1042 有限元网格）的状态，这些状态受到如下目标的影响，即在某些约束（如物体非彼此贯穿）要满足的同时某些函数（例如能量）要达到最小。他们的实际模型把这些约束和函数的最小一样作为要达到的目标，并且这些约束仅仅是近似满足。每一个约束的作用越强，微分方程的求解就越困难。尽管有这种数值上的困难，但结果当然是足够令人印象深刻的，有理由进一步研究。

20.7.4 地形建模

在另外一个基于物理建模的例子中，Kelley和他的助手们[KELL88]拓展了来自地形学的溪流侵蚀模型。他们从支流分布、连接的长度（一个连接是两个支流汇合或者一个源和它的第一个汇合的一段）和流梯度的关系、山谷侧坡的平均坡度等的统计观察着手。他们使用这些去模拟一个给定的总初始地形的溪流模式的二维分布，然后在垂直维上改变地形以便计算适合所构造的溪流系统的精细部分。彩图IV-18显示了一个简单初始水系的结果。

20.8 模拟自然物体和合成物体的特殊模型

在用和自然现象产生原因不直接相关的技术模拟自然现象方面也做了大量的工作；使用椭

球来模拟云就是一个很好的例子。对没有特别视觉表现的自然现象（例如分子）的模拟方面也做了许多工作。本节的例子包括从科学的准确性到为生成吸引人的图像而采取的灵活处理的各个方面。这些模型被当作图形学的工具，而不是严格的科学可视化。要点在于创建这些模型的人在认识到一个好的模仿的好处的同时理解这些基本的现象。

20.8.1 波浪

波浪是在图形学中最早模拟的自然现象。从一个点或者一个线发出的涟漪和正玄波相似，而且同样是最容易模拟的。如果从眼到波浪的距离足够大，实际扰动水面可能是不必要的，涟漪的整个效果可以通过凹凸映射的方法生成（虽然最初的广泛展示的一个例子实际上是光线跟踪了整个高度场[MAX81]）。更复杂的涟漪和波浪模式可以通过使用有限带宽的噪声生成代表波浪串的纹理映射来组装[PERL85]，然后用这些去对平面的水面进行纹理映射。当然，从上面看，这些模式看起来很好，因为真实波浪的侧视图应当表明水面高度的变化。

Fournier和Reeves[FOUR86]采用了完善得多的方法，把水体表面模拟成为一个参数曲面而不是高度场，允许卷曲的波的可能性。他们不仅考虑水下地形学对水面波和障碍物附近波的反射和折射（例如波浪在防浪堤的尽头发生弯曲）的影响，而且还更多地考虑深水波理论。反过来，在 1043
模拟理论知识仍然有限的破裂的波浪中，他们提供了非常聪明的产生了非常好的结果的方法。遗憾的是在破碎处，他们生成的波浪有点过于光滑，而且在当波浪将要破碎时在主曲面和跟随的边之间缺少尖锐的边沿。尽管如此，结果还是好极了（见彩图IV-19）。Peachey[PEAC85]所做的类似工作使用不那么复杂的模型；虽然波浪的整个外表不太真实，但是对破碎的波浪模拟非常好。

20.8.2 云层和气象

雾和烟都可以采用随机的办法模拟并合成图像。为了增强真实性，我们可以用要被组合成的图像的z值来加权雾的效果，以便远处的点比近处的点模糊。类似地，通过根据从眼到光线的第一个交点（或者，更好一点的，对于非规则的雾，通过沿着光线积分雾的浓度来计算一个衰减函数）的距离的某个幂来衰减图像的方法可对雾和烟使用光线跟踪技术。这些技术有物理学中的基础，因为光线穿过雾的距离越长，它散射得越多。已经有几个截然不同的建立云和空气中烟的模型的方法。Voss[VOSS85]采用基于分形的方法生成了云，而Gardner使用了带有纹理的椭球模拟云[GARD84；GARD85]。Voss的技术是在四维空间上生成一个分形，它的第四个坐标代表了水蒸气的密度。通过允许局部光的散射随着水蒸气的密度变化，他生成了某种真实的云（他用的分形维数是3.2~3.5）。

作为对比，Gardner的方法完全根据观察到的云的形状——云看起来像薄片或者斑点，因此它们被模拟为带纹理的平面和椭球。这个模型是由天空的平面（在它之上驻留薄云层）、椭球（用来模拟厚的云，例如积云）和为它们每一个定义的一个纹理函数（这个纹理函数用来处理云和天空的不同的明暗程度和透明度）组成。

Gardner创建了一个非常简单的纹理函数，和Perlin使用的实体纹理类似。他定义

$$T(x, y, z) = k\sum_{i=1}^{n}[c_i \sin(f_i x + p_i) + T_0]\sum_{i=1}^{n}[c_i \sin(g_i y + q_i) + T_0]$$

其中c_i是纹理在不同频率的幅度，f_i和g_i分别是在x方向和y方向的频率，p_i和q_i是相应方向的相位位移。根据不同的常数这个函数有不同的特征。用$f_{i+1} = 2f_i$，$g_{i+1} = 2g_i$和$c_{i+1} = \sqrt{2}/2c_i$赋值产生在几个不同的频率时的变化，随着频率的增加幅度在减小。注意到，这与地形高度的分形模拟是多么相似：山有大的高度变化，山上的巨石变化较小，在巨石上尖角变得更小。

相位位移p_i和q_i被用来阻止所有的正弦函数相互同步，即产生随机性（如果这些被忽略，

纹理函数有一个可以看到的周期性）。对于平面纹理，Gardner建议$p_i = (\pi/2)\sin(g_i\, y/2)$，对$q_i$类似。对于椭球纹理，他定义$p_i = (\pi/2)\sin(g_i\, y/2) + (\pi\sin(f_i\, z/2))$，它在三个方向都产生相位位移，并且他发现使用$0 \leqslant i \leqslant 6$值能提供丰富的纹理。

1044 这个值的集合定义了纹理函数。必须把纹理函数和天空平面或云的椭球组合起来以生成图像。Gardner使用如下形式的光照模型：

$$I_1 = (1 - s)\, I_d + sI_s,\ I_2 = (1 - t)\, I_1 + tI_t,\ I = (1 - a)\, I_2 + a$$

其中I_d和I_s分别是亮度的镜面反射和朗伯反射分量，如在第16章所计算；I_t是$T(x, y, z)$；a, t和s分别决定了环境光、纹理和镜面反射部分。此外，为了获得云而不是把云贴在其上的椭圆的效果，云的边沿必须半透明。这是通过定义一个半透明度V来做到的，它的规则为：

$$V = \begin{cases} 0 & I_t \geqslant V_1 + D \\ 1 - (I_t - V_1)/D & V_1 + D > I_t \geqslant V_1 \\ 1 & \text{其他} \end{cases}$$

其中V_1和D一起决定半透明度在0和1之间的变化范围：当$I_t = V_1$时透明度为1；当$I_t = V_1 + D$时半透明度减小为零。这对天空平面中的云来说是足够的，但是对于一个椭球的云，我们希望边沿的半透明度比中心的半透明度高。Gardner定义了一个函数$g()$，在椭球投影到胶片平面的中心为1并且在投影边沿为0。用这个函数，可以为椭球创建一个不同的半透明度函数V，用两个不同的值V_1和V_2决定边沿和中心的透明度阈值：

$$V = 1 - (I_t - V_1 - (V_2 - V_1)(1 - g()))/D$$

这个值必须限制在0到1之间。组合光照模型和半透明度模型可以给出极真实的云（特别是如果它们聚集得很好）。

比云有更少物质的空气效果（例如烟、灰尘和雾）已经有人使用散射模型生成，比较典型的假设是光在任何一个小的体积内很少发生散射。Blinn的土星环模型[BLIN82a]通过考虑散射的四个方面处理由小球微粒构成的几乎是平面的散射层的特殊情况：

1) *相位函数*——一个小球状粒子反射入射光到观察者眼中的方式和月亮将太阳光反射给我们的方式是一样的，它依赖于地球、太阳和月亮之间的相对位置。

2) *低反照率*——如果每一个粒子的反射率都很低，那么多重散射效果（也就是光在两个或者多个粒子之间来回反射）是没有意义的。

3) *明暗处理和掩模*——距离光源比较远的粒子被在它们前面的粒子遮挡，从一个粒子发射的光被在它和观察者之间的粒子衰减，两个衰减都是粒子层所处深度的指数函数。

4) *透明度*——云层的透明度可以描述为一条穿过它的光线碰不到粒子的概率，透明度是在
1045 这个层中的光线长度的指数函数的反函数。

Max[MAX86]通过吸收由Crow[CROW77a]提出的阴影体而拓展了这个模型。他计算到达眼睛中的光是来自一个表面，这些光是在这个表面反射的光并且加上了由中间的空气反射的部分，就像Blinn的模型一样；但是一部分中间的空气（那些阴影体中的部分）不反射另外的光。这在反射空气中生成一个明暗柱(或光柱)的现象，就像人们在一个有尘土的房间见到一束光通过窗户进来一样。Nishita、Miyawaki和Nakamae[NISH87]开发了一个相似的技术，它处理多个光源、具有变化的亮度的光源（稍后将讨论）和密度变化的散射介质。他们的技术是根据对于光线跟踪绘制中的每一条光线准确穿过的被照到的空气的体和来自每个这样的面片的照明情况决定的。他们结合了一个不同于Blinn模型的相位函数，这个相位函数是根据对于相对小的粒

子（例如灰尘和雾）的更复杂的散射理论的一个逼近。

沿着同一个方向的进一步工作是Rushmeier和Torrance的将辐射度模型推广到处理散射[RUSH87]，辐射度模型基于与热传导类似的理论。在他们的模型中，空间中的每一个体（它被分解为小的立方体）被处理为不同的辐射单元，不仅考虑面和面的相互作用，而且考虑面和体及体和体之间的相互作用。这可能产生很复杂的方程组，但是结果非常真实——它们构成了几个目前由计算机图形学产生的最动人的图像（见彩图IV-20）。

Nishita和Nakamae也研究了散射对照明的影响：一个也许非常纯正的方向光源（像在月亮表面的太阳光）可能由大气（例如地球的大气）漫射而成为一个散射的照明光源。一个在户外地面上的物体不仅由直射阳光照射，而且也由来自于（空气散射的）天空中的其他区域的光源照射。Nishita和Nakamae把整个天空模拟为一个具有变化的亮度的半球形光源，接着通过积分整个半球计算对每一个物体的光照。彩图III-22c显示被这个半球照射的内部场景。

20.8.3 湍流

多年来，对湍流建立精确的数学模型一直是非常令人感兴趣的，现在已有非常好的流体力学模拟程序。这些已经被直接应用到湍流建模上了，如Yeager和Upson[YEAG86]所做的工作；另外可用更多的经验模型来产生湍流的良好逼近效果，如Perlin[PERL85]所做的那样。Perlin的模型用实体纹理（见20.1.2节）的形式模拟是特别简单的。在点$p = (x, y, z)$处的湍流是通过累加各个不同频率Noise()函数集合生成的；伪代码在图20-23给出。

通过定义Marble(x, y, z) = MarbleColor(sin(x + Turbulence(x, y, z)))，这里MarbleColor将 − 1到1之间的数值映射为大理石的颜色值，所得到的函数Tubulence()可用于产生大理石纹理。在sin()函数中的x是被用来生成一个平滑变化的函数，然后它被湍流函数来扰动的。如果MarbleColor在一些点有足够高的导数（也就是足够大的亮度变化），那么在基体和穿过它的纹理之间会有尖锐的边界（见彩图IV-21）。 1046

```
double Turbulence (double x, double y, double z)
{
    double turb = 0.0;      /* 半透明度是Noise()项的和 */
    double s = 1.0;         /* s = 噪声的缩放比；1 = 整个图像 */

    while (S比像素大小要大) {
        turb += fabs (s * Noise ( x/s, y/s, z/s) );
        s /= 2.0;
    }
    return turb;
}  /* Turbulence */
```

图20-23 生成湍流函数的伪代码

20.8.4 滴状物体

分子建模的典型方法是采用球和杆。但是实际的分子物理学揭示出围绕原子的电子云不是球形的，而是被相互存在的原子所扭曲的（也包括其他的影响）。为了获得更好的常量电子密度的表面图像，我们必须考虑附近原子的影响。同样，对于任何一个物体的集合（每一个物体都创建一个球对称的标量场并且它们的场互相叠加）都有不是用互相重叠的球的集合模拟而是用一些更复杂的形状模拟的等值面（沿着这个面的场是常数）。计算准确的等值面可能是不实际的，但是已经有了几个好的逼近方法。这一工作首先是由Blinn[BLIN82b]和Nishimura 等人

[NISH83a]各自独立完成的，在Blinn的系统中每一个物体生成的场是按距离的指数函数来衰减，而Nishimura等人的工作用于LINKS项目中。Wyvill、McPheeters和Wyvill[WYVI86]将Blinn的技术修改得非常漂亮。他们通过在空间放置一个场源的集合并且计算在空间的每一点的场的值来模拟“软物体”。集合的场值是每一个场源的场值的和，并且每一个场源的值都只是距离的函数。他们使用了一个在有限距离R内完全衰减的距离函数，而不像Blinn的指数衰减。他们的函数：

$$C(r) = \begin{cases} -(\frac{4}{9})r^6/R^6 + (\frac{17}{9})r^4/R^4 - (\frac{22}{9})r^2/R^2 + 1 & 0 \leqslant r \leqslant R \\ 0 & R < r \end{cases}$$

有如下特性：$C(0) = 1$，$C(R) = 0$，$C'(0) = 0$，$C'(R) = 0$和$C(R/2) = 1/2$。图20-24显示一个$C(r)$的图。这些特性确保将混合在一起的曲面平滑连接，并且场的定义具有有限的范围。他们计算一个值m，它有如下特性：$C(r) \geqslant m$时集合的体积恰好是$2C(r) \geqslant m$的集合体积的一半。如果两个场源被放在同一个位值并且层次m的等值面被构造，那么它有两倍于单一一个场源的等值面的体积。因此，当软物体合并时，它们的体积相加。（注意到，如果两个场源分开较远，等值面将会有两个分离的部分。）

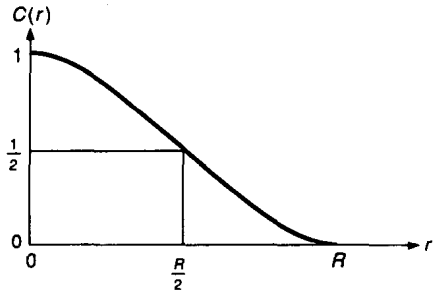


图20-24 函数$C(r)$

一个场的等值面可以使用类似于在20.6节中讨论的移动立方体算法的算法计算，而且它非常快。通过估计从一个场源延伸的轴上的网格点序列上的场，我们求得一条边与等值面相交的一个立方体（这个边位于值大于m的最后一个网格点和值小于m第一个网格点之间）。因为每一个场源的场值随着距离的增加而减小，这个立方体的集合（被称为*种子立方体*）有这样的特性：每个等值面片至少和其中的一个相交。因此，通过从这些种子立方体向外求解，我们可定位整个层次曲面。通过标记已经被处理过的立方体（这些标志和不同的函数值一起，可以存储在一张散列表中以防止有过分大的数据结构）可以避免额外的工作。（计算隐式面的另外一种方法在[BLOO88]中给出。）

使用这个技术模拟的物体非常像泥塑模型[WYVI88]，并且能够用来建立分子的模型或者模拟流动到另一种流体中的液滴。例子参见彩图IV-22。

20.8.5 生物

在20.4节使用基本L文法描述的植物是比较简单的生物，它们规则的形状使其易于模拟。也已经有人对贝壳和珊瑚[KAWA82]及想像中的生物[KAWA88]建模等进行了研究。近来，通过使用基于物理的技术对一些在生物学上较简单的动物做了模拟，但是这些模型是以牺牲某些生物真实性来换得好的图像的[MILL88a]。正如Miller提到的，“模拟像蠕虫这样的生物的一个最大优点是没有人想仔细地看它们”[MILL88b]。尽管生物和物理学上真实的模型的计算代价仍然是受限制的，但是Miller的观察的含义是非常重要的——对外观的关注对模拟一个场景的周边环境可能完全足够了。

1047 ~ 1048

Miller建立的蠕虫和蛇的模型是基于质量和弹簧的相互作用的，肌肉收缩是用弹簧张力的变化来模拟的。蠕虫和蛇的前进运动是通过添加在某个方向的摩擦力（它们身体的每一段只允许向前移动，而这个力试图将生物的尾部向后移动）。Miller用凹凸映射和模式映射模拟了蛇和蠕虫的外观。他也通过下述方法生成了毛毛虫的皮发，即在其身体上随机分布毛发根，对每一

根毛发在根据发根处表面的法向和切向所决定的局部坐标系中确定发稍的分布。因此，他可以模拟有方向的毛发。他绘制的一个蛇的模型如彩图IV-23所示。

Reynolds[REYN87]已经对大批鸟的飞行聚集和鱼群进行了建模。在这个模型中行为的模拟非常好，以至于仅仅这些生物的粗糙的外表让人觉得有一点缺憾。当对高级智能生物进行模拟时，精度的需求就增加了，因为我们对这些生物的熟悉程度使忽略这种模型的缺点是不可能的。

20.8.6 人

对人的建模是最终的前沿。我们识别和区分人脸的能力是令人惊讶的；人的计算机图形学的图像必须非常可信以满足我们对真实性的要求。模拟一房间真实物体要比模拟一个真实的人脸要容易得多。

一段时间以来人们已经认识到建立有关人的模型的需要。许多我们熟悉的场景都有人在里面，所以用一种比较令人满意的方式建立这些人的模型是非常有用的。最终的目标是将人在计算机生成的动画中从作为“附加”使用到以“小部分”使用以至于最后到作为主要角色使用。在这个领域已经取得了一定的进步。Catmull[CATM72]已经把手模拟为多边形物体。手的各个部分（手指、单个的关节等）都是以层次方法构造的，因此运动一个手指就是运动它的所有关节。进一步说，一个关节上的每一点或都被描述为关节本身的一部分或者在其父层次上描述（如和手掌更近的下一个关节）。因此，当父层次移动时，关节的形状也会改变。

Parke[PARK82]、Platt和Badler[PLAT81]以及Waters[WATE87]都对脸部建模做了研究。Waters把脸模拟成为相连的多边形网，它的位置是由几块肌肉运动决定的；这些肌肉被模拟为薄片并且可以收缩。有些肌肉薄片固定到在头部的一个固定点，有些嵌入到皮肤的组织中。前者的动作是向固定点收缩，而后者在它们自己内部收缩。脸部的多边形是通过给定其顶点作为肌肉薄片的点来建立模型的。激活一块肌肉将会使脸变形以形成一个表情。这种方法是Platt和Badler对类似的模型的一个改进，原方法将肌肉模拟作为可以收缩的线的网格而不是薄片。Parke还通过既允许控制表情又允许控制形态（使得一个人脸不同于其他人脸的特征）扩展了这个工作。在所有这些模型中，一个基本的特征是使控制人脸的表情的参数减少为几个参数，以便建模者不必显式地放置每个多边形的每个顶点。

Zeltzer[ZELT82]在模拟有骨骼生物的运动方面已经做了扩展工作。真实的骨骼结构是相当 1049
简单的（一个刚体的层次连接的集合）；只是运动的模拟是更困难的。最近由Girard[GIRA87]所做的在有腿动物运动方面的工作是非常吸引人的。运动的模拟将在第21章进一步讨论。

20.8.7 来自于娱乐业的一个例子

最后一个为特殊目的建模的例子来自娱乐业，在这个行业中计算机图形学应用很广泛。在电影《Young Sherlock Holmes》中，有这样一个场景，一个牧师产生一个幻觉，他被一个玻璃骑士攻击，而这个玻璃骑士是从一个彩色的玻璃窗户跳出来的。由于控制骑士运动的每一块盔甲透过半透明的玻璃都应当可见，对于这种效果如果使用传统的动画技术是相当困难的，因而用到了计算机图形学。

在彩图IV-24的一系列图像显示了玻璃建模中涉及的不同技术。实际上所有的这些技术都是通过修改使用模式映射和凹凸映射技术的反射函数来完成的。彩图IV-24a显示单片玻璃（护肩），用颜色图定义金色条文。彩图IV-24b使用了一个环境图显示在玻璃之后的教堂的场景。彩图IV-24c同时应用一个光照函数和一个凹凸图，它们一起修改彩图IV-24b的环境图，使环境看起来像通过玻璃发生折射。拱形的形状依然可见。彩图IV-24d将脏点和小气泡加到前面的效果上。彩图IV-24e中，附加的凹凸图描述在玻璃前面和玻璃的右边沿的不均匀的表面。彩图IV-24f显示了物体的细节信息。总的来说，为了给出玻璃的真实外观，需要三个颜色图、三个凹凸图、一个透明图和一个环境图。彩图IV-24g部分显示了完整的图像；这块护肩在右上方。

玻璃片装配成为一个层次模型并且使用三维关键帧动画程序使其运动，“聚光灯”精心地放置在场景中，以便恰在武士刺向牧师时他的剑上有闪光。在一个镜头中，拍摄运动动作的电影摄相机正在移动，因此记录计算机生成的动作的合成相机也必须移动，以便精确地匹配电影摄相机的运动。最后的效果是非常吸引人的，并且在一个实例中是非常令人吃惊的：当合成相机绕到后面去看骑士的身后时，我们看到同样的移动，而且看到的不是骑士的头的后部，而是他的脸。这给移动赋予了一个离奇的效果，因为肢体似乎发生了错误的弯曲。

20.9 自动放置物体

本章的大部分内容是讨论物体的创建；有些物体（如使用腐蚀的办法建立的地形模型）构成一个场景的环境，但是它们中的大部分需要被放到一个场景中去。一个人体模型的建模者选择一个位置然后在那放一棵树、一面旗或者一块手帕。然而在一个场景中很多物体需要放置时，建模过程的某些自动处理可能是必需的。考虑另外一个时间维的情况，我们看这种情况：当单
1050 个物体在两个时间的位置是知道的，但是它在所有中间时间的位置可能需要确定。这实在是一个动画的主题，它涉及到对于在时间上位置和属性都变化的物体建模，如将在第21章将进一步讨论的那样。对于能量最小合成的真实运动被模拟的场合，我们可以自动插值中间的动画（人类运动或许具有这种形式，因为人经常试图以可能最有效的方式从一个位置移动到另一个位置）。我们也将讨论这种特定情况的物体放置。

在场景中自动放置物体的问题还没有被广泛地研究。Reeves和Blau[REEV85]讨论了一种特殊情形，在这个特殊的情形下，森林中的树通过运用一个通用的概率规则自动放置。建模者提供一个决定树之间的间距的网格尺寸和一个决定两棵树之间最小距离的参数，要被森林覆盖的水平平面的区域以及在这区域上的表面轮廓(它决定这些树基的高度)。这个程序在每一个网格点上至多生成一棵树，然后通过随机地在x和y方向上移植这些树以免在最后的结果看起来像网格的构造。对于一个网格，如果经过移植之后，新树和其他树挨得太近，那么它被删掉，继续处理下一个网格点。这个模型不太真实：树的放置位置有点随机，并且森林的密度基本上是常数，所以一个人在一个区域很难看到很多树。Reeves和Blau也让树的放置影响单个树的建模。树的海拔(概率性地)决定树是落叶的（低海拔）还是常青的（高海拔）。这种在地形和树之间的互相影响在形式上和Amburn、Grant和Whitted[AMBU86]的交互过程模型相似，后者在20.2节中描述过，其中地形的特征影响树的放置。

Gardner[GARD84]使用一个既包括随机放置又包括和地形交互的机制，并且也形成物体的聚类而不是规则的网格。为了决定特征在场景中的放置（例如，在哪里放一棵树），他使用一个特别像在他的云模型中使用过的纹理函数的函数。当这个“纹理”函数在某个临界值之上时，产生一个特征。利用这个技术，Gardner生成了一些特别真实的场景中的特征分布（彩图IV-25）。

在所有这些情况中，既要避免规则性又要避免完全的随机性是非常重要的。还有很多工作要做，但对这样的应用，能够和环境相互作用的随机控制机制看起来将会提供很好的结果。

另外一种类型的自动物体放置是确定在具有约束的物体动画中的中间阶段。在某些情况下，动画过程中一个物体的位置完全由物理学来决定；实际计算这些位置是非常困难的。Witkin和Kass描述了一个决定这些中间位置的方法[WITK88]。基本的思想是简单的：假设一个物体已经被一个由不同的肌肉（组合中可以产生能量的部分）去移动其他部分的物理组合体模拟，那么我们能够把组合体的所有部分的状态（位置和速度）描述为时间的函数（这些状态包括在任意时刻被每块肌肉所耗费的能量，这个能量和肌肉的张力相关）。这个函数可被认为取一个时

间的值t，它在一个初始和结束时刻之间，并且将一个描述组合体状态的数的集合与t相关联。因此，这个函数可以被认为是一个通过高维空间的路径。（空间的维数大约是在组合体中自由度的两倍。）在所有这些函数的集合中，有些函数的总能量耗费比其他的低。也有些函数，它们的一些部分初始位置就是要求的初始位置，并且它的结束位置就是所要求的结束位置，然后，我们能够测量一条路径离开这些条件多远。有些函数将代表在物理上可能的事件序列（例如，在某些路径上，在缺少外力的情况下，每一部分的惯量和这部分的位置的导数成正比）。

为了计算物体随时间变化的路径，我们现在采取了一个被称为变分法的方法，它和找到普通实值函数最小值的梯度方法相似。我们以任意路径开始并且通过在路经上沿着某一方向移动特定的点来稍微地改变它。我们现在判断这个路径是接近好的路径（这里“好”意味着“低能量消耗”、“满足物理规律”和“开始和结束条件满足”）还是远离。如果它接近好的路径，它就成为我们的一条新路径，然后重复这个操作。如果它是远离的，那么我们使用一个反向的扰 1051
动改变原始的路径，并且使它成为我们的一条新的路径。当我们重复执行这个过程时，我们与满足约束的低能量路径越来越近。一旦我们得到一条满足约束的路径，我们就可继续这个过程，直到我们达到尽可能最低能量的路径。它将产生这样的结果，在任何时候对路径的最佳改变都可找到，所以我们很快达到一个最小能量路径。

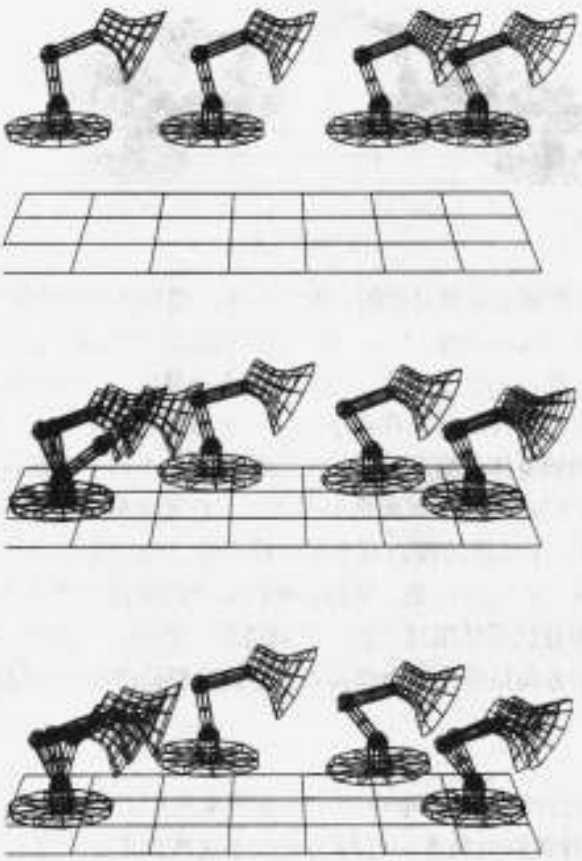


图20-25 要求Luxo Jr.从桌子上的一个位置跳到另一个位置。初始的路径已经被指定，即Luxo在桌子上面移动。一个变分技术的迭代导致Luxo找到一个下蹲-伸展-连续起飞的方法去移动，使得能量最小，并且满足约束。（经Michael Kass 和Andrew Witkin许可。）

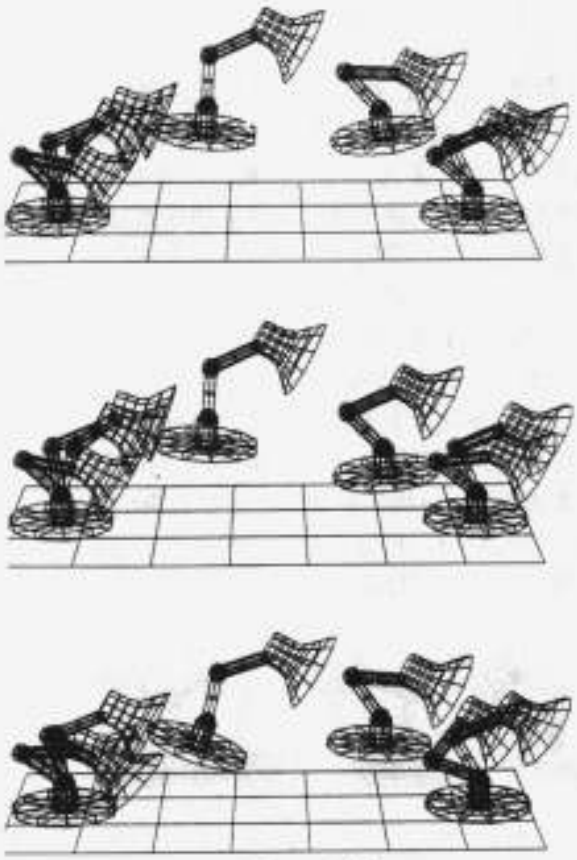


图20-25（续）

1052
-
1053

影响这种改变的实际机制是非常复杂的，但是基本思想是非常简单的。图20-25是实际中的例子。来自于Pixar公司动画[PIXA86]的“Luxo Jr.”模型被假设为从桌子上的一个位置跳到另一个位置。Luxo由一个头、三个线段和底座组成。每个连接点都是没有摩擦力的，并且有一块肌肉去决定连接的角度。计算的初始路径Luxo从在桌子上的一个点到在桌子上的一个远点的运动。这条路径逐渐地修改为由一个初始身体压缩、一个伸展和跳跃、一个向上向前拉底座和一个为防止倾到的连续起飞等组成。这个运动在几个方面是不平常的：它完全是合成的（即下蹲和伸展没有被编入程序），并且同时它显示了传统的漫画家不寻常的直觉，他们为物体划出一个相似的运动，而且隐含解决了一个特别复杂的变分问题。Witkin和Kass评论说这个结论具有一般性：我们可以创建要满足的许多约束，并且仍可使用这个技术找到结果。然而，不必说，这种方法在计算上是极费时的。将来研究的一个方向是能让一个建模者去建议路径修改的方向以加速寻找答案的过程。

20.10 小结

越来越多的学科对在计算机图形学中建立复杂现象模型有所帮助，我们现在看到的产生的图像的丰富性是由于各种各样的技术一起使用来产生这些现象的结果。成功的模型仍然有两种形式：它们是那些基于被建模物体的基本结构和物理特性的模型和那些基于使事物看起来很好的模型。第二种形式经常比第一种形式常用。我们预期在将来看到更广泛种类的物体被建模。模拟人的形态和运动以及动物的表现和行为，是特别有意义的挑战。即使在本章所讨论的领域

中，也有进一步研究的丰富空间。植物和树的模拟可以扩展到小区域的生态的模拟，包括不同植物形态之间对不同资源的竞争。波浪的模拟可以扩展为更精确地包括风的影响和碎浪的形式。像布料、泥和液体这种相互连接结构的建模可以扩展为包括破裂的模型、混合介质（泥浆和液体的运动是如何的不同？）和状态的改变（例如，融化的冰）的模拟。我们期待着看到这些新模型及它们的后继者，并且热切地等待这样的一天到来，即计算机合成的场景习惯地当成摄影图像。尽管这种欺骗眼睛的能力可能不总是建模的最终目标，但它对计算图形学来说是很好的能力测度：如果我们能模拟现实，我们就能够模拟任何东西。

习题

20.1 使用二项式理论证明在Sederberg-Parry形变技术用到的Bernstein多项式$Q_{n,i}(t)$满足$\sum_{i=0}^{n} Q_{n,i}(t)=1$，这就是说：

$$\sum_{i=0}^{n}\binom{n}{i}a^i b^{n-i}=(a+b)^n$$

20.2 在你的计算机上实现Perlin纹理模型。你能精确地调整这个模型使之能计算在特定面的点上的纹理（例如球面），以便能生成一个实时的纹理编辑器吗？在以实时方式生成多频率噪声时的困难是什么？试使用在文中提到的凹凸纹理映射函数dNoise()计算法向量；也就是说，以*newNormal* = *oldNormal* + dNoise (*currentPoint*)形式的规则调整法向量，令$f(x)$是这样一个函数，当$x<a$时为0和$x>b$时为1，其中a和b是满足$a<b$的正值，假设你做纹理映射时使用如下规则：*newNormal* = *oldNormal* + f(Noise(*currentPoint*))*dNoise (*currentPoint*)，那么对于不同的a和b的值你希望看到什么样的结果？试使用这个方法生成Perlin的灰泥纹理。 1054

20.3 Perlin使用三次插值来产生在计算噪声中用到相关系数的值。你能够想到使用查找表来加速这个过程的一种简单方法吗？你能够想到生成波段限制的噪声的更快的方法吗？

20.4 实现烟火的粒子系统，方法是：第一个阶段的粒子沿着抛物线轨道，并且随后的粒子沿着一个较小抛物线轨道。你能够将这个系统和软物体模型结合起来生成水的爆炸溅射模型吗？

20.5 完成Gardner的云模型，并且调整参数以生成一个更好看的积云。

20.6 思考一下你怎样建立一个一维、二维或者三维现象的模型。对于一维物体，我们是指一个物体的位置可以由单个参数来确定，例如平面上的一个曲线（单个参数是从曲线开始的距离），二维物体是指其位置是由两个参数确定的，例如一个球形面是二维物体，因为位置可以用经度和纬度来确定。注意到像头发这样的一维物体是很难被绘制的，因为一个像素宽的线对头发的图像来说已经是太宽了。解决这个问题需要理解第14章和第17章中的滤波理论。可以考虑花瓣（你能想到一个方法制作玫瑰开花的动画吗？）、有棱纹的表面（例如伞，骨骼上的皮肤）、带子（仅给你几个在带子上的点，并允许力学决定余下的事，你能模拟一个带子的形状吗？）等有趣的二维物体。你也可以考虑海绵（这是一个真实的分形吗？）、透明的玻璃和珠母层等三维物体。 1055

第21章 动 画

动画，顾名思义，就是使静止的画面活动起来。人们通常认为动画与运动同义，但实际上动画包括一切具有视觉效果的变化。也就是说，它包括物体的位置（*运动动力学*）、形状、颜色、透明度、结构和纹理（*更新动力学*）等随时间的变化，以及光照、相机的位置、方向或焦距乃至绘制方法的变化。

动画在娱乐业得到了广泛的运用，也被使用于教育、工业应用（诸如控制系统、智能终端和飞行模拟器）以及科学研究。计算机图形（特别是动画）在科学计算这方面的应用已逐渐归结到*科学计算可视化*这一主题之下。不过，可视化不仅仅是将图形学应用于自然科学和工程学中，它还涉及其他学科，譬如信号处理、计算几何和数据库理论。通常，科学计算可视化中的动画是对自然现象的模拟。这些模拟的结果可能是表示二维或三维数据的大型数据集（例如，对流体的模拟）；这些数据首先被转换为图像，而后组成动画。另一方面，模拟的结果也可能生成实际物体的位置和运动方向，而后，它们必须以某种特定方式进行绘制来产生动画。比如说，对化学过程的模拟就属于这种情形，模拟会生成各式各样的、相互作用的原子的位置和运动方向，但动画所显示的却可能是表示分子的一个个球-棍结构，或是表示原子的相互遮盖的光滑的球体。某些情况下，模拟程序会包含一种嵌入式的动画脚本语言，这时，模拟和动画的过程是同时进行的。 1057

如果因每秒播放的帧数过多而造成动画中视角变化太快，就会发生*时域走样*。这方面的例子有：车轮有时看起来会向后转，物体在刹那间穿过很大视角范围时所产生的跳跃运动。录像的播放速率是每秒30帧，电影通常是每秒24帧，两者都能为许多应用提供合适的效果。当然，要利用这样的速率，就必须为录像或电影的每一帧创建一个新图像。如果动画制作者不这么做，而仅为每两帧录像记录一幅图像，则所得到的动画有效帧速率仅为每秒15帧，并且看起来有跳动。[⊖]

这里所描述的某些动画技术，已部分或完全由硬件实现。对于建造飞行模拟器或其他实时控制系统来说，具有基本的实时动画功能的体系结构是基本的要求。在第18章中，已对某些这类体系结构进行了讨论。

传统的动画（也就是非计算机动画）本身就是一门学科，我们不准备全面探讨它。在这里，我们着重介绍计算机动画的基本概念和一些经典的系统。我们将首先探讨传统动画及现有的计算机辅助动画设计的方法。然后，我们的重点将转向计算机动画。因为这些动画中大部分是三维的，所以传统的二维角色动画所采用的许多技术不能被直接沿用。而且，当动画制作者不是直接用手来绘制动画时，对动画流程的控制也更加困难：描述某件事该怎么做比直接动手做这件事要难得多。因此，在讲解了各种计算机动画语言后，我们将深入学习几种动画控制技术。在本章末尾，我们再讲解动画的一些基本规则和动画所特有的一些问题。

21.1 传统动画和计算机辅助动画

21.1.1 传统动画

传统动画的制作流程是很固定的：第一步是写动画剧本（也可能仅打打腹稿），下一步是建

⊖ 但是，这使动画制作者仅需要生成一半的帧数。在某些应用中，在时间节省和动画质量间进行权衡时，倾向于前者。

立故事板。所谓故事板，是一种大纲形式的动画——能体现动画结构和构思的、高度简化的草图序列。第三步，录制声道（假如有的话）；接着，制作详细布局（为动画中每个场景画一张图），同时读入声道——这意味着，各种声音是以其出现时刻为序依次录制的。这样，上述的详细布局和声道之间就相互关联了⊖。再下一步，画出动画中特定的*关键帧*——在这些帧中，动画中的运动物体处于指定位置，由此可以推算出物体在中间各帧中的位置。然后，在关键帧之间插入这些中间帧（这个过程被称为*插值*——inbetweening），这样，初步的胶片就完成了（上述整个过程被
1058 称为*素描*（pencil test））。而后，用彩笔手工复制或直接拍照，将素描的各帧转换为*醋酸盐胶片*。在多层动画中，使用了多层醋酸盐胶片，一部分作为背景的胶片保持不变（但也可能要做一定的平移），而另一些前景对象则随时间而变化。对胶片进行着色后，将它们按正确的次序进行组合，就可以正式录制了。制作动画的人员有着十分明确的分工：一些人设计剧本，一些人画关键帧，还有些人完全是插值帧制作员，其余的人仅仅是为最终的胶片着色。因为关键帧和插值帧的使用，这种动画被称为*关键帧动画*。这个名字也适用于模拟上述过程的基于计算机的系统。

在[CATM78a]中通过介绍动画的故事板、*流程表*（route sheet）和*帧列表*（exposure sheet）等描述了一部动画的组织过程，其中流程表描述了每一个场景以及场景制作人员对场景制作过程的各方面所负的责任。帧列表的每行信息对应动画中的一帧，它描述了对话内容、帧中各角色的顺序、背景的选择及相机在帧中的位置。这种组织细节的层次对于制作一部连贯的动画来说是至关重要的。要更进一步了解传统动画的有关知识，请阅读[LAYB79；HALA68；HALA73]。

制作动画的整个过程被认为是有序但经常是有重复的（特别是用计算机设计时）：现有的音响效果的限制可能会导致对故事板稍做改动，动画的最终效果也许需要对某些步骤进一步扩充，这反过来又要求新的声道片段，等等。

21.1.2 计算机辅助动画

传统动画的许多阶段用计算机实现似乎很理想，特别是插值阶段和着色阶段，它们可使用19.5.2节中讲到的种子填充技术在计算机上加以实现。但是，在用计算机进行处理之前，图像必须被数字化，其途径有：使用光学扫描，在数据输入板上对图像进行跟踪，或一开始就用绘图软件来生成原始图像。所得到的数字图像也许还得进行后期处理（例如滤波）来消除输入过程（特别是光学扫描）所产生的干扰，并对图像的轮廓进行适当的平滑。在合成阶段，即将前景和背景对象合成起来产生独立的帧以得到最终动画的阶段，则可用17.6节讲到的图像合成技术实现。

将动画的若干帧缩小并降低分辨率后存放在一个以矩形图像为元素的数组中，使用某些帧缓存提供的移动-缩放功能，就可得到与素描等价的效果。帧缓存能存储上述图像的一个特定区域（此区域包含一个低分辨率的帧），并将这个区域移动到屏幕中央（移动，panning），而
1059 后将其放大至全屏（放缩，zooming）⊜。上述过程可对存储在一幅图像中的某几帧动画反复进行；只要速度足够快，就能产生连续的效果。因为动画的每一帧仅是整幅图像的一小部分（通常是1/25或1/36），而后要放大至全屏，所以上述过程最终降低了显示器的分辨率。但是，这些低分辨率的图像序列有助于得到动画的概貌，因此可作为某种形式的素描。

21.1.3 插值

中间插值的过程显然也可在计算机上实现，但这会碰到不少问题。虽然插值帧制作人员能

⊖ 这里所讲述的顺序来自传统的动画制作室。传统动画中，声道可能是最后录制的；而在计算机动画中，这个过程可能包括若干次反复。

⊜ 移动和缩放实际上是通过改变帧缓存寄存器的值来控制的。一组寄存器决定帧缓存中哪个像素对应屏幕的左上角，另一组寄存器则决定像素复制的方式——每个像素在水平和垂直方向各被复制多少次。通过调整这些寄存器的值，用户可以依次显示每一帧，并且每帧都是用逐点复制的方式充满全屏的。

看到被插值物体周围的环境（一个小球是在下落还是在滚动？），但是计算机动画系统通常只能获得物体的起止位置。这种情况下最简单的插值方式是线性插值：给出某个属性（位置，颜色或尺寸）在起始帧和结束帧中的值v_s和v_e，则中间帧中相应的值v_t是$v_t = (1-t)v_s + tv_e$，当t的取值是从0到1时，v_t的值由v_s到v_e连续变化。虽然线性插值（有时也叫lerping——Linear intERPolation）适用于某些情形，但也有许多限制。例如，一个小球被抛向空中，仅使用图21-1a中所示的三个关键帧序列。此时采用线性插值法计算小球的中间位置，所得到的小球运动轨迹如图21-1b所示。此结果完全不合实际。其中最成问题的是轨迹顶点的尖角：虽然线性插值产生了连续的运动，但其导数却不连续，所以用线性插值法求物体的中间位置时，可能会产生突变。即使三个关键帧中小球的位置在同一直线上，若第二和第三关键帧中小球间的距离大于第一和第二关键帧中的话，线性插值法也会导致在第二关键帧处小球的速度不连续。因此，线性插值使得对时间的导数和对空间的一样，产生不连续（时间上的不连续性可用第11章讲到的参数连续性进行度量）。

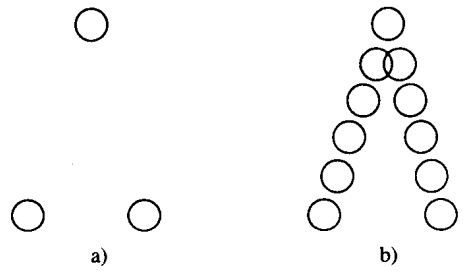


图21-1　对小球的运动进行线性插值得到不真实的结果。a)小球的三个关键帧位置，b)所产生的插值位置

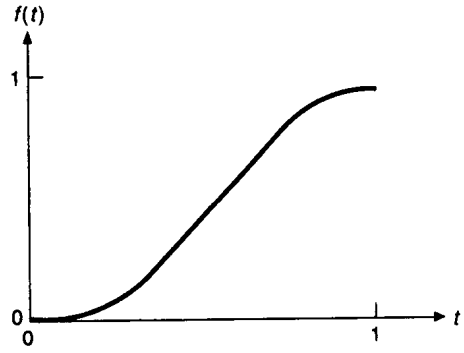


图21-2　端点处斜率为0，中段斜率恒定的函数$f(t)$的图

因为线性插值存在这些缺陷，所以人们用样条代替它以使关键帧间的插值更平滑。使用样条可使任何参数随时间平滑变化。样条不一定是多项式[⊖]。比如，为了使变化开始和结束时显得平滑（称为*渐入*和*渐出*），而且变化的速率相当恒定，我们可用图21-2中所示的$f(t)$之类的函数作为样条。令$v_t = (1-f(t))v_s + f(t)v_e$，就可以进行插值了。因为当$t=0$和$t=1$时，$f(t)$的斜率都是0，所以$v$在开始和结束时变化都十分平滑。又因为$f$的斜率在中段是恒定的，所以$v$的变化率在中间阶段也是不变的。

使用样条可以使单个点（或单个物体）沿空间和时间平滑运动，但这远未能解决插值帧制作中的问题。中间帧插值制作也包括在中间帧中对物体的形状进行插值。当然，我们可为动画中每一帧的每一点的运动定义一条样条路径，但是，只有当运动在空间和时间上都只有少量控制点时，样条插值的结果才最平滑。因此，只在某些时刻指定某些点的位置，并用某种方式把样条插值拓展到中间点和中间各时刻，才适合用样条插值。至少有一种特殊情况值得注意：用折线画出的图形，可通过对其各顶点位置从起始帧到结束帧插值来实现其在关键帧间的插值。只要关键帧间差别不是太大，结果将会是合乎实际的（例外情形请见习题21.1）。

现在，对物体形状进行插值已有了好几种解决方案。Burtnyk和Wein [BURT76]选择用一条多边形弧来描述二维对象或其一部分的大体形状，再加这一圆弧的邻域，得到了一个用于制作动画的骨架（skeleton），见图21-3。人体表示在基于这个骨架的坐标系中。他们还定义了弧的

⊖ 这是对第11章中样条概念的扩展，当时样条被定义为分段的三次曲线。这里我们将这个术语用于更广泛意义上的用来逼近一组控制点的任意曲线。

粗细，各顶点在依次的各关键帧中的位置。而后，在以变形后的弧为参照物的坐标系中重画形状。中间帧插值制作通过对关键帧间骨架特征进行插值来实现。（利用第20章讲到的三元Bernstein多项式形变或树型B样条，类似技术可推广到三维动画中。）

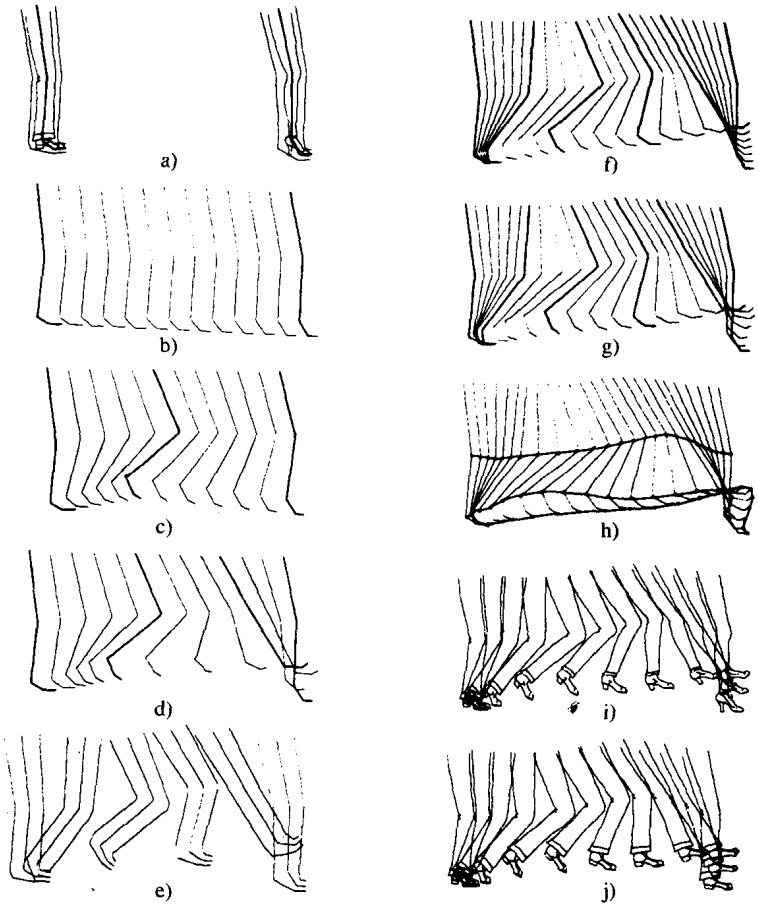


图21-3 用骨架的邻域来定义内插的形状。（由加拿大国家研究委员会M.Wein和N. Burtnyk许可。）

Reeves [REEV81]设计了另一种方案：用手工绘出的路径来决定相邻关键帧间动画对象的特定顶点的轨迹（由动画制作人员标出时间等分点）。由两个关键帧间的两条上述的运动点路径和两关键帧中动画对象的一段弧所界定的区域，决定了动画的片。动画对象的这段弧的插值是通过计算它在该区域内的各中间位置得到的。中间位置的确定以使运动尽可能平滑为准。

上述两种技术都是针对线条图设计的，但是对三维对象的插值也会有同样的问题。两者之间最大的区别是，大多数计算机动画倾向于对三维物体精细建模，而不是仅绘出轮廓。因此，在插值过程中，建模和位置信息是可用的，而且，动画制作者通常无须指出物体上的点在不同关键帧中的对应关系。尽管如此，关键帧间的插值仍是一个难题。

1060
~
1062

我们暂时只考虑对刚体的位置和方向进行插值。用二维动画中使用的技术即可实现对位置的插值：指定特定关键帧中刚体中心的位置，则所有中间位置可沿某条样条路径插值得出。而且也可事先指定沿样条路径的运动速率（例如，标出轨迹上的时间等分点或定义沿插值路径的运动速率为时间的函数）。许多不同的动画系统都实现了上述技术，在21.2.3节中将讨论其中一些系统。

对刚体的方向进行插值更为困难。事实上，甚至对方向的定义都不简单。如果我们用绕三

个基本坐标轴旋转的角度（称为欧拉角）大小来定义方向，则定义的顺序将十分重要。比如说，一本书的书脊开始时朝左，当它先绕x轴旋转90°，再绕y轴旋转 - 90°后，它的书脊就朝向你了；但若按相反的次序旋转，它的书脊将朝下。这样的插值结果难以预测，因为对欧拉角的插值会导致对旋转的不正常插值：先绕z轴旋转90°，再绕y轴旋转90°，其效果等价于绕向量（1, 1, 1）旋转120°。但先绕z轴旋转30°，再绕y轴旋转30°，其效果并非绕向量（1, 1, 1）旋转40°——而近似于绕向量（1, 0.3, 1）旋转42°！

所有可能旋转的集合恰好对应于一个相应的代数结构，即*四元数*（quaternions）[HAMI53]。旋转正好就是*单位四元数*，其符号形式为$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$，其中$a$, b, c, d都是实数，且满足等式$a^2 + b^2 + c^2 + d^2 = 1$；四元数的乘法满足分配律和规则$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$, $\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$, $\mathbf{jk} = \mathbf{i} = -\mathbf{kj}$以及$\mathbf{kj} = \mathbf{j} = -\mathbf{ik}$。以单位向量$[b\ c\ d]^T$为轴旋转角度为$\phi$的旋转对应于四元数$\cos\phi/2 + b\ \sin\phi/2\ \mathbf{i} + c\ \sin\phi/2\ \mathbf{j} + d\ \sin\phi/2\ \mathbf{k}$。有了这种对应关系，连续的旋转操作就相当于相应的四元数的乘积。它们之间的逆对应关系在习题21.7描述。

单位四元数满足条件$a^2 + b^2 + c^2 + d^2 = 1$，它们可看成是四维单位球面上的点。要在两个四元数间插值，我们可以简单地沿它们之间的球面最短路径（一条*大圆弧*）进行。这种*球面线性插值*（记为slerp）是线性插值的自然推广。Shoemake [SHOE85]提出对图形插值时使用四元数，并给出了样条插值的四元数推广形式。

四元数有紧凑性和简单性等明显的优点，但同时也带来新的问题，其中有三个值得注意。第一，物体的每一个指向实际上可用两个四元数来表示，因为绕$\mathbf{v}$轴旋转角度ϕ等价于绕 - $\mathbf{v}$轴旋转 - ϕ；它们所对应的四元数是四维球面上相对的两个点。因此，从一个指向转到另一个指向，我们只需要在一个四元数和另一指向所对应的两个相对点中任选其一的四元数间进行插值；通常我们选取其对应的大圆弧较短的那个四元数。第二，指向和旋转，严格说来是两个不同的概念：旋转360°和旋转0°在动画中的表现是完全不一样的，但两者均可用四元数（$1 + 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$）来表示。因此，用四元数定义多重旋转需要许多中间控制点。

第三个问题是四元数使旋转具有*各向同性*（isotropic）的性质——插值仅依赖于起始的和结束的旋转，而与其他毫不相干。这对于插值滚动物体的位置十分理想，但不适合对场景中照相机的指向 1063
进行插值：人们很喜欢竖直放置照相机（即摄影平面的水平轴位于（x, z）平面上），而对倾斜放置的照相机很不适应。四元数不能对此进行选择，因而不能用于对照相机的插值。由于缺乏对复杂的照相机运动进行插值的适当方法，导致许多计算机动画中照相机是静止的或只能进行很有限的运动。

21.1.4　简单的动画效果

在这一节里，我们介绍几个能即时生成简单计算机动画的小窍门。它们属于最早提出的动画技术，因此是面向硬件的。

在4.4.1节中，我们介绍了帧缓存中颜色查找表（Color Look-Up Table，luts）的使用及双缓冲的过程；还在17.6节中介绍过通过对查找表的操作进行图像合成。回想一下，通过操作查找表可以生成查找表动画。最简单的方法是在查找表中对颜色值进行循环（用第$(i - 1) \bmod n$号颜色值取代第i号颜色值，其中n为表中总颜色数），这样就可改变图像中不同色块的颜色。图21-4描绘了一个水源、一个蓄水池和连接两者之间的一根管子。图中每一部分都用它的查找表索引号标出。查找表显示在右侧。从1到5对颜色值进行循环，就可生成一个液体流经管道的动画。

使用这种查找表动画比每帧送一幅新图像到帧缓存要快得多。假设帧缓存中有一幅640 × 512的图像，其中每个像素的颜色用8位表示，则它所包含的信息是320 KB。每1/30秒将这样的一幅图像送到帧缓存中所需带宽大于9 MB/s，这超出了绝大多数小型计算机的负载能力。然而，

给查找表传一个新值却很迅速，因为查找表通常只需几百到几千个字节。

1064 查找表动画一般看起来有闪烁，因为色彩变化有跳跃。使用如下方法某种程度上可减轻这种闪烁：选出将要改变的颜色，在某几个连续帧中使其查找表项值由原来的颜色值向新的颜色值渐变，然后，类似地，在下一个查找表项值淡入时使原来的淡出。上述及其他一些技巧的细节由Shoup给出[SHOU79]。

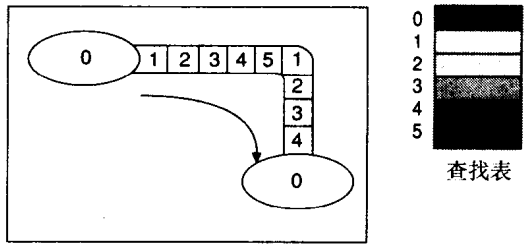


图21-4 通过对查找表项值进行循环，可得到液体流经管道的效果

查找表动画可与先前讲过的移动-放缩动画技术结合起来，以便用更少的颜色生成更长的移动-放缩动画。例如，为了用有八个面的帧缓存来制作一段很长的两色移动-放缩动画，我们可以生成一个两百帧的动画，每帧的大小为全屏的1/25。第1帧到第25帧放在一幅图中，准备用于制作移动—放缩动画。对第26帧到第50帧进行同样的处理，如此反复直到第176帧至第200帧，共得到8幅图。逐像素地将它们合成为一幅包含8位深度的图像。而后，这幅图像被装入帧缓存。我们将所有的查找表项值置为黑色，除了项00000001置为白色。接下来，我们就可播放一个25帧的移动-放缩动画并看到整个动画的前25幅图。然后，把项00000001置为黑色，项00000010置为白色，播放另外25帧的移动-放缩动画，则可看到整个动画中的第二个25幅图。以这种方式继续，可看到全部200帧动画。通过为每幅图像申请多个面，我们可以制作色数更多的、较短的移动-放缩动画。

最后，我们来看看被称为精灵的基于硬件的动画技术。所谓精灵是帧缓存中的一小矩形区域，它在显示时与帧缓存的其余部分混合使用。精灵在任何时刻的位置都由帧缓存的某些寄存器值决定，因此，改变这些寄存器值就能移动精灵。在帧缓存的任一像素处，精灵可覆盖其中相应的值或与其混合。我们可用精灵在帧缓存中实现光标，也可通过使精灵在背景图像上四处移动来生成动画。有些帧缓存的设计允许有若干不同优先级的精灵，这可以使某个精灵位于其他精灵之上。

视频游戏是精灵最常见的应用之一，游戏中的动画几乎全是精灵在固定背景上移动。因为每个精灵的位置和尺寸都保存在寄存器中，所以很容易检测到精灵间的碰撞，这进一步增加了精灵在这方面的使用。

21.2 计算机动画语言

现在有许多用于描述动画的计算机语言，而且新的还在不断开发中。总的来说，它们分为三大类：线性表表示法、嵌入了动画指令的通用计算机语言和图形语言。下面，我们将详细介绍每种语言，并给出相应的示例。许多计算机动画语言是和建模语言混合在一起的，因此，对动画中的物体的描述和这些物体的动画是同时完成的。

21.2.1 线性表表示法

在如[CATM72]中所描述的动画的线性表表示法中，动画中的每个事件用一个起始和结束帧序号和一个将要发生的动作来表示。其中的动作通常带有参数，于是下面的语句

```
42, 53, BROTATE "PALM", 1, 30
```

1065 的含义是："在帧42到帧53之间，将称为PALM的物体绕轴1旋转30°，每帧的旋转角度由表B决

定。”也就是说，动作被给定了使用的插值方法（这儿是一张数值表）和对象所做的动作等等。因为这些语句描述了单独的动作及与它们相关的帧数，所以在大部分地方，它们之间的顺序是没有关系的。但是，如果两个动作同时作用于同一物体，它们的顺序就可能有关系：先绕x轴旋转90°再绕y轴旋转90°和先绕y轴旋转90°再绕x轴旋转90°是不同的。

目前已开发了许多别的线性表表示法，而其中许多表示法都是上述的基本线性表思想的超集。比如，Scefo(SCEne FOrmat) [STRA88]中包括了线性表表示法的一些方面，但也包括了组的概念和对象层次结构，同时，支持对变动（称为动作）的抽象和一些高级编程语言的构造（变量、控制流、表达式求值）等，将其与简单的线性表区分开来。Scefo还支持一个与许多计算机动画语言不同的动画模型，即它是与绘制程序无关的。一个Scefo脚本只描述一个动画，其中任一独立对象都可用任意的绘制程序来绘制，而且很容易在以Scefo为核心的动画软件包中加入新的绘制程序。

21.2.2 通用计算机语言

另一种描述动画的方式是在某种通用编程语言中嵌入动画功能[REYN82；SYMB85；MAGN85]。其中变量可作为实际生成动画的任何例程的参数，所以高级语言事实上都可用来编制一些模拟程序，而后附带地生成动画。这些语言非常有潜力（事实上，它们可做到线性表表示法可做到的一切），但它们大部分都要求用户在这方面有相当的编程经验。

这种系统可利用相关语言的结构来生成能实现复杂效果的简单例程。当然，这些结构有时是隐含的。ASAS [REYN82]就是这种语言的一个实例。它建立在LISP的基础之上，其基本实体包括向量、颜色、多边形、多面体（多边形的组合）、组（对象组合）、观察点、子世界以及灯光。观察点由一个物体或相机的坐标和方向构成（即它对应于PHIGS中对象的累加变换矩阵）。子世界是与一个观察点相关的实体；观察点可用于处理与其相关的子世界中实体和动画中其他对象的关系。

ASAS还提供了大量的作用于对象的几何变换函数；它们把对象作为一个参数，返回值是这个对象经变换后的副本。这些变换函数包括up、down、left、right、zoom-in、zoom-out、forward、backward。下面给出ASAS的一个程序片段，它描述了一个动画序列，其中一个叫my-cube的对象在相机摇动时不断旋转。跟在分号后面的是注释。在每一帧中对这个程序段进行求值以生成整个动画序列。

```
(grasp my-cube)                ;my-cube成为当前对象
(cw 0.05)                      ;将它顺时针旋转一点
(grasp camera)                 ;camera成为当前对象
(right panning-speed)          ;将它右移
```

1066

ASAS相对于线性表表示法的优点在于它具有在语言中生成过程化对象和动画的能力。这种能力的获得是以提高对动画制作者编程能力的要求为代价的，即动画制作者必须是一位熟练的编程人员。Scefo介于两者之间，它提供某些控制流结构，并与用高级语言编写的程序有动态绑定的能力，对非程序员用户来说，其学习和使用也足够简单。

21.2.3 图形语言

对于我们所介绍的文本语言，有一个问题是，仅仅看脚本，动画制作者很难知道动画中究竟会发生什么。当然，这并不令人惊讶，因为脚本是一段程序，只要程序使用的语言允许使用高级函数，就能以简洁的形式对复杂事件进行编码。如果有动画语言的实时预览器的话，这并不成问题；但是，实时动画生成依然是大多数硬件所不支持的。

图形语言以一种更直观的方式描述动画。这类语言用于表示、编辑和理解动画中同步发生的

变化。这种语言的主要想法是用可视化的实例来代替文本语言：动画制作者不是详细写出动作的描述，而是为动作提供一张图。这个领域的一些早期工作是由Baecker [BAEC69]完成的，他在GENESYS动画系统中提出了P曲线表示法的概念。P曲线是场景中物体或组合物体运动（或任何其他属性）的参数表示。动画制作者通过图形形式定义一个物体的坐标为时间的函数（就和样条一样，用函数$X(t),Y(t),Z(t)$来定义曲线上某点的三维坐标，其中曲线是某个独立变量的函数）来描述物体的运动路径。图21-5a中显示了平面上的一条运动路径；图21-5b显示了作为时间函数的路径的x、y分量。请注意，图21-5b中的曲线惟一确定图21-5a中的那条曲线，但反过来则不成立：沿图21-5a中曲线运动的速度是可变的。通过在图21-5a中路径上标出时间等分点，我们可以得到路径对时间的依赖关系，如图21-5c所示，这就是Baecker所谓的P曲线。注意图21-5c可这样得到：如图21-5d所示，在对应坐标系中画出作为t的函数的x和y分量的图像，将图21-5b中两个坐标系旋转使得两坐标系的t轴相互垂直，然后用线段连结相应的时间等分点。因此，对参数曲线中各分量的修改会导致P曲线的变动，而修改P曲线中散列标记点的位置也会引起各分量的变动。

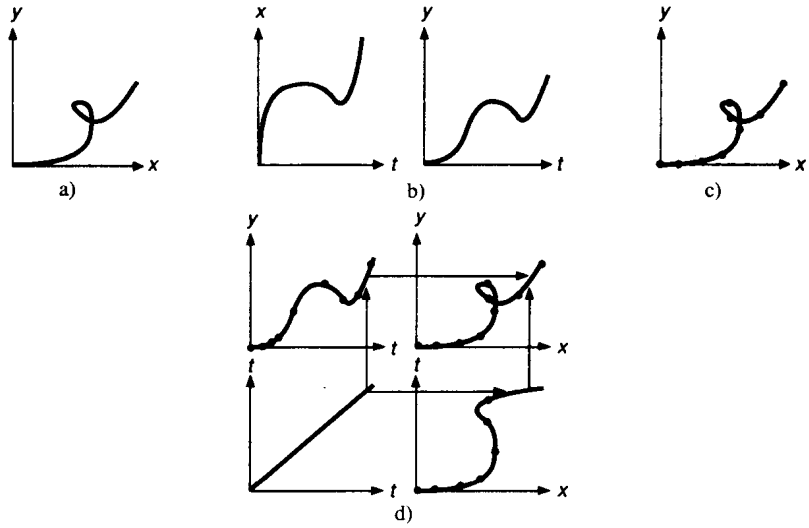


图21-5 a)平面上的一条参数路径，b)它的x和y分量都是时间的函数，c)原曲线的标记点表示等分的时间段，d)由分量函数构造P曲线

图表形式的动画语言DIAL [FEIN82b]包含线性表表示法的某些特性，但它用一系列平行的标记行来显示动画中的事件序列：用竖条表示动作的开始，一些短线表示动作发生的时间段。动作在DIAL脚本中用“`%t1 translate"block"1.0 7.0 15.3`”形式的语句加以定义（上句的意思是动作t1定义为：称为block的物体沿向量（1.0, 7.0, 15.3）平移），接下来就可以定义动作的应用。DIAL所执行的这些特殊指令由运行时给出的用户定义的后台程序加以实现。DIAL本身对动画一无所知：它仅提供指令执行序列的描述。下面是一个典型的DIAL脚本（以空格开始的行是注释）：

```
    从文件中读入一个对象,并将其命名为block
! getsurf "block.d" 5 5 "block"
    在xy平面上定义一个窗口
! window -20 20 -20 20
    定义两个动作，(1) 平移，
% t1 translate "block" 10 0 0
    (2) 在xy平面上旋转360度
% r1 rotate "block" 0 1 360
```

```
        以下描述一次平移，一次转动和再次的平移:
t1      |---------        |--------
r1                |--------

r1      |-------------------------
```

标以“t1”的行表示动作t1在帧1到帧10间发生（每帧平移一个单位，因为线性插值是默认动作），而后在第17帧到第25帧时重复此动作。在帧11处，block停止平移并在接下来的6帧中每帧旋转40°（第一个标有“r1”的行就是表示这个含义），而后3帧既平移又旋转，接着又仅平移。

1067
-
1068

对于较长的动画，每个时间标记可表示多个帧，因此几秒钟的动画可很容易地加以定义而不用表示出每一帧。长动画的描述是这样的：在新的一行开始一个新的时间标记序列，其后跟随一个空行。（上面第二个标有“r1”的行就是一个例子：它表示在第28帧到第50帧间要做360°旋转。）这种形式就像交响乐队指挥的总谱：每个动作对应于乐谱的一条指令，每组短线对应于总谱中的一个五线谱。DIAL和许多线性表表示法都具有完全由ASCII文本定义的优点，这使它们更易于移植（虽然必须为每种新机器写一个这种语言的后台程序）。

S-Dynamics系统[SYMB85]在可视化图示方面更进了一步，并结合了与P曲线类似的动作的参数化描述。为了做到这一点，S-Dynamics使用了功能强大的图形工作站。图21-6显示了一个S-Dynamics的窗口。和DIAL中一样，时间轴沿窗口的水平方向。动作有效时间由表示动作的区域宽度表示。每个动作（或用S-Dynamics的术语*序列*）都能用一个指示动作的时间范围的盒子来表示，而盒子可以是“开着的”；也就是说，可以显示更多的内部细节。序列可由若干串行的或并行的动作组成，其中每个动作都可能被打开以显示更多细节，包括一个表示时间与动作的参数相关的图。

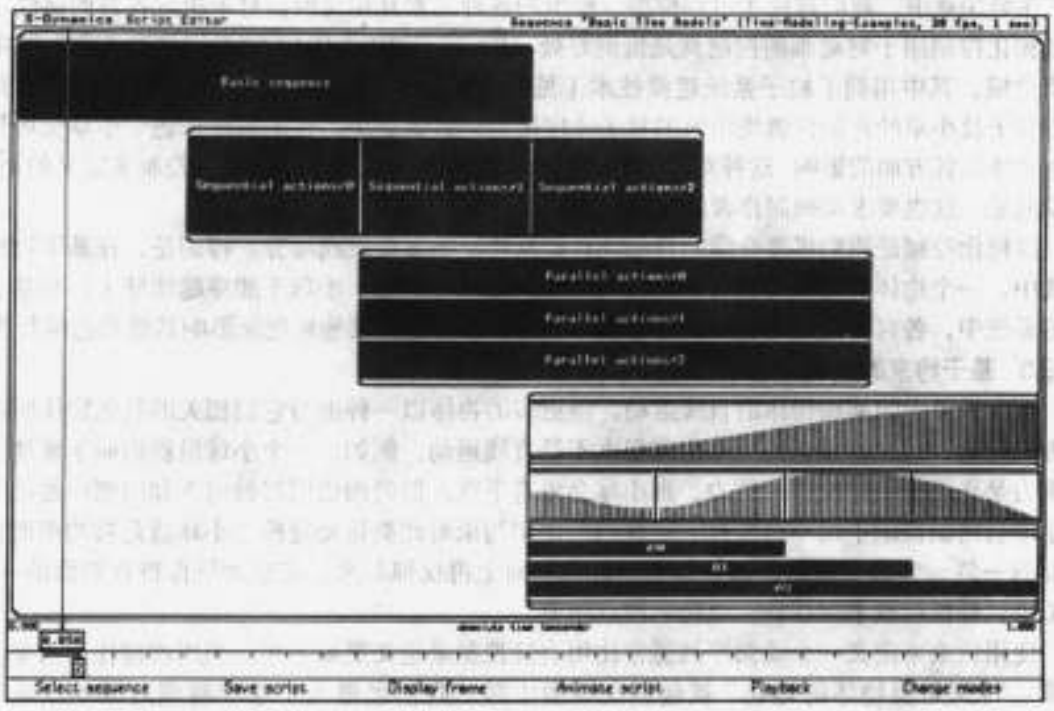


图21-6　一个S-Dynamic窗口。（由Symbolics Graphics Division许可。首次公开这一图像的软件和SIGGRAPH论文由Craig Reynolds撰写。）

21.3 动画控制方法

动画控制某种程度上与描述动画所用的语言无关——绝大多数控制机制可用不同类型的语言实现。动画控制机制涉及的范围从完全显式的控制技术到由基于知识系统所提供的高度自动化的控制技术，前者由动画制作者使用平移、旋转及其他对物体的位置或属性进行改变的操作来显式地描述场景中每个对象的位置和属性；后者对动画进行高度抽象的描述（如“使人走出房间”）并产生所要的显式控制，这种控制对完成动画制作中的变化是必要的。在本节中，我们将剖析这些技术，并通过相应的例子比较各种技术的优缺点。

21.3.1 完全显式的控制

显式控制（explicit control）是动画控制中最简单的类型。这种情况下，动画制作者给出动画中所发生的一切事件的描述，要么定义一些简单的变化，如缩放、平移和旋转，要么给出关键帧信息和关键帧间使用的插值方式。这种插值可显式给出，也可（在交互式系统中）用鼠标、游戏杆、数据手套或其他输入设备手工控制。

BBOP系统[STER83]提供了这种交互式的控制方式。它的基本对象模型由分层链接的多面体对象组成（即棒状体间有关节相连），动画制作者可在每个关节点处用游戏杆或其他交互设备来控制相连棒状体之间的变换矩阵。这样的交互定义了每个关键帧处的变换，而交互程序则定义了关键帧间的插值方式。注意，在这样的系统中，两个关键帧间定义的一系列动作可能很难进行修改；而且对某个动作进行延长可能需要缩短其相邻的动作来保持动画的连贯性。比如，考虑下面的动画：一个小球向前滚动，撞上另一小球并将其弹开。如果第一个小球滚动得更慢一些，第二个动作（即第二个小球弹开）的引发就会被延迟。

21.3.2 过程化控制

在第20章中，我们讨论了过程模型，模型的各种元素互相交换信息来决定各自的属性。这种过程化控制用于对动画的控制真是恰到好处。Reeves和Blau [REEV85]用这种方法对风中的小草建模，其中用到了粒子系统建模技术（见20.5节）。而后，在动画生成过程中，涉及到的风的粒子及小草叶片的位置均由风的粒子密度决定。这就是说，粒子系统描述了小草受场景中其他物体的各方面的影响。这种对象间的过程化交互可用于生成难以用显式控制来定义的动画。不幸的是，这也要求动画制作者同时也是程序员。

1069 ~ 1070

过程化控制是我们将要介绍的其他几种控制机制的重要组成部分。特别是，在基于物理的系统中，一个物体的位置可能会影响另一物体的运动（例如，小球不能穿越墙壁）；在基于角色的系统中，各自独立的角色可以将它们的位置信息传递给其他角色来影响其他角色的行为。

21.3.3 基于约束的系统

现实世界中的某些物体沿直线运动，但更多的物体以一种由与它们相关的其他物体所决定的方式运动，且这种复杂的运动也许根本不是直线运动。例如，一个小球沿斜面向下滚动。如果重力是作用于小球的惟一的力，则小球会垂直下落。但斜面也同时被向下和向侧面推动，因此小球会沿斜面滚下而不是穿过它。我们可使用约束对此类运动建模。小球就是被约束而位于斜面的一侧。如果小球是从高处掉下，打在斜面上再反弹起来，它也始终保持在斜面的一侧。类似地，钟摆绕轴来回摆动，也是一种点约束。

使用约束来定义一个动画序列通常比用直接控制来定义更易实现。当用物理作用力来定义约束，特别是当物体的运动[⊖]被包括在模型中时，我们已进入了基于物理造型的领域（见

⊖ 这里所讲的运动是物理意义上的运动，指的是位置或动作随时间改变，而不仅仅指前几章中的“改变”。

21.3.7节）。但是，简单的基于约束的建模能产生有趣的结果。如果连杆上的约束用来定义其可能的位置，就像图21-7a所示的那样，我们就可以用简单的方法变动连杆来得到它的动画。例如，在图21-7中，动画制作者可以像图21-7中b)、c)、d)部分那样，仅转动一下驱动轮就可以生成连杆的动画。



图21-7　a)图中连杆的运动是通过旋转驱动轮来实现的。带约束的运动如图b)、c)和d)所示

Sutherland的Sketchpad系统[SUTH 63]第一个使用了这种基于约束的动画（见图21-8）。它允许用户像现在的二维绘图软件一样生成对象集合的各个部分。对象集合的各个部分（直线、圆等）可使用点约束（“这条直线可自由移动，但它的一个端点固定在这个点的位置”）、连杆约束（“这些线段必须保持首尾相连”）或角度约束（“这些直线必须保持平行”或“这些直线的夹角必须为60°”）来进行约束。这允许用户用四条线段画一个四边形，在各顶点处定义连杆约束，在某个顶点处定义点约束，并在相对的边上定义角度约束以使它们平行。这将得到具有固定顶点的平行四边形。约束可用一种渐近技术来满足，其中对象集合被移动以使之更接近于满足约束。因此，用户可见到对象集合被逐渐移动以满足约束条件[⊖]。当然，系统可能是过约束的，例如，要求直线段具有单位长度，且它的两端点与相距三个单位的两点相连。在Sketchpad中，这样的约束由一个报错函数给出，当满足约束条件时，函数返回值是0，否则为正。渐近技术尝试着使这些函数都返回0；当它失败时，许多约束条件可能没被满足。类似地，系统也可能是欠约束的，这时有多个解满足约束条件。这种情况下，渐近技术找到一个与初始设置接近的解。

1071

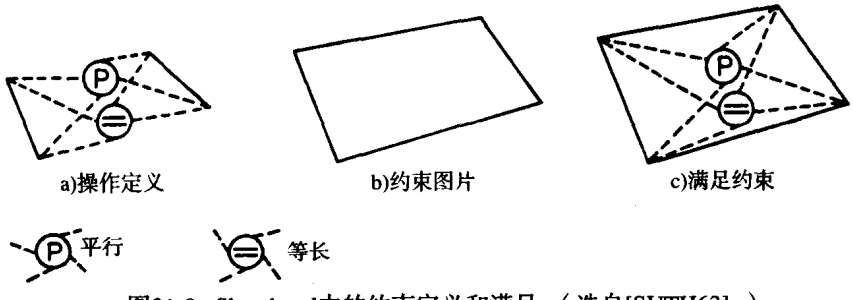


图21-8　Sketchpad中的约束定义和满足。（选自[SUTH63]。）

类似地，Borning开发的ThingLab [BORN79]是一个真正的元系统：它提供与Sketchpad相似的系统定义机制，但用户可在同一框架中定义一个集成电路建模系统。后来，这个系统进行了改进，加入了图形界面[BORN86b]。一个系统一旦被设计，就提供一个用户可建立各种实验的世界。例如，在一个用于几何建模的世界中，用户能用例子展示线约束、点约束、中点约束等，而后在这些约束之下移动对象集合。图21-9给出了一个例子，用户示例了4个有中点的线段，其中的约束是它们的端点相连，并在相邻的中点间画4条线。用户可以变动外面的四边形，同时可看到内部的四边形总保持是平行四边形。相关工作参见文献[BIER86a]。

将基于约束条件的动画系统扩展为支持层次的约束系统，以及由物理实体的动力学行为及材质的结构特征（见20.7.3节中的塑料模型）进行建模的约束，是一个活跃的研究课题。

1072

⊖ 因此，这些动画为两个目的服务：它们生成满足约束的对象集合，同时它们给出了渐近技术的可视化。

21.3.4 真实动作跟踪

动画过程中物体的运动轨迹可通过对真实运动进行跟踪来生成。跟踪的方式有许多种。传统动画使用了rotoscoping：胶片录制时部分角色由人（或动物）扮演，而后动画制作员在胶片基础上进行绘制，以增强背景效果并将动画中的人类演员用动画对等体来代替。这种技术可得到十分真实的动画。或者，对一系列胶片帧中的物体上的关键点进行数字化处理，再插值生成中间点，从而产生相似的动画。

另一种真实运动技术是将某种传感器贴在人体的关键位置，通过跟踪它们的位置，就可得到动画模型中相应关键点的位置。例如，在人身上一些关键位置贴上一些小灯，而后，从若干方向对这些小灯的位置进行记录就可得到每一时刻各关键点的三维位置。这种技术被Ginsberg和Maxwell[GINS83]用来生成一个图形化提线木偶；在房间中来回走动的演员的位置被记录下来，被处理后就得到运动的实时视频图像。演员自己可看到这一运动，并得到他或她所做运动的反馈。如果这种反馈用头盔显示器给出，同时在头盔显示器上显示预先录制的动画片段，演员就可与其他图形实体进行交互。

另一类交互机制是第8章中所介绍的数据手套，它可以测出穿戴者手的位置和方向以及每个手指关节的屈张。这种设备也可用于描述动画中的运动序列，和三维数据板十分类似。就像二维运动可由一条P曲线来描述一样，三维运动（包括方向）可由数据手套的移动来描述。

21.3.5 演员

使用*演员*是过程化控制的高级形式。动画中的一个演员就是每帧都执行的一小段程序，用于决定动画中某个物体的特征。（也就是说，动画中的演员对应于面向对象程序设计中的对象）。一个演员，在它的每帧一次的执行过程中，也许会发送消息给其他演员来控制它们的行为。因此，我们可以通过让引擎演员响应某些预先指定的规则集（以固定速度沿轨道运动）来创建一辆火车，同时向火车上的第二辆汽车发送下列消息“将你自己置于轨道上，将你的前端置于引擎的末端。”每辆车传送一条相似的消息给下一辆，于是所有的汽车会依次排在引擎后面。

这种演员的思想最初是从Smalltalk[GOLD76]和其他语言中类似的概念中派生而来，并且是21.2.2节中所讲的ASAS动画系统的核心。这一概念被进一步引伸，可包括具有各种“行为”
1073 的演员（演员可根据环境执行这些行为）。

21.3.6 运动学和动力学

运动学（kinematics）涉及点的位置和速度。例如，场景的一个运动学描述也许是“立方体在$t=0$时刻位于原点。而后，它以恒定加速度沿向量（1, 1, 5）方向运动。”相对地，*动力学*（dynamics）考虑了运动学所遵循的物理法则（关于大型物体的牛顿运动定律，关于液体的欧拉-拉格朗日方程等等）。粒子运动的加速度与作用于它的力成正比，比例常数是粒子的质量。也就是说，场景的动力学描述可能是“在$t=0$时刻，立方体位于坐标（0米，100米，0米），立方体质量为100克，作用于立方体的力是重力。”这样的动力学模拟[⊖]结果自然是立方体将下落。

运动学和动力学都是可逆的，即我们可以提出这样问题：“立方体的（恒定）初速度必须是多少才能使它在5秒后到达坐标（12, 12, 42）”或“我们必须对立方体施加什么样的力才能使它在5秒后到达坐标（12, 12, 42）”。在简单系统中，这类问题也许有惟一答案；但对更复杂的系统来说，特别是层次模型，也许会有一大堆答案。造型中的这种解决方案被称为*反向运动学*和*反向动力学*，它与已经讲过的*前向运动学*和*前向动力学*对应。

例如，如果你想挠耳朵，就得把手移到耳朵上。但当它到达那里时，你的肘节可位于若干不同位置（贴近你的身体或伸向身体侧面）中的任何一个。也就是说，你的上臂、小臂和手腕

⊖ 这种模拟既可以基于运动方程的解析解，又可以基于解微分方程的软件包提供的数值解。

的运动不完全决定于指令“把手移向耳朵”。因此，解决反向运动学问题可能很困难。但是，通常情况下，解只有惟一解的方程比解有多个解的方程要容易一些，所以，如果我们给上述问题加入约束条件（例如，“在运动的每个阶段，你的胳膊的势能尽可能小。”），则解就是惟一的了。注意，你还受以下条件的约束：你的身体的结构及环境中其他物体——当你穿着宇航服时挠耳朵比你穿浴衣时要难得多。

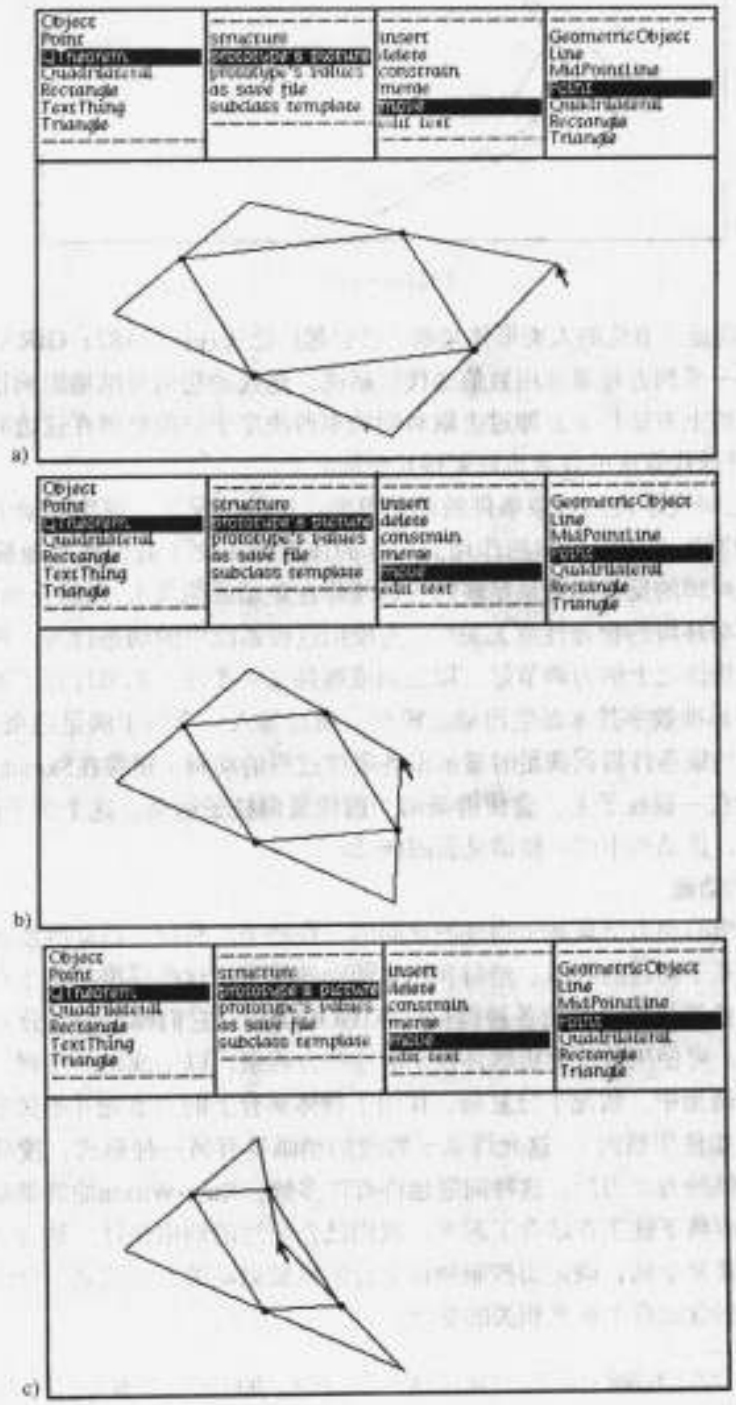


图21-9　ThingLab的运行界面。（经Alan Boming、Xerox PARC和华盛顿大学许可。）

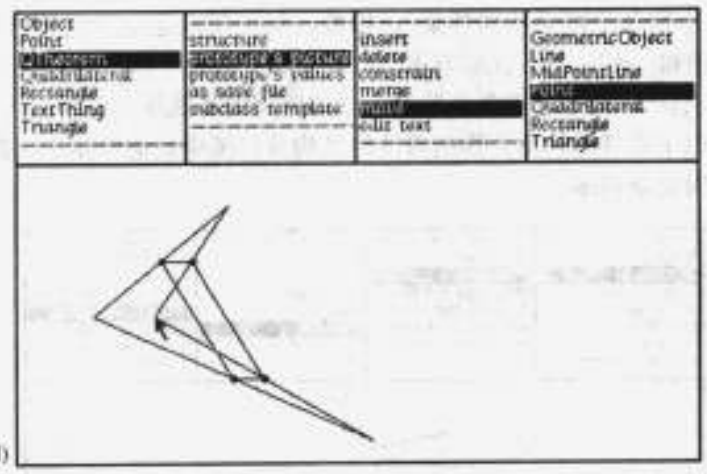


图21-9（续）

这类问题，特别是关节化的人类形体动画，已引起广泛关注[CGA82；GIRA85；WILH87]。反向问题所产生的一系列方程通常用数值迭代法解决。迭代的初值对结果影响很大（例如，机器人的胳膊从桌子的上方还是下方伸过去取对面的东西决定于它的胳膊在这边时是在桌子上方还是在下方），而且迭代算法的收敛也许要很长时间。

[BARR88]中已研究了使用约束条件的动态模型。这种情况下，模型的动力学可能会复杂得多。例如，你的脚底对地板施加的作用力与你的体重成正比（假设你和地板都没有运动），一般情况下，地板对脚的反作用力是足够的（即使你在走动或跑动），以防止你的脚陷入地板。这就是说，作用力与环境的物理性质无关[⊖]。为模拟这种系统中的动态行为，我们可使用动态约束，它将作用于物体之上的力调节好、以达到或维持某一条件。如果计算了维持约束条件所必需的力，就可用标准数字技术派生出动态模型。通过加入一个用于满足约束条件的作用力，我们可以产生出在约束条件得到满足时显示出各事件过程的动画（很像在Sketchpad中）。例如，规定链条的一端拴在一根柱子上，会使链条由当前位置向柱子运动。这个例子是Barr和Barzel的一个动画的主题，该动画中的一帧请见彩图IV-26。

21.3.7 基于物理的动画

1074 - 1076

我们上面所介绍的动力学是基于物理的动画的一些例子。所以，以动画形式来讲，它们就是第20章所介绍的基于物理的衣服、塑料和刚体运动的模型。这些模型都基于对物理系统演变的模拟。已开发出典型机械运动的各种模拟方案[GOLD80]；它们都用偏微分方程组的解来表示物理系统的演变。可使用数值分析软件包来解这些方程组，以生成动画序列。在第20章所讲的Kass-Witkin运动造型中，情况十分复杂。作用于物体集合上的力事先并不知道，因为物体可能自身能提供力（如使用肌肉）。这允许基于物理的动画具有另一种形式：搜寻产生某些动作所必需的由肌肉提供的力。当然，这种问题也许有许多解，Kass-Witkin的答案是选择工作量最小的途径。这类动画将下述工作结合了起来：我们已介绍过的约束条件、动力学、过程化控制和演员。它也是非常复杂的；确定出控制物体集合的机械运动的方程组是十分困难的，因为这样的方程组也许会包含成百个彼此相关的变量。

⊖ 当然，地板在你踏上时确实移动了，但只不过是一点。通常，我们避免将地板模拟为大质量物体，而是将其模拟为固定物体。

21.4 动画的基本规则

从1925年到20世纪30年代末，传统的角色动画由一种艺术形式发展为类似沃特迪斯尼工作室这样的工业。起初，动画所需要的不过是一系列的卡通画片——静止图像的一个集合，将它们组合在一起，就成了动画。动画技术不断发展的同时，其中所涉及的一些基本原则成为了角色动画的基本规则，而且在今天仍被使用[LAYB79；LASS87]。尽管这种原则来源于卡通人物动画，但许多同样适用于逼真的三维动画。这些规则以及在三维人物动画中它们的运用在[LASS87]中有全面的评述。在这里，我们仅介绍一些最重要的规则。我们知道这些规则并不是绝对的，这一点十分重要。就像许多现代艺术已脱离绘画的传统框架一样，现代的动画制作者也已脱离了传统动画的框架，而且其作品通常十分精彩（例见[LEAF74；LEAF77]）。

传统规则中最重要的一条是*挤压和拉伸*，它通过物体的变形来指示其物理属性。橡胶球或皮球在掉在地上时都会变形（以不同方式）。一个跳动的皮球在到达地面时看上去被拉长了（运动模糊的先兆），撞击地板时则被压扁，而再度弹起时又被拉长了。与此相反，一个金属球撞上地板时变形很小，但也许会在撞击后产生振动，表现为幅度很小而频率很高的多次变形。对第20章中介绍的Luxo Jr.的那一跳，用本章所介绍的基于物理模型的建模技术进行模拟，得到的是挤压和拉伸运动的动画：Luxo先蹲下，在肌肉中存储势能；而后蹦出去，身体完全伸展开来并将自己抛向前方，接着着地，再次蹲下以吸收向前运动的动能使自己不至于摔倒。它是对Kass-Witein模拟技术的很好的证明，因为运动过程是自动生成的；它也很好地证明了传统动画制作者有能力估算出复杂的偏微分方程的解。

第二条重要规则是使用*渐入*和*渐出*来辅助生成平滑的插值，因为突发的快速运动特别烦人。这在对相机位置（绘出或计算动画时所基于的观察点）进行插值时尤为明显。观众观看动画与 1077
相机拍摄是一样的，因此，相机位置的快速变化可能会使他感到眼花缭乱。这意味着，相机位置的变化应尽可能平滑。

第三条规则是正确地*分解*动作的各阶段。这一条规则自然地由二维角色动画沿用到三维动画，不管是用于娱乐业还是科学计算可视化。这包括选择一个可传达动画中所发生事件的尽可能多的信息的视角，以及（如果可能的话）对事件进行分解以使每一时刻只有一个事件获取观察者的注意力。在科学计算可视化动画中，这种分解也许是不可能的（所模拟的事件也许是并发的），但却可能从某个角度观察场景，其中不同事件处于图像的不同部分，且每个事件都可单独观察而不再像和其他事件一起观察时那样一团糟。

动画设计的许多其他方面也非常重要。虽然逐渐发展出一些基本规则，但很多情况都可视情况而定，并无严格要求。色彩的正确使用常被忽略，结果就产生了花花绿绿的、不能凭颜色分辨出各个物体的动画。动画制作的时间大部分花在计算上面而不是用来设计最终效果；不花点时间设计动作的前奏，将动作合理分解或平滑地结束这些动作，则最终动画中动作看起来就是一闪而过。过多关注动画细节的代价是整体感觉的下降，这样的动画毫无美感。如果你打算做动画，要考虑到上述难点，并在动画制作过程中分配尽可能多的时间在对美感的考虑上。

21.5 动画特有的一些问题

正如从二维图形到三维图形的过渡产生了许多新问题和新的挑战一样，从三维过渡到四维（加入时间维度）也有其特有的问题。其中之一就是*时域走样*。就像二维和三维图形的走样问题可由提高屏幕分辨率部分地加以解决一样，动画中的时域走样问题可由提高时域分辨率部分地

解决。当然，二维图形对此的另一解决办法是反走样；三维图形中相应的办法是时域反走样。

四维绘制的另一个问题是我们需要绘制许多十分相似的图像（在理想动画中，相邻帧间图像差别不大，否则，我们会得到帧间有跃变的动画）。这个问题很像二维图像中对多条扫描线进行绘制：一般来说，每条扫描线与它上面的那条看起来很相似。正如扫描线绘制利用了扫描线间的相关性一样，我们也能利用帧间的相关性。在光线跟踪中，我们通过在一个四维时空的外包盒（三个空间维度和一个时间维度）中考虑整个动画来做到这一点。每个物体在它沿时间轴运动时，都描述了四维时空的一个区域。例如，一个静止的球描述了四维空间的一个球形管道。图21-10中显示了三维空间相应的情形：假设我们制作一个关于图21-10a所示的圆的二维动画，
1078 其相应的三维时空的外包盒见图21-10b。这个圆在三维空间中扫出一个圆柱体。在四维的情况下，所绘制的每幅图像对应于四维时空的一个三维片的二维图像。这就是说，我们从一个特殊的时空点（$x, y, z; t$）投射出一些光线，它们的方向向量的时间分量为0，则所有的光线均经过所有时间坐标也为t的点。通过在这个四维时空中应用光线跟踪的空间分割技术，我们可以节省大量时间。动画的全过程可使用同样的超空间分割方法，于是不需要为每帧重复进行空间分割而浪费时间了。这个思想以及光线跟踪中其他利用了帧间相关性的想法在[GZAS88]中有介绍。

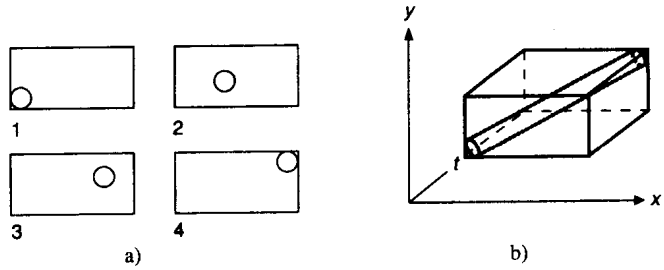


图21-10 a)图中的圆从左下角向右上角移动。沿另一条轴将这些图叠起来，就得到如图b)所示的空间-时间动画；这组圆在空间-时间系中变成为一根管子

高时域分辨率（每秒很多帧）看上去似乎不必要。毕竟，视频动画[⊖]看起来是平滑的，而它的帧速不过是每秒30帧。但是，帧速是每秒24帧的电影经常有跳跃感，特别是当大型物体朝观察者的方向快速移动时，就像有时在移动动作中发生的那样。而且，像以前提到过的，由于视觉暂留的缘故，电影里车轮有时看起来向后转。高时域分辨率帮助解决了这些问题。每秒的帧数加倍使得车轮的转速加倍而看起来不再向后转，当然这有助于使屏幕上快速移动的物体的运动更平滑。新的Showscan技术[SHOW89]包括了使用70 mm胶片录制和播放每秒60帧的电影；这可以产生更大的图像，因而包含了更大的可视区域，并且使动作更平滑。

时域反走样可通过取信号的多个样本并计算它们的加权平均值来实现。在这种情况下，这些样本必须是沿时间方向而不是沿空间方向的，因此我们取某一点在连续几个时刻的图像中的值，并对它们进行加权计算，以得到某帧中该点的值。时域走样问题已有多种解决方法；超采样、时域内的盒式滤波及其他所有源于空间反走样技巧（包括后置滤波！）都有应用。最成功的方法之一是第16章中介绍的分布式光线跟踪法 [COOK86]。

1079 另一个减少时域走样的技巧值得注意：区域动画。传统电影的每个图像要扫描两次；先画出所有偶数行的扫描线，再画奇数行，如此反复。每条扫描线每隔1/30秒被重画一次，但偶数行和奇数行的扫描线分两轮画。这就是说，电子束每1/60秒扫过一次整个屏幕。若动画中偶数

⊖ 这里所说的视频动画是用摄像机拍摄的，而非合成的。

行扫描线上点的颜色在时间t时计算，而奇数行的在时间$t + 1/60$秒时计算，它们合并成一幅点阵图。每帧重复一遍这个过程，则当动画显示时，虽然每条扫描线隔1/30秒才刷新一次，其效果却类似于每秒60帧的动画。但使用这种技巧需付出一定代价：动画的静止图像看上去质量不如原来图像，因为它们是由两个不同时刻的图像组合而成的，而且由此导致它们在隔行显示器上看起来有闪烁。还有，必须绘制两倍数目的帧，而且必须计算同样数目的插值位置，等等。尽管有上述缺点，这项技术还是在计算机动画工业领域中广泛采用。

动画中的另一个极端是基于“二拍一”动画、“三拍一”动画或“多拍一”的动画，其中动画以低于显示器刷新率的时域分辨率生成。通常，动画的每一帧被显示为电影的两帧（“二拍一”），于是视频的有效刷新率只有每秒12帧而不是每秒24帧。这种方法必然产生跳变图像或模糊图像（如果不采用时域反走样）。但是，以多帧动画作为基础，而后填入插值帧，对制作动画来说很有用的，因为它允许动画制作者远在单独的各帧生成之前就能得到动画的大致轮廓。

21.6 小结

计算机动画是一个新兴学科，其中高级动画最近才开始研究。因为动画制作者可利用的计算机能力不断增加，动画软件包也更加成熟，生成高质量的计算机动画将更为简单。但在目前，必须有许多折衷。模拟软件似乎在飞速发展，而图形模拟的自动生成也似乎仅一步之遥。另一方面，在动画软件包含传统动画技巧的知识之前，计算机角色动画与其说是艺术，不如说是科学，而且动画制作者的“眼睛”也将继续对动画质量产生巨大影响。

习题

21.1 考虑一个对角顶点为(0, 0)和(1, 1)的单位立方体。假设我们用点(0, 1)，(1, 1)和(1, 0)定义了一条多边形路径，并想按此顶点顺序将它变形为由点(1, 0)，(0, 0)和(0, 1)定义的多边形路径（即我们想把它旋转180°）。请画出对顶点进行线性插值所得结果的各中间阶段。这说明除非关键帧间距离不大，否则严格的顶点插值并不适合用于关键帧插值。

21.2 假设你正在制作动画，而且可以以任意顺序生成各帧。若动画长度为128帧，第一张“速写”可由绘制第一帧得到，而后在所有128帧中都显示这张“速写”。（这是很低的时域分辨率！）。第二种近似生成动画的方法是在前64帧中显示第1帧，后64帧中显示第64帧。假定你有一台录影机，可为给定数目的视频帧在每一给定帧处记录一幅给定的图像。基于下列想法：采用绘制某些帧再记录它们以得到整个动画的近似，并最终获得完整动画，写出序列化近似记录格式的伪码。假定整个动画的帧数是2的幂。（这道题由Michael Natkin和Rashid Ahmad提供。） 1080

21.3 使用基于颜色表的显示设备，实现图21-4中所示的动画。你能想出使生成的动画更平滑的方法吗？

21.4 使用支持移动和放缩操作的帧缓存，实现21.1.2节中所介绍的移动-放缩动画技术。

21.5 使用20.5节中的粒子系统制作一个焰火动画。如果你的帧缓存不能显示图片，你可以使用PostScript编写粒子系统的程序，而后将结果图片打印出来。把它们像书一样叠起来，而后快速翻页，就能得到一个翻动书页（flip-book）动画。

21.6 假设你想做一个以扫描进来的手工绘图作为数据源的二维动画系统，想想可用哪些技术对它进行自动修饰，包括将接近封闭的环闭合起来，对曲线（除尖角部分外）

进行平滑，等等。这个过程的自动化十分困难，可以试着思考在以交互绘图软件的结果作为二维动画材料来源的前提下，怎样实现此过程的自动化。

21.7 a. 假设q和r分别对应于绕轴$\mathbf{v}$旋转了ϕ和θ的四元数。精确计算qr的值并使用三角恒等式证明它对应于绕$\mathbf{v}$轴旋转角度$\phi+\theta$。

b. 证明两个单位四元数之积仍为单位四元数。

c. 若q为单位四元数，记为$a+b\mathbf{i}+c\mathbf{j}+d\mathbf{k}$，$s$为四元数，记为$x\mathbf{i}+y\mathbf{j}+z\mathbf{k}$，我们可得到一个新的四元数$s'=qsq^{-1}$其中$q^{-1}=a-b\mathbf{i}-c\mathbf{j}-d\mathbf{k}$，如果我们将$s'$记为$x'\mathbf{i}+y'\mathbf{j}+z'\mathbf{k}$，则系数$x', y', z'$取决于$x, y, z$。找出满足条件$[x'\ y'\ z']^t=Q[x\ y\ z]^t$的矩阵$Q$。当我们用四元数生成旋转时，使用的应该是这种矩阵形式，而不是显式地计算四元数的乘积。

d. 证明：若向量$[b\ c\ d]^t$被Q左乘后值不变，则Q代表绕向量$[b\ c\ d]$的旋转。它表示角度为$2\cos^{-1}(a)$的旋转，因此，它给出了四元数和旋转间的对应关系。

1081

附录　计算机图形学的数学基础

该附录回顾了本书所用的大部分数学知识。这只是简要的介绍，而不是像线性代数、几何、微积分的教科书那样进行详细的讨论。但是，我们会着重讨论一下仿射空间，因为现在许多关于线性代数的书中都没有提到它。在本附录的论述中，将穿插一些习题。在查看习题的答案前，读者最好能认真思考一下。习题的解答一般很简单，主要是让你知道你的答案是否正确，而不是告示你如何去做。

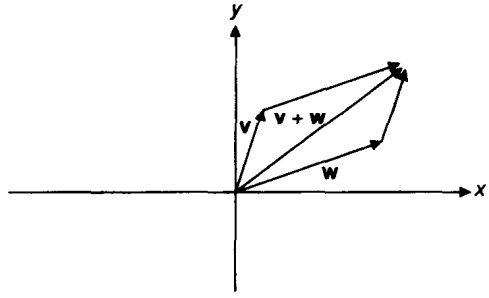
我们假定读者已学过平面几何、微积分和线性代数等课程，只是对它们已有些淡忘了。这样，我们主要是给出一些重要概念的定义和一些重要的结果，而省略关于它们的大部分证明。因为我们发现：对证明感兴趣的读者，一般都能自己给出证明，而对证明不感兴趣的读者，却觉得这些证明比较烦人。如果读者想详细了解这些内容，请参考[BANC83；HOFF61；MARS85]。

附录的第一部分将比较详细地描述仿射空间的几何特性。在随后各节中所讨论的内容，大家应该是比较熟悉的，因此只做简要介绍。最后一节，将讨论实数方程的求根方法，这与其他各节内容关系不大。

A.1　向量空间和仿射空间

通俗地说，向量空间就是对数量的加法和乘法能够实施操作的一个空间。准确地说，向量空间包含一个集合和两种操作。集合中的每个元素称为*向量*，它们一般用黑体字母表示，如**u**, **v**, **w**。两种操作就是向量加法和实数与向量相乘的纯量乘法[⊖]。这里，加法操作必须满足交换律、结合律以及基于一个单位元素（通常称为**0**）的可逆性（即，对任何一个向量**v**，必有一个向量**w**满足$\mathbf{v}+\mathbf{w}=\mathbf{0}$；**w**被写为“$-\mathbf{v}$”），单位元素也是一个向量，它的特点是：对任何一个向量**v**，$\mathbf{v}+\mathbf{0}=\mathbf{v}$。纯量乘法必须满足下列规则：$(\alpha\beta)\mathbf{v}=\alpha(\beta)\mathbf{v}$, $1\mathbf{v}=\mathbf{v}$, $(\alpha+\beta)\mathbf{v}=\alpha\mathbf{v}+\beta\mathbf{v}$, $\alpha(\mathbf{v}+\mathbf{w})=\alpha\mathbf{v}+\alpha\mathbf{w}$。

向量空间的这个定义概括了平面的基本几何特性。因此，一个平面可以转换成一个向量空间，而此时的向量集合就是平面上点的集合。这种将向量当成点的处理是临时的，只在此例中使用。现在，我们就将平面上的一个点和一个向量当成一回事。在将平面转换成一个向量空间时，我们还得先在平面上选择一个称为*原点*的特殊点。根据熟知的*平行四边形法则*，向量加法定义为：向量**v**和**w**的相加，就是先从原点出发画一条带箭头的直线到**w**，然后，平移该直线使它的起始点落在点**v**上，则此时箭头端点就为$\mathbf{v}+\mathbf{w}$的向量。如果再画一条从原点到**v**的带箭头的直线，并且也对它进行类似的平移操作，我们就得到了一个平行四边形，如图A-1所示。一个实数α与一个向量相乘的纯量乘法的定义也是类似的：从原点到**v**画一条带箭头的直线，根据α的值改变这条直线的长度，而该直



图A-1　平面上的向量加法运算

⊖ 标量（或实数）将用希腊字母表示，特别是用那些在字母表中靠前的一些字母。

线的起始点保持不变，则直线的箭头端点即为所定义的$\alpha\mathbf{v}$。显然，在实数域的三维欧氏空间中，向量加法和纯量乘法的定义也是相同的。

习题 考察上面所讨论的向量空间的结构，问这种结构是否依赖于平面上的点被赋予的坐标，或者该结构是纯几何结构？假设我们以绘图中所熟悉的方式给平面上的点赋予坐标，那么如果我们对坐标为(a, b)和(c, d)的两个向量相加，其结果向量的坐标是怎样的？再假设我们重
1084 新定义坐标轴，使一组坐标线是水平的，而另一组坐标线却不是垂直的，而是与垂直方向有一个30°的夹角，那么，这两个向量的和的坐标又会怎样呢？

解答 向量空间的结构与坐标无关，只是一种纯几何结构（在此，几何包括距离的量度）。在这两种情况下，两个向量的和的坐标都是$(a+c, b+d)$。 □

一个典型的向量空间是$\mathbf{R}^n$，它的每个向量都是一个由实数的有序排列作成的n元组。其中加法定义为各个相应的分量相加，而纯量乘法也是对各个分量分别相乘。$\mathbf{R}^n$中的元素是以垂直的方式书写的，于是$\mathbf{R}^3$的一个向量如下：

$$\begin{bmatrix}1\\3\\1\end{bmatrix}$$

而其向量加法，如下所示：

$$\begin{bmatrix}1\\3\\1\end{bmatrix}+\begin{bmatrix}2\\3\\6\end{bmatrix}=\begin{bmatrix}3\\6\\7\end{bmatrix}$$

图形学中通常仅考虑$\mathbf{R}^2$，$\mathbf{R}^3$，$\mathbf{R}^4$。

假设一个向量空间中有上述两种操作，那么用向量就可以很方便地处理一些事情。其中之一是线性组合。向量$\mathbf{v}_1,\cdots,\mathbf{v}_n$的线性组合就是可以表示成后面形式的任何向量：$\alpha_1\mathbf{v}_1+\alpha_2\mathbf{v}_2+\cdots+\alpha_n\mathbf{v}_n$。向量的线性组合可以用来描述许多对象。以笛卡儿平面为例，一条穿过坐标原点和一个非零的点$\mathbf{v}$的直线就能表示成所有形式为$\alpha\mathbf{v}$的向量的集合，在此α是任意实数。而从原点出发穿过$\mathbf{v}$的射线，其形式也是一样的，只是α不能是负实数。以上是关于一个向量的线性组合的两个例子。下面，我们会遇到一些复杂的线性组合情况。

一般，一组向量的所有可能的线性组合就构成了该组向量生成的线性空间。以笛卡儿平面为例，一个非零向量的线性空间就是穿过原点的一条直线，而两个方向不同的向量的线性空间就是一个平面。

在继续讨论向量空间之前，我们先讨论一下仿射空间。一个仿射空间可以大致地定义为能进行几何操作的一个集合，但其中没有任何特别的点。（在向量空间中，$\mathbf{0}$向量就是一个特别的点，这在笛卡儿平面的例子中就能反映出来，在其加法和乘法的定义中，原点具有很重要的作用。）仿射空间的比较精确的定义是：它有一个集合（其中的元素称为仿射空间的点）、一个相关的向量空间和两种操作。任给两个点，如P和Q，我们能生成这两个点之间的差，该差是向量空间中的一个元素；对于一个点P和一个向量$\mathbf{v}$，我们可以将这个向量加到这个点上得到一个新的点：$P+\mathbf{v}$。当然，这些操作还必须满足一些特征，如$(P+\mathbf{v})+\mathbf{w}=P+(\mathbf{v}+\mathbf{w})$，以及当且仅当$\mathbf{v}=0$时，$P+\mathbf{v}=P$。

这个定义是基于更一般的、没有所谓原点的几何模型给出的，比如说像桌面这种有边界的平面，它上面并没有一个自然的原点，因为桌面上没有哪个点与其他点不同。但是，如果你在

桌面上选择一个点P，再从P点出发形成一组坐标轴，那么，基于这个坐标系就能根据桌面上 1085
其他点与点P之间的差别，为它们进行度量。将坐标轴上的所有点以一个常量平移，我们就得到了定位于另一个点的新坐标轴。在这个模型中，仿射空间的点就是桌面上的点，而向量就是这些点之间的有向连线。一个向量$\mathbf{v}$加到一个点P上，就是以P为起始点画一条有向线段，该线段的端点就是这个加法的结果（该端点被称为$P+\mathbf{v}$）。而对于两个点P和Q之间的差$Q-P$，就是画一条从P到Q的有向线段。

仿射平面给计算机图形学提供了一个便利的模型，因为在图形学中，通常并没有特别的点。比如说，当你为一间房子建模时，这房子中不会有一个自然的点要被选为原点。因此，我们在后面将一边讨论向量空间，一边讨论仿射空间。

仿射空间中点的线性组合没有什么意义（其中，甚至都没有关于纯量乘法的定义），但是我们可以定义一个与实数t相关联的关于点P和Q仿射组合。这个仿射组合的意义是指从P到Q的路径上由t作为比例因子决定的那个点。（如果t是0和1之间的小数，这就称为凸组合。）我们可以考察Q和P之间的差，$\mathbf{v}=Q-P$，即从P指向Q的一个向量。如果我们对此乘以t，我们就得到了长度变为t倍的一个向量。将此向量再加回到P上，我们就得到了与实数t相关联的关于点P和Q的仿射组合，即

$$P+t(Q-P)$$

对上式中包含P的项，读者可能会想到将其合并，以变成$(1-t)P+tQ$。但是，这根本就没有意义，因为这里没有定义点的纯量乘法。我们不排斥这样的表示法，而是将它简单地定义为：如果α和β是和为1的两个标量，并且P和Q是仿射空间中的点，我们将$\alpha P+\beta Q$定义成$P+\beta(Q-P)$。

对于更多点的仿射组合可以进行类似的定义：给出n个点$P_1,\cdots,P_n$，以及n个实数$t_1,\cdots,t_n$，它们满足$t_1+\cdots+t_n=1$，我们将由这些与t关联的这些P的仿射组合定义为$P_1+t_2(P_2-P_1)+\cdots+t_n(P_n-P_1)$，它们也可以写为$t_1P_1+t_2P_2+\cdots+t_nP_n$。

习题 每个向量空间都可以变成一个仿射空间。仿射空间的点就是向量空间中的向量。相关的向量空间就是原来的向量空间。点与点之间的差定义为向量与向量之间的差，而一个点与一个向量的和就是一般的向量和。试证明，在这种情况下，我们定义的形式为$\alpha P+\beta Q$（在此，$\alpha+\beta=1$）的点，实际上就是运用向量空间的操作所得的向量$\alpha P+\beta Q$。

解答 $\alpha P+\beta Q$定义为$P+\beta(Q-P)$。但是，运用一般的向量操作后，这就是$P+\beta Q-\beta P=(1-\beta)P+\beta Q=\alpha P+\beta Q$。 □

A.1.1 仿射空间中线的方程

如果P和Q是仿射空间中的两个点，形式为$(1-t)P+tQ$的点的集合就形成了穿过P和Q的一 1086
条直线；直线的这种表达形式有时称为*参数形式*，因为其中有参数t。笛卡儿平面是一个仿射空间，其上的点都用坐标(x,y)标识。因此，点(a,b)与点(c,d)之间的参数化直线是

$$L=\{((1-t)a+tc,(1-t)b+td)\mid t\text{是实数}\}$$

习题 假设我们定义两个点$(a,b,1)$和$(c,d,1)$之间的差是向量$(a-c,b-d)$，并类似地定义一个点与一个向量的和，证明：对于所有形式为$(a,b,1)$的实数三元组的集合，也能够形成一个仿射空间，其相关的向量空间是$\mathbf{R}^2$。运用前面所给的参数化直线的定义，证明：点$(1,5,1)$与点$(2,4,1)$之间的直线就是所有最后的坐标分量为1的点的集合。

解答 这条直线的定义是形式为$(1-t)(1,5,1)+t(2,4,1)$的点的集合，也就是$(1,5,1)+$

$t(1,-1)$。它们就是形式为$(1+t,5-t,1)$的点。所以，它们最后的坐标分量是1。 □

A.1.2 仿射空间中的平面的方程

如果P、Q和R是仿射空间中的3个点，并且它们不共线（例如，假设R不位于连接P和Q的线上），那么，由P、Q和R定义的平面就是如下形式的点的集合：

$$(1-s)((1-t)P+tQ)+sR$$

习题 解释为什么上述表达式具有几何意义。

解答 该表达式是两个点的仿射组合。第一个点是$(1-t)P+tQ$；第二个点是R。第一个点是P和Q的一个仿射组合。因此，其所有的项都是有几何意义的。

补充一点，关于平面的这种描述是参数化的，因为其中有两个参数s和t。

习题 包含所有实数三元组的集合$\mathbf{E}^3$是一个仿射空间，相关的向量空间是$\mathbf{R}^3$，其中的元素也是实数的有序三元组，只是它又基于这些元素定义了基本的加法和纯量乘法。像一个点与一个向量的和被定义成基本操作一样，$\mathbf{E}^3$中两个点的差也定义成基本操作。那么，包含点(1, 0, 4)，(2, 3, 6)和(0, 0, 7)的平面上的点是怎样的呢？

解答 这个平面上的点就是具有形式为$(1-s)((1-t)(1,0,4)+t(2,3,6))+s(0,0,7)$的所有点。因为所有的操作都定义成基本的操作，所以可将它表示成具有如下形式的所有点的集合，
1087 即$((1-s)(1-t)+2(1-s)t, 3(1-s)t, 4(1-s)(1-t)+6t+7s)$。 □

A.1.3 子空间

如果我们有一个向量空间V，以及V的一个非零的子集S，那么，如果S满足下面的条件，它就是V的一个线性子空间：对于S中任意的向量$\mathbf{v}$和$\mathbf{w}$，$\mathbf{v}+\mathbf{w}$和$\alpha\mathbf{v}$都是S中的向量，在此α是任意的一个实数。例如，如果$\mathbf{v}$是一个向量，那么，所有形式为$\alpha\mathbf{v}$的向量的集合就构成了一个子空间，因为，当$\mathbf{v}$的任意两个数量乘积相加时，我们得到的就是$\mathbf{v}$的另一个纯量乘积，而$\mathbf{v}$的一个纯量乘法的纯量乘法，就是对$\mathbf{v}$的另一个纯量乘法。在$\mathbf{R}^3$中，其子空间可以明确地列举出来。它们是：(1) 原点，(2) 任何穿过原点的直线，(3) 任何包含原点的平面，(4) $\mathbf{R}^3$自身。

习题 证明一个向量空间的任何线性子空间都必须包含$\mathbf{0}$向量。

解答 设$\mathbf{v}$是S中任意的一个向量。那么$-1\mathbf{v}=-\mathbf{v}$也在S中，因此$\mathbf{v}+(-\mathbf{v})$在S中，而这就是$\mathbf{0}$向量。但下面的解答就不合适：对向量乘以数量0就会得到$\mathbf{0}$向量。因为在向量空间中，并不能根据前提规则推导出$0\mathbf{v}=\mathbf{0}$，实际上，因为$(1+(-1))\mathbf{v}=\mathbf{v}+(-\mathbf{v})=0$，$0\mathbf{v}=\mathbf{v}$是成立的。由此所得的结论不会成为公理。 □

仿射子空间是一种更一般的情况。对于向量空间V中的一个非空的子集S，如果满足下面的条件，它就被称为一个仿射子空间：集合$S'=\{\mathbf{u}-\mathbf{v}\mid \mathbf{u},\mathbf{v}$是$S$中的向量$\}$是$V$的一个线性子空间。例如，笛卡儿平面上的任何直线都是一个仿射子空间。如果S是这样的一条直线，那么S'就是平行该线而穿过原点的直线。

如果S是一个向量空间的一个仿射子空间，那么S本身就可以当成是一个仿射空间。（注意，这不是一个向量空间。分析$\mathbf{R}^2$中的直线$x=1$就可知这一点，$\begin{bmatrix}1\\3\end{bmatrix}$和$\begin{bmatrix}1\\5\end{bmatrix}$都在这个仿射子空间中，但它们的和$\begin{bmatrix}2\\8\end{bmatrix}$就不是。）这个仿射空间的结构如下：相关的向量空间就是S'；根据定义可知S中两

个点的差属于S'，S中一个点与S'中的一个向量的和也是S中的一个点。

重要习题 证明：形式为$(x, y, z, 1)$的点的集合S是$\mathbf{R}^4$中的一个仿射子空间。相关的向量空间是什么？这个仿射子空间中两个点的差是怎样的？

解答 S中任意两个点的差是形式为$(a, b, c, 0)$的点，所有具有这种形式的点的集合是一般的加法和乘法定义下的一个向量空间（其实，它们本质上与$\mathbf{R}^3$是“一样的”）。相关的向量空间是形式为$(a, b, c, 0)$的四元组的集合。$(x, y, z, 1)$和$(x', y', z', 1)$的差就是$(x-x', y-y', z-z', 0)$。 □

上面这个例子很重要，因为它是第5章中所有内容的基础。在这个例子中，我们可以清楚地看到空间中的点和向量的区别：点是用来定义图形空间中物体的位置的，而向量是用来定义点到点的位移或方向的。不巧的是，Pascal语言中形式为（$x, y, z, 1$）的点是用一个带3个实数的数组来保存的，而形式为（$a, b, c, 0$）的向量也能以这种方式来保存。因此，许多人都误以为点和向量是可以互相交换的。其实，根本不能这样。在仿射空间和环境空间中，对点的表示 1088
都是用几个数形成的纵列，

这更加深了这种误解，但这已在实际中成了习惯。我们将$\mathbf{R}^4$中最后的坐标元素为1的所有点的集

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

合称为$\mathbf{R}^4$中*标准三维仿射空间*。类似地，我们在$\mathbf{R}^3$中也可定义标准的二维仿射空间，等等。

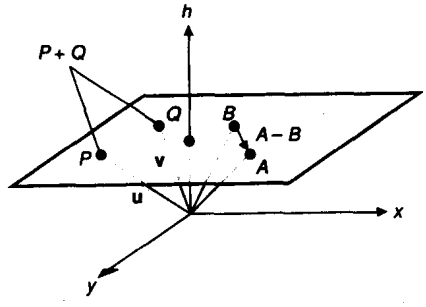
图A-2中显示了$\mathbf{R}^3$中的*标准仿射空间*。相比于画$\mathbf{R}^4$中标准的三维仿射空间，这个图是很容易画的。我们是想通过它来对一些比较复杂的情况进行直观的解释。标准仿射平面上的点是如下形式的三元组：

而其上的向量具有下面的形式：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ 0 \end{bmatrix}$$

（对于水平面，我们以字母“x”和“y”进行标记，而对垂直的轴用字母“h”来标记。这样处理是为了反映第三个坐标的特殊本质。）这个仿射空间的点的集合就形成了高出(x, y)平面一个单位的平面。此仿射平面上所画的有向线段是为了说明作为点之间差的向量的作用，若将这些向量的始点放到原点$(0, 0, 0)$，则它们的终点都会在(x, y)平面上。如图A-2所示，我们在仿射空间任选两个点P和Q，它们的和（作为$\mathbf{R}^3$中的向量）高出仿射平面整整一个单位。这从几



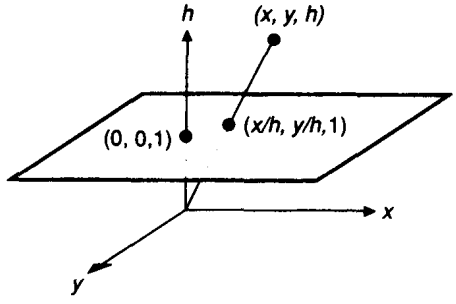
图A-2 $\mathbf{R}^3$中标准的仿射平面，即位于$h=1$的平面。P和Q是平面上的点，但它们的和位于平面之上。点A和B的差是一个水平向量

1089 何意义上揭示了对点进行加法操作的不恰当性。

图A-3显示了仿射空间上的一个重要操作：齐次化处理。如果我们在三维空间中随便选一个点

$$\begin{bmatrix} x \\ y \\ h \end{bmatrix}$$

并将它与原点连接成一条直线，这条直线将与仿射平面交于一个点。



图A-3 $\mathbf{R}^3$中的齐次化操作。点(x, y, h)被齐次化到了点$(x/h, y/h, 1)$

习题 计算这个交点。

解答 从点

$$\begin{bmatrix} x \\ y \\ h \end{bmatrix}$$

到原点的直线包含所有具有下面形式的点：

$$\begin{bmatrix} \alpha x \\ \alpha y \\ \alpha h \end{bmatrix}$$

我们要找到第3个坐标为1的那个点。这个点肯定位于$\alpha h = 1$的地方，也就是$\alpha = 1/h$。因此，这个点的坐标是

$$\begin{bmatrix} x/h \\ y/h \\ h/h \end{bmatrix} = \begin{bmatrix} x/h \\ y/h \\ 1 \end{bmatrix}$$

自然，当$h = 0$时，不能进行这种操作。毫不奇怪，在几何上，(x, y)平面上的一个点与原点
1090 的连线，永远不会与此仿射空间相交。

A.2 向量空间中的一些标准结构

A.2.1 线性相关性和生成空间

我们已经把一组向量的生成空间定义为这些向量的所有线性组合的向量的集合。如果我们在三维空间$\mathbf{R}^3$中讨论，一个向量（**0**向量除外）的生成空间就是包含该向量的穿过原点的直线；而两个向量（它们是非零的，且它们之中的任一个不在另一个的生成空间中）的生成空间是包含这两个向量且穿过原点的平面；至于三个向量（它们是不共面的非零向量）的生成空间，就是三维空间$\mathbf{R}^3$本身。

在上段讨论的各种奇异情况（也就是由附加的条件所排除的那些情况）中，所涉及的向量被称为是线性相关的（或者，简单地称为相关的）。其实，如果一组向量中的某一个位于其他向量的生成空间中，则这组向量是线性相关的。

习题 对于三个向量$\mathbf{a}=\begin{bmatrix}1\\3\end{bmatrix}$，$\mathbf{b}=\begin{bmatrix}2\\6\end{bmatrix}$，$\mathbf{c}=\begin{bmatrix}4\\1\end{bmatrix}$，通过证明**b**位于**a**和**c**的生成空间中，来证明这三个向量是相关的。然后，再证明**c**并不在**a**和**b**的生成空间中。

解答 $\mathbf{b}=2\mathbf{a}+0\mathbf{c}$。另一方面，**a**和**b**的生成空间是由所有具有以下形式的向量组成：

$$t\mathbf{a}+s\mathbf{b}=\begin{bmatrix}t\\3t\end{bmatrix}+\begin{bmatrix}2s\\6s\end{bmatrix}=\begin{bmatrix}t+2s\\3t+6s\end{bmatrix}=(t+2s)\begin{bmatrix}1\\3\end{bmatrix}$$

因此，这个生成空间中的任何向量必是$\begin{bmatrix}1\\3\end{bmatrix}$的纯量乘积，而向量**c**并非如此。

线性相关的更精确的定义是：向量$\mathbf{v}_1,\cdots,\mathbf{v}_n$是线性相关的，如果存在标量$\alpha_1,\cdots,\alpha_n$，满足：(1)$\alpha_i$s中至少有一个非零；(2)$\alpha_1\mathbf{v}_1+\cdots+\alpha_n\mathbf{v}_n=\mathbf{0}$。

习题 如果向量$\mathbf{v}_1,\cdots,\mathbf{v}_n$是相关的，证明它们中的一个向量必定位于其他向量的生成空间中。

解答 因为这些向量是相关的，所以存在标量数$\alpha_1,\cdots,\alpha_n$，使得$\alpha_1\mathbf{v}_1+\cdots+\alpha_n\mathbf{v}_n=\mathbf{0}$。并且这些标量数不全为0。不失一般性，设$\alpha_1$是非零的，（如果必要，通过重新排序可以做到这一点）。然后，我们对上面的方程求解，可得$\mathbf{v}_1=(1/\alpha_1)\alpha_2\mathbf{v}_1+\cdots+(1/\alpha_1)\alpha_n\mathbf{v}_n$，这样，$\mathbf{v}_1$就是位于其他向量的生成空间中。 □

如果向量$\mathbf{v}_1,\cdots,\mathbf{v}_n$不是相关的，就称为是线性无关的（或简称无关的）。这个定义比较麻烦，因为它要求证明一个相反的命题。下面将会看到，至少对于$\mathbf{R}^n$中的向量组，我们可以正面地再给出一个定义：当且仅当有某个数（某个矩阵的行列式）不为零时，一组向量是线性无关的。

我们也可以给仿射空间定义相关性和生成空间。仿射空间中一组点$P_1,\cdots,P_n$的生成空间可以用几种方式定义。其一，它是这组点的所有仿射组合的集合；其二，考察它们在相关的向量空间中的向量$P_2-P_1,P_3-P_1,\cdots,P_n-P_1$所构成的生成空间$S$，那么，这个仿射生成空间就可定义为具有形式为$P_1+\mathbf{v}$的所有点的集合，这里，**v**是$S$中的向量。 1091

如同向量空间一样，仿射空间中的一群点可以被称为是相关的，如果它们中的某一个属于其他点的（仿射）生成空间。如果不是线性相关的，就是无关的。

在此，我们再讨论$\mathbf{R}^3$中关于标准仿射平面的一种特殊情况，它的点具有下面的形式，

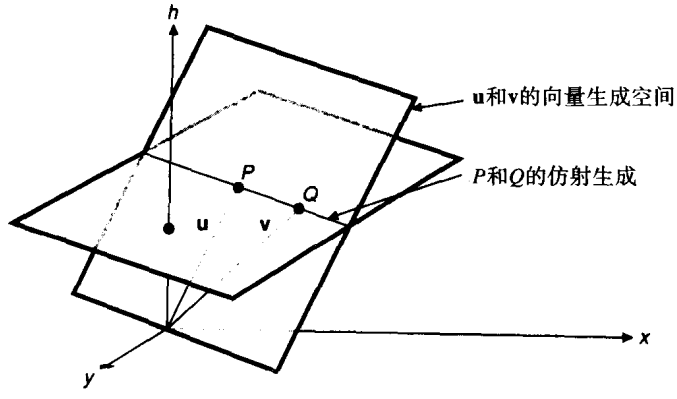
$$\begin{bmatrix}x\\y\\1\end{bmatrix}$$

如果我们在此仿射空间中选取两个点，就可以形成它们的仿射生成空间，即包含此两点的直线。我们也可以用另一种方法来形成它们的生成空间，即把它们当成$\mathbf{R}^3$中的向量，从而形成这些向量的生成空间。图A-4揭示了这两个生成空间之间的关系——仿射生成空间是向量生成空间（穿过原点的一个平面）与仿射空间相交的部分。

A.2.2 坐标、坐标系和基

运用生成空间的理论，我们可以用一些简洁的符号来描述向量空间中一些大的集合。例如，我们已经把直线和平面分别作为一个或两个向量的生成空间。对于直线，我们选取位于其中的

两个向量，并用它们来描述这条直线，并称该直线是这两个向量的生成空间，当然，这有些多余——一般来说，这两个向量中的每一个都位于另一个的生成空间中。



图A-4 $\mathbf{R}^3$中向量u和v构成的向量生成空间与P和Q构成的仿射生成空间之间的关系，在此，P和Q是u和v在仿射平面中的端点

所谓向量空间的基，就是一个向量子空间的最小的生成集合。最小的意思是：任何一组数量比较少的向量都有一个比较小的生成空间。这样，在我们前面的例子中，由两个向量生成直线这种线性空间，并不是最小化的，因为，可以去掉其中的一个，而余下的一个通过线性组合
1092 依然可以生成这条直线。

习题 向量空间的一个子空间的一组基，总是线性无关的。

解答 设$\mathbf{v}_1,\cdots,\mathbf{v}_n$是子空间$S$的一组基，并假设$\mathbf{v}_1,\cdots,\mathbf{v}_n$是相关的。那么（如果必要，可以重新排序），我们可以找到标量数$\alpha_2,\cdots,\alpha_n$使得$\mathbf{v}_1=\alpha_2\mathbf{v}_2+\cdots+\alpha_n\mathbf{v}_n$。在向量$\mathbf{v}_1,\cdots,\mathbf{v}_n$形成的生成空间中选一个典型元素$\beta_1\mathbf{v}_1+\cdots+\beta_n\mathbf{v}_n$，此时，$\beta_1\neq0$，这种表达式可以重写为$\beta_1(\alpha_2\mathbf{v}_2+\cdots+\alpha_n\mathbf{v}_n)+\beta_2\mathbf{v}_2+\cdots+\beta_n\mathbf{v}_n$。通过重新安排，该表达式可以变成向量$\mathbf{v}_2,\cdots,\mathbf{v}_n$的一个线性组合。这样，$\mathbf{v}_1,\cdots,\mathbf{v}_n$的生成空间中的任何向量，也在$\mathbf{v}_2,\cdots,\mathbf{v}_n$的生成空间中，因此，$\mathbf{v}_1,\cdots,\mathbf{v}_n$不是生成该子空间的最少向量的集合。所以，假设$\mathbf{v}_1,\cdots,\mathbf{v}_n$是相关的前提是不对的。

假设向量空间的一组基为$\mathbf{v}_1,\cdots,\mathbf{v}_n$，那么，该空间中的每一个向量v都可以表示成一个线性组合$\alpha_1\mathbf{v}_1+\cdots+\alpha_n\mathbf{v}_n$。假设我们将v写成另一种线性组合的形式$\mathbf{v}=\beta_1\mathbf{v}_1+\cdots+\beta_n\mathbf{v}_n$。那么，通过相减，我们可得$\mathbf{0}=(\beta_1-\alpha_1)\mathbf{v}_1+\cdots+(\beta_n-\alpha_n)\mathbf{v}_n$。因为我们假设这两个线性组合是不同的，因此，某些$\alpha_i$肯定与相应的$\beta_i$不同。通过重新排列，我们可以假设$\alpha_1\neq\beta_1$。于是，我们可以像前面对$\mathbf{v}_1$进行求解一样，将它表示成其余向量的组合，由此可得$\mathbf{v}_1,\cdots,\mathbf{v}_n$不是线性空间的最小化生成集合。所以，向量空间中的每一个向量关于基的线性组合是惟一的。

如果我们有一组向量空间的基，$\mathbf{B}=\{\mathbf{v}_1,\cdots,\mathbf{v}_n\}$，则对于向量空间中的一个向量v，存在一组标量数$\alpha_1,\cdots,\alpha_n$，使得$\mathbf{v}=\alpha_1\mathbf{v}_1+\cdots+\alpha_n\mathbf{v}_n$。这些标量数的集合可以看作是$\mathbf{R}^n$中的一个元素。这个元素表示为以下的形式：

$$\begin{bmatrix}\alpha_1\\ \cdot\\ \cdot\\ \cdot\\ \alpha_n\end{bmatrix}$$

它被称为 $\mathbf{v}$ 的相对于基 $\mathbf{v}_1,\cdots,\mathbf{v}_n$ 的坐标向量。

习题 在空间 $\mathbf{R}^3$ 中，有一组标准基，$\mathbf{E}=\{\mathbf{e}_1,\mathbf{e}_2,\mathbf{e}_3\}$，其中

$$\mathbf{e}_1=\begin{bmatrix}1\\0\\0\end{bmatrix},\qquad \mathbf{e}_2=\begin{bmatrix}0\\1\\0\end{bmatrix},\qquad \mathbf{e}_3=\begin{bmatrix}0\\0\\1\end{bmatrix}$$

那么，相对于这组基，向量

$$\mathbf{v}=\begin{bmatrix}3\\4\\2\end{bmatrix}$$

的坐标是什么？ 1093

解答 因为 $\mathbf{v}=3\,\mathbf{e}_1+4\,\mathbf{e}_2+2\,\mathbf{e}_3$，所以它们是

$$\begin{bmatrix}3\\4\\2\end{bmatrix}$$

在仿射空间中，相应的定义是类似的。仿射空间中一组无关的点，如果它们的生成空间就是整个仿射空间，它们就称为一个坐标系。如果 $P_1,\cdots,P_n$ 是一个坐标系，那么，仿射空间中的每一个点，都能写成惟一的关于 $P_1,\cdots,P_n$ 的仿射组合；这些系数，被称为相对于坐标系 $P_1,\cdots,P_n$ 的点的仿射坐标。

习题 如果 $P_1,\cdots,P_n$ 是仿射空间的一个坐标系，证明：$P_2-P_1,\cdots,P_n-P_1$ 是相关向量空间的一组基。

解答 设 $\mathbf{v}$ 是相关的向量空间中的一个向量，并设 $Q=P_1+\mathbf{v}$。那么 Q 能够写成关于 $P_1,\cdots,P_n$ 的一个仿射组合。这样，会有一组标量数 $\alpha_1,\cdots,\alpha_n$ 满足 $\alpha_1+\cdots+\alpha_n=1$，并且 $Q=\alpha_1P_1+\alpha_2P_2+\cdots+\alpha_nP_n=P_1+\alpha_2(P_2-P_1)+\cdots+\alpha_n(P_n-P_1)$。这意味着 $\mathbf{v}=\alpha_2(P_2-P_1)+\cdots+\alpha_n(P_n-P_1)$。因此，集合 $P_2-P_1,\cdots,P_n-P_1$ 可以生成相关的向量空间。假设这个集合是线性相关的，则仿射空间中相应的点集 $P_1,\cdots,P_n$ 将是线性相关的。因此，它必定是无关的，并生成一个线性空间，所以它构成一个基。

A.3 点积和距离

到目前为止，我们所讨论的向量空间和仿射空间都纯粹是代数形式的。还没有涉及到任何度量的概念，比如距离和角度。但是，我们所生活的这个空间，以及我们讨论图形学时所在的空间，都具有距离和夹角的度量。在这一节，我们讨论 $\mathbf{R}^n$ 中的点积（内积），并探讨如何运用它来度量距离和角度。在距离和角度的度量中，很关键的一点是要发挥向量的作用，而不是点的：度量仿射空间中两个点之间的距离时，我们计算这两个点之间相差的向量，并求得它的长度。

A.3.1 $\mathbf{R}^n$ 中的点积

给出 $\mathbf{R}^n$ 中的两个向量

$$\begin{bmatrix}x_1\\\cdot\\\cdot\\\cdot\\x_n\end{bmatrix}\quad 和\quad \begin{bmatrix}y_1\\\cdot\\\cdot\\\cdot\\y_n\end{bmatrix}$$

1094 我们定义它们的点积或内积为 $x_1y_1+\cdots+x_ny_n$。向量**v**和**w**的点积通常表示为**v** · **w**。

平面上的点(x, y)到原点（0, 0）的距离是$\sqrt{x^2+y^2}$。一般地，n维空间中从点（$x_1,\cdots,x_n$）到原点的距离是$\sqrt{x_1^2+\cdots+x_n^2}$。如果我们设向量**v**为

$$\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

则它到原点的距离就是$\sqrt{\mathbf{v}\cdot\mathbf{v}}$。这就是我们对$\mathbf{R}^n$中向量长度的定义。我们将此长度表示为$\|\mathbf{v}\|$。在标准的$n$维空间中，两个点之间的距离的定义是类似的：$P$和$Q$之间的距离就是向量$Q-P$的长度。

习题 点

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \quad 和 \quad \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$$

都在标准仿射平面上，同时也在三维空间$\mathbf{R}^3$中。在$\mathbf{R}^3$中，其中的每一个点到原点的距离是什么？它们之间的距离是什么？它们中的每一个点到标准仿射平面上的点

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

的距离是什么？两个向量

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \quad 和 \quad \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$$

的点积是什么？

解答 到原点的这两个距离分别是$\sqrt{11}$和$\sqrt{30}$。这两个点之间的距离是$\sqrt{5}$。到

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

的距离分别是$\sqrt{10}$和$\sqrt{29}$。点积是18。注意，对仿射空间中的两个点求点积是没有意义的，因为点积只是为向量定义的。 □

A.3.2 点积的性质

点积有几个很好的性质。首先，它是对称的：$\mathbf{v}\cdot\mathbf{w}=\mathbf{w}\cdot\mathbf{v}$。第二，它不会退化：只有当
1095 $\mathbf{v}=\mathbf{0}$时，$\mathbf{v}\cdot\mathbf{v}=0$。第三，它满足双线性关系：$\mathbf{v}\cdot(\mathbf{u}+\alpha\mathbf{w})=\mathbf{v}\cdot\mathbf{u}+\alpha(\mathbf{v}\cdot\mathbf{w})$。

点积可以用来生成长度为1的向量（这称为对向量进行规格化）。对向量**v**进行规格化，只需要简单地计算$\mathbf{v}'=\mathbf{v}/\|\mathbf{v}\|$。这样，所得的向量的长度是1，并被称为单位向量。

习题 向量

$$\begin{bmatrix} 4 \\ 3 \\ 0 \end{bmatrix}$$

的长度是多少？对此向量进行规格化后的结果是怎样的？考察标准仿射平面上的点

$$P=\begin{bmatrix}1\\1\\1\end{bmatrix} \quad 和 \quad Q=\begin{bmatrix}2\\3\\1\end{bmatrix}$$

与从P到Q的方向一致的单位向量是什么？

解答 向量的长度是5。规格化的向量是

$$\begin{bmatrix}4/5\\3/5\\0\end{bmatrix}$$

从P到Q的单位方向向量是

$$\begin{bmatrix}1/\sqrt{5}\\2/\sqrt{5}\\0\end{bmatrix}$$

注意最后的分量是0。

点积也能用来度量角度（从数学的角度看，就是定义角度）。向量v和w之间的夹角是

$$\cos^{-1}\left(\frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\,\|\mathbf{w}\|}\right)$$

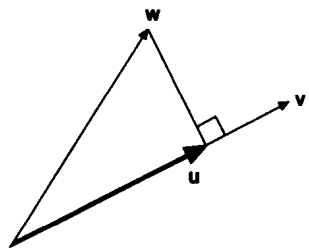
注意，如果v和w是单位向量，则上式中的除法就不必要了。

如果我们有一个单位向量v和另一个向量w，并将w垂直地投影到v上，如图A-5所示，所得结果是u，那么u的长度应当是w的长度与cos(θ)相乘的结果，此处的θ是v和w之间的夹角。这也就是说

$$\begin{aligned}\|\mathbf{u}\| &= \|\mathbf{w}\|\cos(\theta)\\ &= \|\mathbf{w}\|\left(\frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\,\|\mathbf{w}\|}\right)\\ &= \mathbf{v}\cdot\mathbf{w}\end{aligned}$$

因为v的长度是1。这样，就对点积给出了一个新的诠释：假设v是单位向量，则v和w的点积就是w在v上投影的长度。

1096



图A-5 w投影到单位向量v上的结果是向量u，该向量的长度是||w||乘以v和w之间夹角的余弦

习题 如果v和w都是单位向量，证明：v在w上的投影和w在v上的投影具有一样的长度。 □

解答 这两者都能表示成v · w，所以它们是一样的。

因为，在$\theta=90°$或$270°$等情况下，$\cos\theta=0$，所以，我们可以用向量的点积来判定它们是否垂直。当v · w = 0时，向量v和w肯定是垂直的。

A.3.3 点积的应用

因为点积可以用来度量长度，因此可以用它们来生成一些简单的方程。例如，在仿射空间中如果有一个点P，那么我们可以很容易地写出以P为圆心且半径为r的圆的方程，只需求到P的距离是r的所有点Q即可。于是，此方程为

$$\| Q - P \| = r$$

我们可以将它改写为

$$\sqrt{(Q - P) \cdot (Q - P)} = r$$

或

$$(Q - P) \cdot (Q - P) = r^2$$

在标准三维仿射空间中，过点P并垂直向量$\mathbf{v}$的平面方程也是容易表达的。该平面上的任一点Q都可以进行如下的表达：它到P的差的向量$Q - P$是垂直于向量$\mathbf{v}$的。所以，该方程即为

$$(Q - P) \cdot \mathbf{v} = 0$$

习题 假设P和Q是标准仿射平面上的点；P是点

$$\begin{bmatrix} a \\ b \\ 1 \end{bmatrix}$$

1097 而Q是不确定的点

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

那么，对于以P点为中心、r为半径的圆，在坐标上，它的方程是怎样的？

解答 它是 $(x-a)^2 + (y-b)^2 = r^2$，这是我们在高中代数中所熟悉的内容。

习题 如果P是标准三维仿射空间中的点

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

而$\mathbf{v}$是向量

$$\begin{bmatrix} 2 \\ 2 \\ 3 \\ 0 \end{bmatrix}$$

那么穿过P而垂直$\mathbf{v}$的平面的方程是什么？

解答 设待定的点是

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

那么，方程就是 $2(x-1) + 2(y-2) + 3(z-3) = 0$。

一般地，穿过点

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

而垂直向量

$$\begin{bmatrix} A \\ B \\ C \\ 0 \end{bmatrix}$$

的平面的方程是$A(x-x_0)+B(y-y_0)+C(z-z_0)=0$。这可以改写为$Ax+By+Cz=Ax_0+By_0+Cz_0$。 1098
如果我们是在$\mathbf{R}^3$中操作（而不是标准三维仿射空间），该方程的含义就只是说：

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} \quad 和 \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

的点积必须等同于

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} \quad 和 \quad \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

的点积。这个常量（上面方程右边的部分）完全是第二个点积的结果。如果

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

是一个单位向量，那么，该点积就表示了

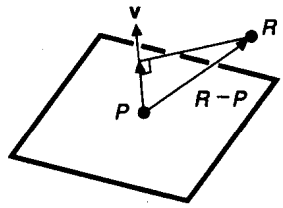
$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

在此单位向量上投影的长度，这也告诉我们在$\mathbf{R}^3$中该平面到原点的距离。

在平面的方程中，我们是通过确定平面的法向量来描述平面的。在某些情况下，这样处理是不合适的，因为法向量与一般的向量稍微有些不同，我们将在后面对此进行讨论。

A.3.4 距离公式

一个穿过P而垂直于$\mathbf{v}$的平面的方程是$(Q-P)\cdot\mathbf{v}=0$，那么一个点R到该平面的距离是多远呢？计算该距离的一种方法是求$(R-P)$到向量$\mathbf{v}$的投影（见图A-6）。该投影向量的长度就是R到该平面的距离。这个长度，也就是$(R-P)$和向量$\mathbf{v}$的点积，再除以$\mathbf{v}$的长度。这样，在考察由方程$Ax+By+Cz+D=0$定义的平面时，从点(r, s, t)到此平面的距离就是$(Ar+Bs+Ct+D)/\sqrt{A^2+B^2+C^2}$。注意，除数中的平方根就是平面的法向的长度。因此，如果一个平面是用一个单位法向量来定义的，就不必做除法。 1099



图A-6 从点R到平面的距离可以用R在平面的法方向上的投影来度量

习题 假设在仿射空间中有一条参数形式的直线，即所有形式为$P(t)=P_0+t\mathbf{v}$的点的集合，另有一个不在直线上的点R。从点R到此直线的距离是多少？

解答 一个点到一条直线的距离是这样定义的：该点到此直线上所有点的距离中最短的距离。当连接R与此线上的点的直线垂直于此直线，也就是说，当$(R-P(t))\cdot\mathbf{v}=0$时，就得到了此最短的距离。对此进行扩展，我们得到

$$(R - P_0 - t\mathbf{v}) \cdot \mathbf{v} = 0$$

$$(R - P_0) \cdot \mathbf{v} = t\mathbf{v} \cdot \mathbf{v}$$

$$t = \frac{(R - P_0) \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}}$$

所以，这就是距离达到最小时的t值。把它代入直线的方程中，我们得到最靠近R的点是

$$P_0 + \frac{(R - P_0) \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v}$$

在求从R到该点的距离时，我们只需要用R减去该点以得到一个向量，然后再计算该向量的长度即可。当$\mathbf{v} = \mathbf{0}$时，此公式无效。当然，此时$P(t) = P_0 + t\mathbf{v}$也不能定义一条直线。

A.3.5 求交的公式

在标准仿射空间中，假设我们有一个圆$(X - P) \cdot (X - P) = r^2$和一条线$S(t) = Q + t\mathbf{v}$。那么，这两者相交的点是什么？显然，交点一定对应满足$S(t)$的某个t值，并且，它也必须满足圆的方程，所以，我们必须求解方程

$$(S(t) - P) \cdot (S(t) - P) = r^2$$

1100 进行代数运算就可以将此表达式简化为

$$t^2 (\mathbf{v} \cdot \mathbf{v}) + t (2\,\mathbf{v} \cdot (Q - P)) + ((Q - P) \cdot (Q - P) - r^2) = 0$$

这是一个关于t的二次方程。求解这个二次方程，得到t值，然后，将此t值代入$S(t)$的公式中，就可以求得实际的交点。注意，这与坐标的个数无关。如果我们考察标准的三维仿射空间中由$(X - P) \cdot (X - P) = r^2$定义的点的集合（这是关于一个球的方程），对它运用同样的求解操作，就会得到直线与球的两个交点。

习题 在标准的仿射平面上，对于过点

$$\begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}$$

且方向为

$$\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

的直线，求它与中心在

$$\begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}$$

而半径为5的圆的交点。

解答 求得t_1和t_2分别为$(-2 \pm 2\sqrt{31})/5$。于是，交点为

$$\begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix} + t_i \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

其中$i = 1, 2$。

在标准三维仿射空间中，假设有一条直线和一个平面。怎样求它们的交点？如果平面的方

程是点-法向形式的，如$(X-P)\cdot\mathbf{v}=0$，而直线是参数形式的，如$Q(t)=Q+t\mathbf{w}$，我们可以简单地将X替换为$Q(t)$并求解

$$(Q+t\mathbf{w}-P)\cdot\mathbf{v}=0$$

解此方程，得到

$$t=\frac{(P-Q)\cdot\mathbf{v}}{\mathbf{w}\cdot\mathbf{v}}$$

以及交点的位置

$$Q+\frac{(P-Q)\cdot\mathbf{v}}{\mathbf{w}\cdot\mathbf{v}}\mathbf{w}$$

1101

这种求交点的方法是常用的。在标准三维仿射空间中，如果有一个方程为$F(x,y,z,1)=0$的面，我们可以将点$P+t\mathbf{v}$替换方程中的参量$(x,y,z,1)$，由此得到一个关于单个变量t的方程。对t求解，就可以得到光线上位于交点处的点所相关的参数值。将此t值代入$P+t\mathbf{v}$中，就得到实际交点的位置。

一般地，一个隐式定义的面（例如，在标准仿射三维空间中，以$F(x,y,z,1)=0$形式的方程所定义的面）在各个点$(x,y,z,1)$处都会有一个面的法向；该向量的坐标是F在该点处的偏导数。（平面中相应的情况已经在第3章关于椭圆的扫描转换中讨论过。）于是，法向量是

$$\begin{bmatrix}\frac{\partial F}{\partial x}(x,y,z,1)\\ \frac{\partial F}{\partial y}(x,y,z,1)\\ \frac{\partial F}{\partial z}(x,y,z,1)\\ 0\end{bmatrix}$$

A.3.6 正交基

如果两个向量$\mathbf{u}$和$\mathbf{v}$满足$\mathbf{u}\cdot\mathbf{v}=0$，则称它们为*正交的*。如果$\mathbf{B}=\{\mathbf{b}_1,\cdots,\mathbf{b}_n\}$是向量空间的一个基，并且每个$\mathbf{b}_i$都是一个单位向量，基中的任何两个向量都是正交的，这个基就称为*正交基*。对这些条件，我们可以很简单地说：如果$i\neq j$时，$\mathbf{b}_i\cdot\mathbf{b}_j=0$；而$i=j$时，$\mathbf{b}_i\cdot\mathbf{b}_j=1$，则$\mathbf{B}$是正交基。

正交基有一些别的基所没有的优点。例如，如果$\mathbf{B}$是正交基，当我们想把一个向量$\mathbf{v}$表示成$\mathbf{B}$中向量的线性组合时，即$\mathbf{v}=\alpha_1\mathbf{b}_1+\cdots+\alpha_n\mathbf{b}_n$，我们可以很容易地找到$\alpha_i$的值，它就是$\mathbf{v}\cdot\mathbf{b}_i$。

习题 在$\mathbf{R}^n$中，证明标准基$\mathbf{E}=\{\mathbf{e}_1,\cdots,\mathbf{e}_n\}$（其中，$\mathbf{e}_i$的分量中，只有第$i$个分量是1，其余的均为0）是一个正交基。证明向量

$$\begin{bmatrix}1/\sqrt{5}\\ 2/\sqrt{5}\end{bmatrix}\quad\text{和}\quad\begin{bmatrix}-2/\sqrt{5}\\ 1/\sqrt{5}\end{bmatrix}$$

构成了$\mathbf{R}^2$中的一个正交基。在这个基下，向量$\begin{bmatrix}3\\4\end{bmatrix}$的坐标是什么？

解答 前两个问题只需直接计算。后一个问题是，其坐标是$11/\sqrt{5}$和$-2/\sqrt{5}$。 □

正因为有这个特性，一个基通常要被转换成正交基。这可以运用Gram-Schmidt过程来实现。

1102

这一过程的基本思想是依次处理每个向量，使它与已经处理过的向量都正交，然后再进行规格化。如果我们从一个基$\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$开始，这一过程的步骤如下：

令 $\mathbf{v}_1' = \mathbf{v}_1$（在这一步，还没有任何向量被处理，所以选择任何向量都是无所谓的）。

$\mathbf{w}_1 = \mathbf{v}_1' / \| \mathbf{v}_1' \|$.

$\mathbf{v}_2' = \mathbf{v}_2 - (\mathbf{v}_2 \cdot \mathbf{w}_1)\mathbf{w}_1$（此向量与$\mathbf{w}_1$正交）

$\mathbf{w}_2 = \mathbf{v}_2' / \| \mathbf{v}_2' \|$.

$\mathbf{v}_3' = \mathbf{v}_3 - (\mathbf{v}_3 \cdot \mathbf{w}_1)\mathbf{w}_1 - (\mathbf{v}_3 \cdot \mathbf{w}_2)\mathbf{w}_2$.

$\mathbf{w}_3 = \mathbf{v}_3' / \| \mathbf{v}_3' \|$.

所得的向量$\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$是一个正交基。如果向量的数目很大，这个过程可类似地进行。对于三个向量的情况，最后一步可以简化，参见习题A.7。

A.4 矩阵

矩阵就是将数排列成矩形样子的一个数组。其中的每一个元素都有两个序号，按照惯例，第一个序号是关于行的，而第二个序号是关于列的。按照数学上的习惯，这些序号都从1开始记数；但有一些程序语言是将序号从0开始记数。对于用那些程序语言的程序员而言，他们要将所有序号移动1位。因此，如果**A**是一个矩阵，那么$a_{3,2}$是指第3行第2列的元素。当运用符号形式的序号时，比如说a_{ij}，则序号之间的逗号被省略。

在$\mathbf{R}^n$中我们写成如下形式的元素

$$\begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

可以认为是一个$n \times 1$的矩阵。

A.4.1 矩阵乘法

矩阵的乘法是按照下列的原则进行的：如果**A**是一个$n \times k$的矩阵，其元素为a_{ij}，而**B**是一个$k \times p$的矩阵，元素是b_{ij}，那么**AB**就是一个$n \times p$的矩阵，元素是c_{ij}，且 $c_{ij} = \sum_{s=1}^{k} a_{is} b_{sj}$ ⊖。如果将**B**的列都当成单独的向量，$\mathbf{B}_1, \cdots, \mathbf{B}_p$，而**A**的行也当成单独的向量$\mathbf{A}_1, \cdots, \mathbf{A}_k$（旋转90°后就恢复成水平的了），那么，我们看到$c_{ij}$就是$\mathbf{A}_i \cdot \mathbf{B}_j$。除了交换律外，矩阵乘法保持了一般乘法的其他特点。一般而言，**AB**和**BA**是不同的。当然，乘法满足相对于加法的分配律：$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$。对于乘法，有一个恒定的元素，称为单位矩阵**I**，它是一个正方形的矩阵，除了对角线上的元素为1之外，其余的元素均为0（例如，对于元素δ_{ij}，除$i = j$时$\delta_{ii} = 1$之外，其他项$\delta_{ij} = 0$）。

A.4.2 行列式

一个方形矩阵的行列式就是一个数，它能告诉我们关于矩阵的许多情况。当且仅当矩阵的
1103 行列式是一个非零数的时候，这个矩阵的各个列才是线性无关的。每一个$n \times n$的矩阵都是表示一个从$\mathbf{R}^n$到$\mathbf{R}^n$的变换，并且该矩阵的行列式能告诉我们这个变换能带来怎样的大小变化（例如，它可以告诉我们单位立方体在变换后是放大了还是缩小了）。

行列式的计算有点复杂，因为它是递归定义的。2×2的矩阵 $\begin{vmatrix} a & c \\ b & d \end{vmatrix}$ 的行列式就是$ad - bc$。

⊖ 原书为 $c_{ij} = \sum_{s=0}^{k} a_{is} b_{sj}$，原书有误。——译者注

一个$n \times n$的矩阵的行列式是在比它小的矩阵的行列式的基础上进行计算的。假设A_{1i}是将$n \times n$的矩阵A的第一行和第i列的元素去掉以后所得的$(n-1) \times (n-1)$矩阵的行列式，那么，行列式A是由下面的式子定义的：

$$\det \mathbf{A} = \sum_{i=1}^{n} (-1)^{i+1} a_{1i} A_{1i}$$

计算行列式的另外一种方法是*高斯消元法*。高斯消元法是通过一系列有关行的操作来完成的。在一个矩阵中，有3种关于行的操作：(1)任意两行进行交换；(2)以一个非零的标量乘以一行元素；(3)将行i加到行j上去（行i保持不变，而第j行被替换成(行j) + α(行i)。用高斯消元法简化一个$n \times n$的矩阵**A**的算法很简单，即通过行的交换以及缩放操作，可以使得$a_{11} = 1$。对于每一个不等于1的j，用a_{j1}乘以第1行，并从j行减去它，就可以使a_{j1}变成0。随后，对第2行以及后面的行（如果需要的话）进行交换、缩放等操作，可以使$a_{22} = 1$。对于每个不等于2的j，用a_{j2}乘以第2行，并从j行减去它。反复进行这种操作，直至该矩阵变成单位矩阵。

在这个过程中，可能无法使某些$a_{ii} = 1$（例如，当整个第i列的元素都为0时的情况）；此时，行列式的值就为0。否则，行列式的计算就是将高斯消元法中运用(2)的行操作时的所有缩放系数都相乘，并将乘积再乘以$(-1)^k$，其中，k是在高斯消元法中交换行的次数。

在$\mathbf{R}^3$中，行列式有一种特殊的应用：*叉积*。两个向量

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \text{和} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

的叉积是运用矩阵的行列式进行计算的，即

$$\begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

其中，字母**i**, **j**, **k**是符号变量。计算的结果是关于变量**i**, **j**, **k**的一个线性组合；此时，这些变量可以分别替换成向量$\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$。于是，所得的结果为

$$\begin{bmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{bmatrix}$$

1104

表示成$\mathbf{v} \times \mathbf{w}$。该向量的一个特点是垂直于由**v**和**w**所定义的平面，而它的长度是$\|\mathbf{v}\| \|\mathbf{w}\| |\sin\theta|$，其中，$\theta$是**v**和**w**之间的夹角。它的另一个特点是，由**v**、**w**和$\mathbf{v} \times \mathbf{w}$作为各个列而构成的矩阵的行列式永远是非负的。

这最后一个特点很有趣，它可以用来定义*定向*（orientation）。如果在将每个基中的各个向量作为一列形成一个矩阵时，这两个基所相关的矩阵的行列式具有相同的符号，则$\mathbf{R}^n$中的两个基被称为是具有相同的定向。如果一个基与标准基具有相同的定向，则它被称为是*正向的*；否则，它就是*负向的*。

习题 在$\mathbf{R}^4$中，证明基$\{\mathbf{e}_2, \mathbf{e}_1, \mathbf{e}_3, \mathbf{e}_4\}$是负向的。

解答 相关矩阵的行列式是 -1。 □

习题 假设两个平面的方程分别是$(X - P) \cdot \mathbf{v} = 0$和$(X - Q) \cdot \mathbf{w} = 0$。这两个平面的交线的方向向量是什么？

解答 因为这条相交线位于这两个平面上，它的方向向量一定与这两个平面的法向都垂直。这样的一个向量就是叉乘$\mathbf{v} \times \mathbf{w}$。如果**v**和**w**是平行的，那么这两个面要么是重叠的，要么就根

本不会相交；所以，在$\mathbf{v} \times \mathbf{w} = 0$的情况下，是个退化问题。 □

A.4.3 矩阵的转置

一个$n \times k$的矩阵相对于它的对角线（从左上到右下）进行对称的换位，就得到了一个$k \times n$的矩阵。第一个矩阵中的元素排列是a_{ij} $(i = 1, \cdots, n;\ j = 1, \cdots, k)$，所得的矩阵中的元素排列是$b_{ij}$ $(i = 1, \cdots, k;\ j = 1, \cdots, n)$，且$b_{ij} = a_{ji}$。这个新的矩阵称为原来矩阵的*转置*。矩阵**A**的转置写为$\mathbf{A}^t$。如果我们将$\mathbf{R}^n$中的一个向量当成是一个$n \times 1$的矩阵，那么它的转置就是一个$1 \times n$的矩阵（有时称为行向量）。运用转置，我们可以对$\mathbf{R}^n$中的点积给出一种新的表达，即$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^t\mathbf{v}$。

习题 先计算一个例子作为验证，然后再做一般性的证明，试证明：如果**A**是一个$n \times k$的矩阵，而**B**是一个$k \times p$的矩阵，那么$(\mathbf{AB})^t = \mathbf{B}^t\mathbf{A}^t$。

解答 此问题由读者自己解决。

A.4.4 矩阵的逆

矩阵乘法与一般的乘法不同：一个矩阵可能没有相应的逆。事实上，逆只是对方形矩阵定义的，并且也不是所有的方形矩阵都有逆。只有那些其行列式不为零的方形矩阵才有逆。

如果**A**和**B**都是$n \times n$的矩阵，并且$\mathbf{AB} = \mathbf{BA} = \mathbf{I}$，其中**I**是$n \times n$的单位矩阵，那么，**B**称为是**A**的逆，并写为$\mathbf{A}^{-1}$。对于元素是实数的$n \times n$的矩阵，只要$\mathbf{AB} = \mathbf{I}$或者$\mathbf{BA} = \mathbf{I}$成立（只要有一个
1105 成立，那么另一个也成立），就足以证明它们是可逆的。

如果我们有一个$n \times n$的矩阵，那么有两种基本的方法来求它的逆：高斯消元法和克莱姆法则。对于大于3×3的矩阵来说，最好是用高斯消元法。

运用高斯消元法求一个矩阵的逆时，先写出矩阵**A**和单位矩阵。当你运用行操作将**A**简化成单位矩阵时，你也对单位矩阵进行相同的行操作。当**A**变成单位矩阵时，原来的单位矩阵就变成了$\mathbf{A}^{-1}$。在运用高斯消元法时，如果有些对角线上的元素不能变为1，那么，正如我们已提到过的，它的行列式是0，这样，它的逆就不存在。对该方法可以进行多方面的改进。比较好的参考书是[PRESS88]，它包含了一些可运行的程序。

计算逆的另一种方法称为*克莱姆法则*。它是以显式的方式求逆，但它要计算很多行列式。下面介绍它是如何做的。

对于一个元素为a_{ij}的$n \times n$的矩阵**A**，我们建立一个新的矩阵$\mathbf{A}'$，它的元素是A_{ij}。计算A_{ij}时，我们从矩阵**A**中去掉第i行和第j列的元素，然后对余下的$(n-1) \times (n-1)$矩阵的行列式进行计算，最后将此行列式的值乘以$(-1)^{i+j}$，就得到A_{ij}的值。一旦计算了$\mathbf{A}'$，**A**的逆就是$(1/\det \mathbf{A})(\mathbf{A}')^t$。

因为要计算大量的行列式，克莱姆法则一般不宜处理大的矩阵。但是，对于2×2的情况，该方法是很有用的。它可以直接给出矩阵的逆如下：

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

有关矩阵的逆，还有一种特殊的情况值得讨论。假设**U**是一个矩阵，它各列的向量形成一个正交基。也就是说，对于所有的i和j，$\mathbf{u}_i \cdot \mathbf{u}_j = \delta_{ij}$。在计算$\mathbf{U}^t\mathbf{U}$时，会发生什么情况呢？我们已经提到乘积的第$ij$项是第一个因子的第$i$行元素与第二个因子的第$j$列元素的点积。由于它们分别是$\mathbf{u}_i$和$\mathbf{u}_j$，所以它们的点积是$\delta_{ij}$。这就是说$\mathbf{U}^t\mathbf{U} = \mathbf{I}$，并且$\mathbf{U}^{-1} = \mathbf{U}^t$。顺便提一下，这也就是说，$\mathbf{U}^t$也能形成一个正交基。

A.5 线性变换和仿射变换

*线性变换*就是从一个向量空间到另一个向量空间的一个保持线性组合特性的映射。更精确

地说，它就是一个具有下列特点的映射$\mathbf{T}$，即$\mathbf{T}(\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \cdots + \alpha_n\mathbf{v}_n) = \alpha_1\mathbf{T}(\mathbf{v}_1) + \alpha_2\mathbf{T}(\mathbf{v}_2) + \cdots + \alpha_n\mathbf{T}(\mathbf{v}_n)$。关于线性变换，我们在第5章中已进行了详细的讨论。

*仿射变换*就是从一个仿射空间到另一个仿射空间的保持仿射组合的一个映射。精确地说，它就是一个具有下列特点的映射$\mathbf{T}$，即$\mathbf{T}(P + \alpha(Q - P)) = \mathbf{T}(P) + \alpha(\mathbf{T}(Q) - \mathbf{T}(P))$。自然地，$\mathbf{T}$可将此映射扩展到相关的向量空间。即定义$\mathbf{T}(\mathbf{v})$为$\mathbf{T}(P) - \mathbf{T}(Q)$，其中$P$和$Q$是任意的两个点，且$Q - P = \mathbf{v}$。仿射变换包括平移、旋转、缩放和裁剪变换。注意，在第5章中定义的变换既是仿射变换也是线性变换。它们是从$\mathbf{R}^4$到$\mathbf{R}^4$的线性变换，但是它们操作的形式是从标准三维仿射空间变换到自身（$\mathbf{R}^4$中的点的最后坐标是1），所以，它们也就刻画了在此仿射空间上的仿射变换。 1106

A.5.1 关于$\mathbf{R}^n$上变换的矩阵

在$\mathbf{R}^n$中，假设我们有n个无关的向量$\mathbf{b}_1,\cdots,\mathbf{b}_n$。我们希望找到一个线性变换将这些向量变换到向量$\mathbf{a}_1,\cdots,\mathbf{a}_n$。（之所以用这个临时的名称表达，是因为那些$\mathbf{b}_i$是无关的，形成了一个基。）对此，我们怎么做呢？最简单的方法是将$\mathbf{R}^n$上的一个线性变换用一个矩阵来表示。也就是说，我们将找一个$n \times n$的矩阵$\mathbf{A}$，使得对$\mathbf{R}^n$中的所有$\mathbf{v}$有$\mathbf{T}(\mathbf{v}) = \mathbf{A}\mathbf{v}$。

首先，我们讨论一个简单的问题。我们找一个变换矩阵，它可以将标准基向量$\mathbf{e}_1,\cdots,\mathbf{e}_n$变换到任意一组向量$\mathbf{v}_1,\cdots,\mathbf{v}_n$。

假设$\mathbf{Q}$是一个$n \times n$的矩阵，元素是q_{ij}，并将它乘以$\mathbf{e}_j$。如果我们让$\mathbf{r} = \mathbf{Q}\mathbf{e}_j$，那么$\mathbf{r}_i = \mathbf{q}_{ij}$。这就是说，用第$j$个标准基向量乘以一个矩阵，就是将该矩阵的第$j$列提取出来。基于此，我们可以找到这样的矩阵：它可以将标准基向量变换到$\mathbf{v}_1,\cdots,\mathbf{v}_n$。就是用向量$\mathbf{v}_1,\cdots,\mathbf{v}_n$作为矩阵中的列。

习题 求一个矩阵，它将$\mathbf{R}^2$中标准基变换到向量$\begin{bmatrix}1\\2\end{bmatrix}$和$\begin{bmatrix}3\\3\end{bmatrix}$。

解答 $\begin{bmatrix}1 & 3\\2 & 3\end{bmatrix}$。 □

现在，我们讨论本节最初的问题，即找一个变换使得能把那些$\mathbf{b}_i$变换到$\mathbf{a}_i$。对此，我们把上面对简单问题的处理方法运用两次。首先，我们找到一个矩阵$\mathbf{B}$（它的列就是那些$\mathbf{b}_i$）将标准基变换到那些$\mathbf{b}_i$；然后，我们找到一个矩阵$\mathbf{A}$将标准基变换到那些$\mathbf{a}_i$。矩阵$\mathbf{B}^{-1}$的作用与$\mathbf{B}$的作用刚好相反，它把那些$\mathbf{b}_i$变换到标准基，所以矩阵$\mathbf{A}\mathbf{B}^{-1}$就是我们所要的结果。它是将$\mathbf{b}_i$变换到$\mathbf{a}_i$的矩阵。

习题 找一个矩阵，可以将$\begin{bmatrix}1\\2\end{bmatrix}$和$\begin{bmatrix}2\\5\end{bmatrix}$分别变换到$\begin{bmatrix}1\\1\end{bmatrix}$和$\begin{bmatrix}3\\2\end{bmatrix}$。

解答 将标准基变换到第一组向量的矩阵是$\begin{bmatrix}1 & 2\\2 & 5\end{bmatrix}$，而将标准基变换到第二组向量的矩阵是$\begin{bmatrix}1 & 3\\1 & 2\end{bmatrix}$。所以，结果矩阵是$\mathbf{T}(\mathbf{v}) = \mathbf{Q}\mathbf{v}$，其中

$$\mathbf{Q} = \begin{bmatrix}1 & 3\\1 & 2\end{bmatrix}\begin{bmatrix}1 & 2\\2 & 5\end{bmatrix}^{-1} = \begin{bmatrix}-1 & 1\\1 & 0\end{bmatrix}$$

A.5.2 点和法向量的变换

当我们将一个矩阵线性变换作用于标准的n维仿射空间的点时，点之间的差（也就是仿射空间中的向量）是怎样变换的呢？假设变换定义为$\mathbf{T}(P) = \mathbf{A}P$，并进一步假设这个变换是将仿射平面变换到自身（也就是说，在变换后不必做齐次化处理——这相当于除了右下角的元素为1之外，$\mathbf{A}$最后一行中其余的元素都是0）。那么，$\mathbf{T}(Q - P) = \mathbf{A}(Q - P)$。但是，$Q - P$最后的分量是0（因为，$P$和$Q$在那个位置都是1）。因此，$\mathbf{A}$中的最后一列对于变换结果没有任何作用。所以，我们将$\mathbf{A}'$定义成与$\mathbf{A}$一样，只是它的最后一列中的元素除了最后一个为1之外都被替换成0。

1107 这个矩阵A'可以用来变换仿射空间中的向量。

我们在前面曾提到用法向量来定义平面是不合适的，现在，我们来讨论这是因为什么。假设有一个平面，其方程为$(X-P)\cdot\mathbf{v}=0$。当我们用A对此平面进行变换时，将得到一个包含点AP的新平面，所以，该平面的方程将是相对于某个向量w的方程$(Y-AP)\cdot\mathbf{w}=0$。我们想使满足这个方程的点具有AX的形式（其中X是原来平面上的点）满足后一个方程。这样，我们就要找具有如下性质的向量：$(X-P)\cdot\mathbf{v}=0$时，$(AX-AP)\cdot\mathbf{w}=0$。换一句话说，我们就是要在$(X-P)\cdot\mathbf{v}=0$时有$(AX-AP)^t\mathbf{w}=0$。

利用置换运算，我们可将上式化为：当$(X-P)\cdot\mathbf{v}=0$时，有$(X-P)\,A^t\mathbf{w}=0$

显然，当$A^t\mathbf{w}=\mathbf{v}$时，即$\mathbf{w}=(A^t)^{-1}\mathbf{v}$时，这个方程是成立的。因此，$(A^t)^{-1}\mathbf{v}$就是变换后的平面的法向量。当A是正交矩阵时（比如在旋转变换的情况下），我们知道$(A^t)^{-1}=A$，所以，法向量就像点一样进行变换（但是没有平移操作，因为向量最后的分量是0）。但是，对于一般的矩阵而言，这并不总是成立的。对于A的转置的逆的计算可以简化为计算A'的转置的逆，因为它对向量的作用和A对向量的作用是一样的。关于A'的计算要容易一些，因为从效果上看A'是一个较小的矩阵（它的最后一行和最后一列和单位矩阵是一样的）。⊖

计算一个矩阵的逆是比较困难的，并且，如果用克莱姆法则，将要分成许多有关行列式的计算。因为变换后的法向量可能不再是一个单位向量，它需要再进行规格化，所以省去除法操作不会有什么影响。为此，人们有时用余因子矩阵来变换法向量。关于这个矩阵的计算是这样的：从A矩阵中删去第i行和第j列所得的矩阵的行列式，再乘以$(-1)^{i+j}$就是该矩阵第ij项元素。

A.6 特征值和特征向量

变换T的一个特征向量是满足以下条件的一个向量v，即$\mathbf{T}(\mathbf{v})$对v的作用相当于一个纯量乘法。如果$\mathbf{T}(\mathbf{v})=\lambda\mathbf{v}$，那么$\lambda$就称为是与向量v相关的特征值。理论上求矩阵（至少是对一个矩阵变换$\mathbf{T}(\mathbf{v})=A\mathbf{v}$而言）的特征值的方法，是令$\mathbf{B}=\mathbf{A}-x\mathbf{I}$，其中I是单位矩阵，而$x$是一个变量。那么，B的行列式就是一个关于$x$的多项式$p(x)$，$p$的根就是特征值。如果$\lambda$是一个实数特征值，那么，对某个向量v而言，一定有$\mathbf{T}(\mathbf{v})=\lambda\mathbf{v}$。通过重新整理，我们得到$A\mathbf{v}-\lambda\mathbf{v}=0$，或者$(\mathbf{A}-\lambda\mathbf{I})\mathbf{v}=0$。于是，对最后的方程求解，就得到了所有关于$\lambda$的特征向量。

1108 尽管这种方法在理论上是可行的，但实际上用处不大，特别是对大型的矩阵。代替这样算法的是使用基于迭代的变换方法。其中，主要的有Gauss-Seidel迭代法，有关该方法的详细内容，请参见[PRES88]。

在本节的最后，我们讨论两个特别有趣的习题。

习题 在一个对称矩阵中（即$\mathbf{M}^t=\mathbf{M}$），证明：与不同的特征值相对应的特征向量是正交的。再证明：对于任何方形矩阵A，矩阵A^tA是对称的。

解答 假设$\mathbf{Mv}=\lambda\mathbf{v}$，$\mathbf{Mu}=\mu\mathbf{u}$。我们以两种方法来计算$\mathbf{u}^t\mathbf{Mv}$：

$$\mathbf{u}^t\mathbf{Mv}=\mathbf{u}^t\lambda\mathbf{v}=\lambda\,\mathbf{u}^t\mathbf{v}=\lambda\,(\mathbf{u}\cdot\mathbf{v})$$

但是

$$\mathbf{u}^t\mathbf{Mv}=(\mathbf{u}^t\mathbf{M}^t)\,\mathbf{v}=(\mathbf{Mu})^t\mathbf{v}=\mu\,\mathbf{u}^t\mathbf{v}=\mu\,(\mathbf{u}\cdot\mathbf{v})$$

⊖ 微分几何中，将法向量这样的向量称为余向量（covectors），因为这些向量是由相对于通常的向量（或者切向量）的一个点积关系来定义的。所有余向量组成的集合，有时称为余切空间，但是这个名称与三角函数没有任何关系。更复杂的情况，就称为张量，它包括切向量和余切向量。对它们进行变换的规则也相应地要复杂一些。

这样，$\lambda(\mathbf{u}\cdot\mathbf{v})=\mu(\mathbf{u}\cdot\mathbf{v})$；因此，$(\lambda-\mu)(\mathbf{u}\cdot\mathbf{v})=0$。因为，$\lambda$和$\mu$是不同的特征值，所以$(\lambda-\mu)\neq 0$。因此，$(\mathbf{u}\cdot\mathbf{v})=0$。

$(\mathbf{A}^t\mathbf{A})$的转置就是$\mathbf{A}^t(\mathbf{A}^t)^t$，由于转置矩阵的转置就是原来的矩阵，所以，结果就是$\mathbf{A}^t\mathbf{A}$。因此，矩阵$\mathbf{A}^t\mathbf{A}$是对称的。 □

习题 假设$\mathbf{T}(\mathbf{x})=\mathbf{A}\mathbf{x}$是$\mathbf{R}^2$上的一个线性变换。我们将它作用于单位圆上的所有点。所得的结果是一个中心在原点的椭圆。证明：椭圆的主轴与副轴的长度平方值的和等于$\mathbf{A}$的奇异值中的最大者与最小者之和，在此，$\mathbf{A}$的奇异值定义为$\mathbf{A}^t\mathbf{A}$的特征值。

解答 变换后的圆上的点具有形式$\mathbf{A}\mathbf{x}$，在此$\mathbf{x}\cdot\mathbf{x}=1$。如此变换后的一个点到坐标原点的距离的平方，就是$\mathbf{A}\mathbf{x}\cdot\mathbf{A}\mathbf{x}$，或者写为$\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x}$。设$\mathbf{u}$和$\mathbf{v}$是$\mathbf{A}^t\mathbf{A}$的两个单位特征向量，它们相应的特征值是$\lambda$和$\mu$。因为它们是正交的，它们可以形成$\mathbf{R}^2$的一个基。因此，我们可以将$\mathbf{x}$写成它们的一个线性组合：$\mathbf{x}=\cos\theta\,\mathbf{u}+\sin\theta\,\mathbf{v}$。现在，如果我们计算$\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x}$，就得到

$$\begin{aligned}\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x} &= (\cos\theta\,\mathbf{u}^t+\sin\theta\,\mathbf{v}^t)(\cos\theta\,\mathbf{A}^t\mathbf{A}\mathbf{u}+\sin\theta\,\mathbf{A}^t\mathbf{A}\mathbf{v})\\ &= (\cos\theta\,\mathbf{u}^t+\sin\theta\,\mathbf{v}^t)(\cos\theta\,\lambda\mathbf{u}+\sin\theta\,\mu v)\\ &= \lambda\cos^2\theta+\mu\sin^2\theta\end{aligned}$$

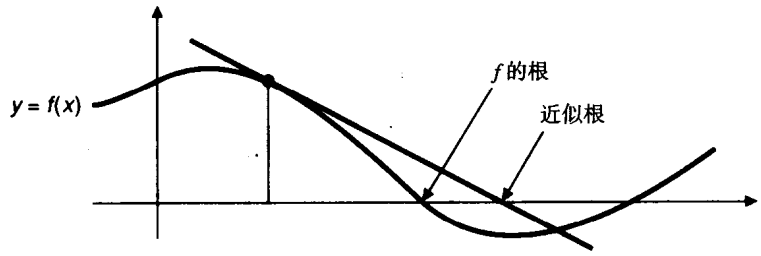
在θ等于90°的倍数，也就是当$\mathbf{x}$等于$\pm\mathbf{u}$或$\pm\mathbf{v}$的时候，这个函数达到它的极值。在这些点上的值恰是λ和μ。

A.7 求根的Newton-Raphson迭代法

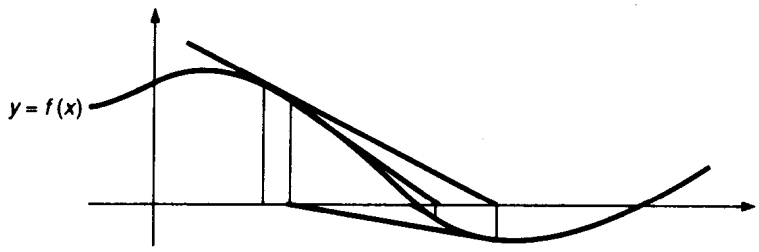
如果我们有一个从实数域映射到实数域的连续函数，记为f，并且已知$f(a)>0$和$f(b)<0$，那么，在a和b之间，一定有f的一个根。找到根的一种方法就是二分法：我们计算在$(a+b)/2$处f的值。如果它是正的，我们就在$(a+b)/2$和b之间继续找根；如果它是负的，我们就在a 1109
和$(a+b)/2$之间找；如果它是0，我们就找到了一个根。将此操作迭代地进行下去，直至f的值非常接近0时，就得到了f的一个根的很好的逼近值。

利用$(a,f(a))$和$(b,f(b))$之间的连线，我们可以对这种方法进行一点改进，即考察它在哪里与x轴相交，然后，用这个交点作为划分点。

如果f是可导的，我们还可以进行进一步的改进。我们可以计算f在某一点的值，以及它在那个点的导数。由此，我们可以得到在该点处与f相切的直线方程。如果f的图形与此切线的图形很接近，那么，切线与x轴相交的点，就是对f的一个根的很好的逼近（见图A-7）。如果它的逼近程度不够，我们可以用它作为一个起点再继续进行迭代过程（见图A-8）。



图A-7 如果一个函数的切线的图形与该函数的图形很接近，切线与x轴的交点将是该函数与x轴交点的一个很好的逼近结果



图A-8 图A-7描述的过程的迭代

如果初始点猜想为x_0，那么切线的方程是

$$y - f(x_0) = f'(x_0)(x - x_0)$$

当$y = 0$时，它就与x轴相交，也就是相交在下面的点处：

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

一般地，按照相应的公式，我们可以从点x_i求到下一个点x_{i+1}，重复此操作，直至求出根。这个过程就称为*牛顿法*或者Newton-Raphson迭代法。

习题 对函数$f(x) = x^2 - 2$用牛顿法，起始点为$x_0 = 1$。

解答 $x_0 = 1$，$x_1 = 1.5, x_2 = 1.4166\overline{6}, x_3 = 1.4142\cdots$等等。 □

这种方法在操作时，可能会有迭代点周期性循环的情况，此时，该方法就无效了。例如，可能会有$x_2 = x_0$，那么，该操作就会永远重复下去，而不能得到一个较好的逼近结果。比如说函数$f(x) = x^3 - 5x$，它有一个根$x = 0$，但如果起始点是在$x_0 = 1$，该迭代法就永远找不到这个根，因为它后续的点是$x_1 = -1$，$x_2 = 1$，并循环出现。

如果函数f比较理想，则该方法可以保证找到根。特别是，如果f在每一个地方的一阶导数都是正的，而二阶导数都是负的，则该方法肯定会收敛到一个根。

习题

A.1 从一个平面（用一个简单的模型）反射出来的光线，是根据“入射角与反射角相等”的原理进行的。如果平面的法向量是$\mathbf{n}$，从灯光来的光线的参数表达形式为$P + t\mathbf{v}$，那么反射光线的方向向量$\mathbf{u}$是怎样的？

答案 如果我们将$\mathbf{v}$表示成两个量的和（一个是$\mathbf{n}$的方向，而另一个是垂直于$\mathbf{n}$的），我们就能很容易地表达$\mathbf{u}$：除了沿$\mathbf{n}$方向中的分量是相反的，它与$\mathbf{v}$中其余的分量是一样的。$\mathbf{v}$中沿$\mathbf{n}$方向中的分量是$(\mathbf{v}\cdot\mathbf{n})\mathbf{n}/\|\mathbf{n}\|$，所以最后的结果是$\mathbf{u} = (\mathbf{v} - (\mathbf{v}\cdot\mathbf{n})\mathbf{n}/\|\mathbf{n}\|) - (\mathbf{v}\cdot\mathbf{n})\mathbf{n}/\|\mathbf{n}\| = \mathbf{v} - 2(\mathbf{v}\cdot\mathbf{n})\mathbf{n}/\|\mathbf{n}\|$。如果$\mathbf{n}$是一个单位向量，这就是$\mathbf{v} - 2(\mathbf{v}\cdot\mathbf{n})\mathbf{n}$。（注意，此处的光线与第16章中的光线方向相反）。

A.2 找一个从标准仿射平面到其自身的变换，它保持h坐标不变，但改变(x, y)坐标，并且，在任何平行xy面而高度为h的平面上，单位正方形$[0, 1] \times [0, 1]$将被变换到$[-1, 1] \times [-1, 1]$。在下面关于此问题的证明过程中，请指出哪些地方错了？

由于我们是在三维空间中进行讨论，所以，通过找3个基向量的变换结果，我们就可以确定这个变换了。显然，

$$\begin{bmatrix}1\\0\\0\end{bmatrix} \quad \text{变成} \quad \begin{bmatrix}1\\-1\\0\end{bmatrix}$$

并且

$$\begin{bmatrix}0\\1\\0\end{bmatrix} \quad \text{变成} \quad \begin{bmatrix}-1\\1\\0\end{bmatrix}$$

还有，

$$\begin{bmatrix}0\\0\\1\end{bmatrix} \quad \text{变成} \quad \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

所以，该矩阵一定是

$$\begin{bmatrix}1 & -1 & -1\\-1 & 1 & -1\\0 & 0 & 1\end{bmatrix}$$

简略答案 在上面的证明过程中，它假设这个映射是三维空间$\mathbf{R}^3$中的一个线性映射，但它实际上是一个仿射映射，并且有平移操作。

A.3 在$\mathbf{R}^3$中，有一系列的顶点$v[1],\cdots, v[n]$，它们表示成xyz的三元组形式，但它们的z坐标都是0，显然，它们在平面上定义了一个封闭的多边形（多边形的边是v_1v_2, $v_2v_3,\cdots$, $v_{n-1}v_n$, v_nv_1）。现在，将此多边形沿着z轴从$z=0$推进到$z=1$，并由此形成一个包含该多边形的多面体。假设这是在右手坐标系进行的，那么：

1110
≀
1111

a. 你怎样检测多边形$v[1],\cdots, v[n]$是逆时钟方向的（就是说，当你沿着多边形的边行进时，多边形的内部总是在你的左边）？

b. 给出一个算法，为上面所述的多面体生成多边形面。对于“面”，你可以使用长方形，也可以将它们剖分成三角形。这些多边形应该用一系列编序的顶点来表达。

c. 在(b)中给出的每个面都包含一系列顶点。假设你沿着初始多边形的边界行走，并且多边形的内部总在你的右方。你能否选择一个顶点顺序，保证你的头总是朝向多面体的外部？如果不是，请修改你给出的(b)的答案。

d. 生成的多面体的每条边都与两个面相连，所以在(c)的遍历过程中，每条边要遍历两次。这样的两次遍历是同一个方向吗？是否总是相反的方向？或者是并没有什么特别的方式？

A.4 假设$P=(x_0, y_0)$和$Q=(x_1, y_1)$是平面上的两个点，证明连接它们的直线的方程是$(y_1-y_0)x-(x_1-x_0)y=y_1x_0-x_1y_0$。这是个很好的公式，因为它对任意方向的直线（包括垂直线）都给出了一个一般的表达形式。

A.5 假设平面上有一个向量$\mathbf{v}=\begin{bmatrix}x\\y\end{bmatrix}$，证明$\mathbf{w}=\begin{bmatrix}-y\\x\end{bmatrix}$与它正交。有时，这称为在平面上的单个向量的叉积（对于三维空间中相应的情况也是类似的）。$\mathbf{R}^n$中$n-1$个向量的叉积也可类似地定义。

A.6 如果P、Q、R是标准的三维仿射空间中的3个点，那么

$$\frac{1}{2}\|(\mathbf{Q}-\mathbf{P})\times(\mathbf{R}-\mathbf{P})\|$$

是三角形ΔPQR的面积。在标准的三维仿射空间中，如果P、Q、R都位于xy平面上，那么

$$\frac{1}{2}\begin{bmatrix}0\\0\\1\end{bmatrix} \cdot ((\mathrm{Q}-\mathrm{P})\times(\mathrm{R}-\mathrm{P}))$$

给出了此三角形的*有向面积*——如果P、Q、R是按照逆时针顺序排列的，则此面积是一个正数，否则，就是负数。（在此，逆时针是指“在一个右手坐标系中，从z轴的正半轴上的一点看过去，点的排列顺序为逆时针方向”。）

a. 点$P=(0,0)$, $Q=(x_i, y_i)$，$R=(x_{i+1}, y_{i+1})$形成一个三角形，计算其有向面积。

答案 $1/2\ (x_i y_{i+1} - x_{i+1} y_i)$。

b. 假设我们在平面上有一个多边形，其点的序列是$v_1,\cdots, v_n$，$(v_n = v_1)$，并且，对每个i，$v_i = (x_i, y_i)$。解释为什么此多边形的有向面积是

$$\frac{1}{2}\sum_{i=1}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$

将此结果与等式(11-2)作比较。

A.7 在A.3.6节中介绍的针对三个向量的Gram-Schmidt过程可以稍微简化一点。在计算了$\mathbf{w}_1$和$\mathbf{w}_2$之后，我们就寻找垂直于这两个向量的第3个单位向量$\mathbf{w}_3$。此时只有两种可能的选择：$\mathbf{w}_3 = \pm \mathbf{w}_1 \times \mathbf{w}_2$。证明：Gram-Schmidt过程在此公式中选择的正负号就是$\mathbf{v}_3 \cdot (\mathbf{w}_1 \times \mathbf{w}_2)$的正负号。这意味着，如果你知道$\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$是正向的，那么，你不必检测$\mathbf{w}_3 = \mathbf{w}_1 \times \mathbf{w}_2$的正负性。

1112

参考文献

下面给出的是计算机图形学泛读的参考文献。除了各章引用的参考文献外，这里尽可能详尽地提供了便于查阅的出处。仅从书名和论文的题目也可以获知本研究领域中已经研究和正在研究的方向和思路。

某些期刊的引用率非常高，这里对期刊名做了简写。其中最重要的是ACM SIGGRAPH国际会议，其中的论文每年在《Computer Graphics》期刊上发表，另一个就是《ACM Transactions on Graphics》，这两个期刊的文献占了这里给出的总文献数量的三分之一以上。

缩写词

ACM TOG	*Association for Computing Machinery, Transactions on Graphics*
CACM	*Communications of the ACM*
CG & A	*IEEE Computer Graphics and Applications*
CGIP	*Computer Graphics and Image Processing*
CVGIP	*Computer Vision, Graphics, and Image Processing (formerly CGIP)*
FJCC	*Proceedings of the Fall Joint Computer Conference*
JACM	*Journal of the ACM*
NCC	*Proceedings of the National Computer Conference*
SJCC	*Proceedings of the Spring Joint Computer Conference*
SIGGRAPH 76	*Proceedings of SIGGRAPH '76 (Philadelphia, Pennsylvania, July 14–16, 1976). In Computer Graphics,* 10(2), Summer 1976, ACM SIGGRAPH, New York.
SIGGRAPH 77	*Proceedings of SIGGRAPH '77 (San Jose, California, July 20–22, 1977). In Computer Graphics,* 11(2), Summer 1977, ACM SIGGRAPH, New York.
SIGGRAPH 78	*Proceedings of SIGGRAPH '78 (Atlanta, Georgia, August 23–25, 1978). In Computer Graphics,* 12(3), August 1978, ACM SIGGRAPH, New York.
SIGGRAPH 79	*Proceedings of SIGGRAPH '79 (Chicago, Illinois, August 8–10, 1979). In Computer Graphics,* 13(2), August 1979, ACM SIGGRAPH, New York.
SIGGRAPH 80	*Proceedings of SIGGRAPH '80 (Seattle, Washington, July 14–18, 1980). In Computer Graphics,* 14(3), July 1980, ACM SIGGRAPH, New York.
SIGGRAPH 81	*Proceedings of SIGGRAPH '81 (Dallas, Texas, August 3–7, 1981). In Computer Graphics,* 15(3), August 1981, ACM SIGGRAPH, New York.
SIGGRAPH 82	*Proceedings of SIGGRAPH '82 (Boston, Massachusetts, July 26–30, 1982). In Computer Graphics,* 16(3), July 1982, ACM SIGGRAPH, New York.
SIGGRAPH 83	*Proceedings of SIGGRAPH '83 (Detroit, Michigan, July 25–29, 1983). In Computer Graphics,* 17(3), July 1983, ACM SIGGRAPH, New York.
SIGGRAPH 84	*Proceedings of SIGGRAPH '84 (Minneapolis, Minnesota, July*

23–27, 1984). In Computer Graphics, 18(3), July 1984, ACM SIGGRAPH, New York.

SIGGRAPH 85 — *Proceedings of SIGGRAPH '85 (San Francisco, California, July 22–26, 1985). In Computer Graphics*, 19(3), July 1985, ACM SIGGRAPH, New York.

SIGGRAPH 86 — *Proceedings of SIGGRAPH '86 (Dallas, Texas, August 18–22, 1986). In Computer Graphics*, 20(4), August 1986, ACM SIGGRAPH, New York.

SIGGRAPH 87 — *Proceedings of SIGGRAPH '87 (Anaheim, California, July 27–31, 1987). In Computer Graphics*, 21(4), July 1987, ACM SIGGRAPH, New York.

SIGGRAPH 88 — *Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1–5, 1988). In Computer Graphics*, 22(4), August 1988, ACM SIGGRAPH, New York.

SIGGRAPH 89 — *Proceedings of SIGGRAPH '89 (Boston, Massachusetts, July 31–August 4, 1989). In Computer Graphics*, 23(3), July 1989, ACM SIGGRAPH, New York.

ABIE89 — Abi-Ezzi, S.S., *The Graphical Processing of B-Splines in a Highly Dynamic Environment*, Ph.D. Thesis Rensselaer Polytechnic Institute, Troy, NY, May 1989.

ABRA85 — Abram, G., L. Westover, and T. Whitted, "Efficient Alias-Free Rendering Using Bit-Masks and Look-Up Tables," *SIGGRAPH 85*, 53–59.

ADOB85a — Adobe Systems, Inc., *PostScript Language Tutorial and Cookbook*, Addison-Wesley, Reading, MA, 1985.

ADOB85b — Adobe Systems, Inc., *PostScript Language Reference Manual*, Addison-Wesley, Reading, MA, 1985.

AKEL88 — Akeley, K., and T. Jermoluk, "High-Performance Polygon Rendering," *SIGGRAPH 88*, 239–246.

AKEL89 — Akeley, K., "The Silicon Graphics 4D/240GTX Superworkstation," *CG & A*, 9(4), July 1989, 71–83.

ALAV84 — Alavi, M., "An Assessment of the Prototyping Approach to Information Systems Development," *CACM*, 27(6), June 1984, 556–563.

AMAN84 — Amanatides, J., "Ray Tracing with Cones," *SIGGRAPH 84*, 129–135.

AMAN87 — Amanatides, J. and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," in Maréchal, G., ed., *Eurographics 87: Proceedings of the European Computer Graphics Conference and Exhibition, Amsterdam, August 24–28, 1987*, North Holland, Amsterdam, 1987, 3–10.

AMBU86 — Amburn, P., E. Grant, and T. Whitted, "Managing Geometric Complexity with Enhanced Procedural Models," *SIGGRAPH 86*, 189–195.

ANDE82 — Anderson, D.P., "Hidden Line Elimination in Projected Grid Surfaces," *ACM TOG*, 1(4), October 1982, 274–288.

ANDE83 — Anderson, D., "Techniques for Reducing Pen Plotting Time," *ACM TOG*, 2(3), July 1983, 197–212.

ANSI85a — ANSI (American National Standards Institute), *American National Standard for Human Factors Engineering of Visual Display Terminal Workstations*, ANSI, Washington, DC, 1985.

ANSI85b — ANSI (American National Standards Institute), *American National Standard for Information Processing Systems—Computer Graphics—Graphical Kernel System*

(GKS) Functional Description, ANSI X3.124-1985, ANSI, New York, 1985.

ANSI88 ANSI (American National Standards Institute), *American National Standard for Information Processing Systems—Programmer's Hierarchical Interactive Graphics System (PHIGS) Functional Description, Archive File Format, Clear-Text Encoding of Archive File,* ANSI, X3.144-1988, ANSI, New York, 1988.

APGA88 Apgar, B., B. Bersack, and A. Mammen, "A Display System for the Stellar Graphics Supercomputer Model GS1000," *SIGGRAPH 88,* 255–262.

APPE67 Appel, A., "The Notion of Quantitative Invisibility and the Machine Rendering of Solids," *Proceedings of the ACM National Conference,* Thompson Books, Washington, DC, 1967, 387–393. Also in FREE80, 214–220.

APPE68 Appel, A., "Some Techniques for Shading Machine Renderings of Solids," *SJCC,* 1968, 37–45.

APPE79 Appel, A., F.J. Rohlf, and A.J. Stein, "The Haloed Line Effect for Hidden Line Elimination," *SIGGRAPH 79,* 151–157.

APPL85 Apple Computer, Inc., *Inside Macintosh,* Addison-Wesley, Reading, MA, 1985.

APPL87 Apple Computer, Inc., *Human Interface Guidelines: The Apple Desktop Interface,* Addison-Wesley, Reading, MA, 1987.

APT85 Apt, C., "Perfecting the Picture," *IEEE Spectrum,* 22(7), July 1985, 60–66.

ARDE89 Ardent Computer Corp. (now Stardent), *Doré Product Literature,* Sunnyvale, CA, 1989.

ARNO88 Arnold, D. B., and P.R. Bono, *CGM and CGI : Metafile and Interface Standards for Computer Graphics,* Springer-Verlag, Berlin, 1988.

ARVO86 Arvo, J., "Backward Ray Tracing," in A.H. Barr, ed., *Developments in Ray Tracing, Course Notes 12 for SIGGRAPH 86,* Dallas, TX, August 18–22, 1986.

ARVO87 Arvo, J., and D. Kirk, "Fast Ray Tracing by Ray Classification," *SIGGRAPH 87,* 55–64.

ATHE78 Atherton, P.R., K. Weiler, and D. Greenberg, "Polygon Shadow Generation," *SIGGRAPH 78,* 275–281.

ATHE81 Atherton, P.R., "A Method of Interactive Visualization of CAD Surface Models on a Color Video Display," *SIGGRAPH 81,* 279–287.

ATHE83 Atherton, P.R., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry," *SIGGRAPH 83,* 73–82.

ATKI84 Atkinson, H.H., I. Gargantini, and M.V.S. Ramanath, "Determination of the 3D Border by Repeated Elimination of Internal Surfaces," *Computing,* 32(4), October 1984, 279–295.

ATKI86 Atkinson, W., U.S. Patent 4,622,545, November 1986.

AYAL85 Ayala, D., P. Brunet, R. Juan, and I. Navazo, "Object Representation by Means of Nonminimal Division Quadtrees and Octrees," *ACM TOG,* 4(1), January 1985, 41–59.

BADL87 Badler, N.I., K.H. Manoochehri, and G. Walters, "Articulated Figure Positioning by Multiple Constraints," *CG & A,* 7(6), June 1987, 28–38.

BAEC69 Baecker, R.M., "Picture Driven Animation," *SJCC,* AFIPS Press, Montvale, NJ, 1969, 273–288.

BAEC87 Baecker, R., and B. Buxton, *Readings in Human-Computer Interaction,* Morgan Kaufmann, Los Altos, CA, 1987.

BALD85 Baldauf, D., "The Workhorse CRT: New Life," *IEEE Spectrum,* 22(7), July 1985, 67–73.

BANC77 Banchoff, T.F., and C.M. Strauss, *The Hypercube: Projections and Slicings*, Film, International Film Bureau, 1977.

BANC83 Banchoff, T., and J. Wermer, *Linear Algebra Through Geometry*, Springer-Verlag, New York, 1983.

BARN88a Barnsley, M., A. Jacquin, F. Malassenet, L. Reuter, and A.D. Sloan, "Harnessing Chaos for Image Synthesis," *SIGGRAPH 88*, 131–140.

BARN88b Barnsley, M., "Harnessing Chaos for Image Synthesis," Lecture at ACM SIGGRAPH '88 meeting in Atlanta, GA, August 1988.

BARR79 Barros, J., and H. Fuchs, "Generating Smooth 2–D Monocolor Line Drawings on Video Displays," *SIGGRAPH 79*, 260–269.

BARR84 Barr, A., "Global and Local Deformations of Solid Primitives," *SIGGRAPH 84*, 21–30.

BARR88 Barr, A., and R. Barzel, "A Modeling System Based on Dynamic Constraints," *SIGGRAPH 88*, 179–188.

BARR88b Barry, P. and R. Goldman, "A Recursive Evaluation Algorithm for a Class of Catmull-Rom Splines," *SIGGRAPH 88*, 199–204.

BARR89a Barr, A.H., ed., *Topics in Physically Based Modeling*, Addison-Wesley, Reading, MA, 1989.

BARS83 Barsky, B., and J. Beatty, "Local Control of Bias and Tension in Beta-Splines," *ACM TOG*, 2(2), April 1983, 109–134.

BARS85 Barsky, B., and T. DeRose, "The Beta2-Spline: A Special Case of the Beta-Spline Curve and Surface Representation," *CG & A*, 5(9), September 1985, 46–58. See also erratum in 7(3), March 1987, 15.

BARS87 Barsky, B., T. DeRose, and M. Dippé, *An Adaptive Subdivision Method with Crack Prevention for Rendering Beta-spline Objects*, Report UCB/CSD 87/348, Department of Computer Science, University of California, Berkeley, CA, 1987.

BARS88 Barsky, B., *Computer Graphics and Geometric Modeling Using Beta-splines*, Springer-Verlag, New York, 1988.

BART87 Bartels, R., J. Beatty, and B. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann, Los Altos, CA, 1987.

BASS88 Bass, L., E. Hardy, K. Hoyt, M. Little, and R. Seacord, *Introduction to the Serpent User Interface Management System*, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, March 1988.

BAUM72 Baumgart, B.G., *Winged-edge Polyhedron Representation*, Technical Report STAN-CS-320, Computer Science Department, Stanford University, Palo Alto, CA, 1972.

BAUM74 Baumgart, B.G., *Geometric Modeling for Computer Vision*, Ph.D. Thesis, Report AIM-249, STAN-CS-74-463, Computer Science Department, Stanford University, Palo Alto, CA, October 1974.

BAUM75 Baumgart, B.G., "A Polyhedron Representation for Computer Vision," *NCC 75*, 589–596.

BAUM89 Baum, D.R., H.E. Rushmeier, and J.M. Winget, "Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors," *SIGGRAPH 89*, 325–334.

BAYE73 Bayer, B.E., "An Optimum Method for Two-Level Rendition of Continuous-Tone Pictures," in *Conference Record of the International Conference on Communications*, 1973, 26-11–26-15.

BEAT82 Beatty, J.C., and K.S. Booth, eds., *Tutorial: Computer Graphics*, Second Edition, IEEE Comp. Soc. Press, Silver Spring, MD, 1982.

BECK63 Beckmann, P., and A. Spizzichino, *The Scattering of Electromagnetic Waves from Rough Surfaces*, Macmillan, New York, 1963.

BEDF58 Bedford, R. and G. Wyszecki, "Wavelength Discrimination for Point Sources," *Journal of the Optical Society of America*, 48, 1958, 129-ff.

BENN84 Bennet, P.P., and S.A. Gabriel, "System for Spatially Transforming Images," U. S. Patent 4,472,732, September 18, 1984.

BENT82 Benton, S.A., "Survey of Holographic Stereograms," *Proceedings of SPIE*, 367, August 1982, 15–19.

BERG78 Bergeron, R., P. Bono, and J. Foley, "Graphics Programming Using the Core System," *Computing Surveys (Special Issue on Graphics Standards)*, 10(4), December 1978, 389–443.

BERG86a Bergeron, P., "A General Version of Crow's Shadow Volumes," *CG & A*, 6(9), September 1986, 17–28.

BERG86b Bergman, L., H. Fuchs, E. Grant, and S. Spach, "Image Rendering by Adaptive Refinement," *SIGGRAPH 86*, 29–37.

BERK68 Berkeley Physics Course, *Waves*, Volume 3, McGraw-Hill, New York, 1968.

BERK82 Berk, T., L. Brownston, and A. Kaufman, "A New Color-Naming System for Graphics Languages," *CG & A*, 2(3), May 1982, 37–44.

BERT81 Bertin, J., *Graphics and Graphics Information Processing*, de Gruyter, New York, 1981. Translated by Berg, W., and P. Scott from *La Graphique et le Traitement Graphique de l'Information*, Flammarion, Paris, 1977.

BERT83 Bertin, J., *Semiology of Graphics*, University of Wisconsin Press, Madison, WI, 1983. Translated by W. Berg from *Sémiologie Graphique*, Editions Gauthier-Villars, Paris; Editions Mouton & Cie, Paris-La Haye; and Ecole Pratique des Hautes Etudes, Paris, 1967.

BEWL83 Bewley, W., T. Roberts, D. Schroit, and W. Verplank, "Human Factors Testing in the Design of Xerox's 8010 'Star' Office Workstation," in *Proceedings CHI'83 Human Factors in Computing Systems Conference*, ACM, New York, 1983, 72–77.

BEZI70 Bézier, P., *Emploi des Machines á Commande Numérique*, Masson et Cie, Paris, 1970. Translated by Forrest, A. R., and A. F. Pankhurst as Bézier, P., *Numerical Control —Mathematics and Applications*, Wiley, London, 1972.

BEZI74 Bézier, P., "Mathematical and Practical Possibilities of UNISURF," in Barnhill, R. E., and R. F. Riesenfeld, eds., *Computer Aided Geometric Design*, Academic Press, New York, 1974.

BIER86a Bier, E., and M. Stone, "Snap-Dragging," *SIGGRAPH 86*, 233–240.

BIER86b Bier, E., "Skitters and Jacks: Interactive 3D Positioning Tools," in *Proceedings 1986 Workshop on Interactive 3D Graphics*, ACM, New York, 1987, 183–196.

BILL81 Billmeyer, F., and M. Saltzman, *Principles of Color Technology*, second edition, Wiley, New York, 1981.

BINF71 Binford, T., in *Visual Perception by Computer, Proceedings of the IEEE Conference on Systems and Control*, Miami, FL, December 1971.

BIRR61 Birren, R., *Creative Color*, Van Nostrand Reinhold, New York, 1961.

BISH60 Bishop, A., and M. Crook, *Absolute Identification of Color for Targets Presented Against White and Colored Backgrounds*. Report WADD TR 60-611, Wright Air Development Division, Wright Patterson AFB, Dayton, Ohio, 1960.

BISH86 Bishop, G., and D.M. Weimer, "Fast Phong Shading," *SIGGRAPH 86*, 103–106.

BLES82 Bleser, T., and J. Foley, "Towards Specifying and Evaluating the Human Factors of User-Computer Interfaces," in *Proceedings of the Human Factors in Computer Systems Conference*, ACM, New York, 1982, 309–314.

BLES86 Bleser, B., and J. Ward, "Human Factors Affecting the Problem of Machine

Recognition of Hand-Printed Text," in *Computer Graphics '86 Conference Proceedings*, Volume 3, NCGA, Fairfax, VA, 1986, 498–514.

BLES88a Bleser, T., J. Sibert, and J. P. McGee, "Charcoal Sketching: Returning Control to the Artist," *ACM TOG*, 7(1), January 1988, 76–81

BLES88b Bleser, T., *TAE Plus Styleguide User Interface Description*, NASA Goddard Space Flight Center, Greenbelt, MD, 1988.

BLIN76 Blinn, J.F., and M.E. Newell, "Texture and Reflection in Computer Generated Images," *CACM*, 19(10), October 1976, 542–547. Also in BEAT82, 456–461.

BLIN77a Blinn, J.F., "Models of Light Reflection for Computer Synthesized Pictures," *SIGGRAPH 77*, 192–198. Also in FREE80, 316–322.

BLIN77b Blinn, J.F., "A Homogeneous Formulation for Lines in 3-Space," *SIGGRAPH 77*, 237–241.

BLIN78a Blinn, J.F., and M.E. Newell, "Clipping Using Homogeneous Coordinates," *SIGGRAPH 78*, 245–251.

BLIN78b Blinn, J.F., "Simulation of Wrinkled Surfaces," *SIGGRAPH 78*, 286–292.

BLIN78c Blinn, J.F., *Computer Display of Curved Surfaces*, Ph.D. Thesis, Department of Computer Science, University of Utah, Salt Lake City, UT, December 1978.

BLIN82a Blinn, J.F., "Light Reflection Functions for the Simulation of Clouds and Dusty Surfaces," *SIGGRAPH 82*, 21–29.

BLIN82b Blinn, J.F., "A Generalization of Algebraic Surface Drawing," *ACM TOG*, 1(3), July 1982, 235–256.

BLIN85 Blinn, J.F., "Systems Aspects of Computer Image Synthesis and Computer Animation," in *Image Rendering Tricks, Course Notes 12 for SIGGRAPH 85*, New York, July 1985.

BLIN88 Blinn, J.F., "Me and My (Fake) Shadow," *CG & A*, 9(1), January 1988, 82–86.

BLIN89a Blinn, J.F., "What We Need Around Here is More Aliasing," *CG & A*, 9(1), January 1989, 75–79.

BLIN89b Blinn, J.F., "Return of the Jaggy," *CG & A*, 9(2), March 1989, 82–89.

BLOO88 Bloomenthal, J., "Polygonisation of Implicit Surfaces," *Computer Aided Geometric Design*, 5, 1988, 341-355.

BOLT80 Bolt, R.A., "'Put-That-There': Voice and Gesture at the Graphics Interface," *SIGGRAPH 80*, 262–270.

BOLT84 Bolt, R.A., *The Human Interface: Where People and Computers Meet*, Lifetime Learning Press, Belmont, CA, 1984.

BORD89 Borden, B.S., "Graphics Processing on a Graphics Supercomputer," *CG & A*, 9(4), July 1989, 56–62.

BORN79 Borning, A., "Thinglab—A Constraint-Oriented Simulation Laboratory," Technical Report SSl-79-3, Xerox Palo Alto Research Center, Palo Alto, CA, July 1979.

BORN86a Borning, A., and R. Duisberg, "Constraint-Based Tools for Building User Interfaces," *ACM TOG*, 5(4), October 1986, 345–374.

BORN86b Borning, A., "Defining Constraints Graphically," in *SIGCHI '86 Conference Proceedings*, ACM, New York, 1986, 137–143.

BOUK70a Bouknight, W.J., "A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations," *CACM*, 13(9), September 1970, 527–536. Also in FREE80, 292–301.

BOUK70b Bouknight, W.J., and K.C. Kelly, "An Algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movable Light Sources," *SJCC*, AFIPS Press, Montvale, NJ, 1970, 1–10.

BOUV85 Bouville, C., "Bounding Ellipsoids for Ray-Fractal Intersection," *SIGGRAPH 85*, 45–52.

BOYN79 Boynton, R. M., *Human Color Vision*, Holt, Rinehart, and Winston, New York, 1979.

BOYS82 Boyse, J.W., and J.E. Gilchrist, "GMSolid: Interactive Modeling for Design and Analysis of Solids," *CG & A*, 2(2), March 1982, 27–40.

BÖHM80 Böhm, W., "Inserting New Knots into B-spline Curves," *Computer Aided Design*, 12(4), July 1980, 199–201.

BÖHM84 Böhm, W., G. Farin, and J. Kahmann, "A Survey of Curve and Surface Methods in CAGD," *Computer Aided Geometric Design*, 1(1), July 1984, 1–60.

BRAI78 Braid, I.C., R.C. Hillyard, and I.A. Stroud, *Stepwise Construction of Polyhedra in Geometric Modelling*, CAD Group Document No. 100, Cambridge University, Cambridge, England, 1978. Also in K.W. Brodlie, ed., *Mathematical Methods in Computer Graphics and Design*, Academic Press, New York, 1980, 123–141.

BRES65 Bresenham, J.E., "Algorithm for Computer Control of a Digital Plotter," *IBM Systems Journal*, 4(1), 1965, 25–30.

BRES77 Bresenham, J.E. "A Linear Algorithm for Incremental Digital Display of Circular Arcs," *Communications of the ACM*, 20(2), February 1977, 100–106.

BRES83 Bresenham, J.E., D.G. Grice, and S.C. Pi, "Bi-Directional Display of Circular Arcs," US Patent 4,371,933, February 1, 1983.

BREW77 Brewer, H., and D. Anderson, "Visual Interaction with Overhauser Curves and Surfaces," *SIGGRAPH 77*, 132–137.

BRIG74 Brigham, E.O., *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

BRIT78 Britton, E., J. Lipscomb, and M. Pique, "Making Nested Rotations Convenient for the User," *SIGGRAPH 78*, 222–227.

BROO88 Brooktree Corporation, *Product Databook 1988*, Brooktree Corporation, San Diego, CA, 1987.

BROT84 Brotman, L.S., and N.I. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *CG & A*, 4(10), October 1984, 5–12.

BROW64 Brown, R., "On-Line Computer Recognition of Hand-Printed Characters," *IEEE Trans. Computers*, Vol. EC-13(12), December 1964, 750–752.

BROW82 Brown, C.M., "PADL-2: A Technical Summary," *CG & A*, 2(2), March 1982, 69–84.

BUIT75 Bui-Tuong, Phong, "Illumination for Computer Generated Pictures," *CACM*, 18(6), June 1975, 311–317. Also in BEAT82, 449–455.

BUNK89 Bunker, M., and R. Economy, *Evolution of GE CIG Systems*, SCSD Document, General Electric Company, Daytona Beach, FL, 1989.

BURT74 Burton, R.P., and I. E. Sutherland, "Twinkle Box: A Three-Dimensional Computer Input Device," *NCC 1974*, AFIPS Press, Montvale, NJ, 1974, 513–520.

BURT76 Burtnyk, N., and M. Wein, "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation," *CACM*, 19(10), October 1976, 564–569

BUTL79 Butland, J., "Surface Drawing Made Simple," *Computer-Aided Design*, 11(1), January 1979, 19–22.

BUXT83 Buxton, W., M.R. Lamb, D. Sherman, and K.C. Smith, "Towards a Comprehensive User Interface Management System," *SIGGRAPH 83*, 35–42.

BUXT85 Buxton, W., R. Hill, and P. Rowley, "Issues and Techniques in Touch-Sensitive Tablet Input," *SIGGRAPH 85*, 215–224.

BUXT86 Buxton, W., "There's More to Interaction Than Meets the Eye: Issues in Manual

Input," in Norman, D., and S. Draper, eds., *User-Centered System Design*, Lawrence Erlbaum, Hillsdale, NJ, 1986, 319–337. Also in BAEC 87, 366–375.

BYTE85 *BYTE Magazine*, 10(5), May 1985, 151–167.

CABR87 Cabral, B., N. Max, and R. Springmeyer, "Bidirectional Reflection Functions from Surface Bump Maps," *SIGGRAPH 87*, 273–281.

CACM84 "Computer Graphics Comes of Age: An Interview with Andries van Dam." *CACM*, 27(7), July 1984, 638–648.

CALL88 Callahan, J., D. Hopkins, M. Weiser, and B. Shneiderman, "An Empirical Comparison of Pie vs. Linear Menus," in *Proceedings of CHI 1988*, ACM, New York, 95–100.

CARD78 Card, S., W. English, and B. Burr, "Evaluation of Mouse, Rate-Controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection of a CRT," *Ergonomics*, 21(8), August 1978, 601–613.

CARD80 Card, S., T. Moran, and A. Newell, "The Keystroke-Level Model for User Performance Time with Interactive Systems," *CACM*, 23(7), July 1980, 398–410.

CARD82 Card, S., "User Perceptual Mechanisms in the Search of Computer Command Menus," in *Proceedings of the Human Factors in Computer Systems Conference*, ACM, New York, March 1982, 20–24.

CARD83 Card, S., T. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

CARD85 Cardelli, L., and R. Pike, "Squeak: A Language for Communicating with Mice," *SIGGRAPH 85*, 199–204.

CARD88 Cardelli, L., "Building User Interfaces by Direct Manipulation," in *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 1988, 152–166.

CARL78 Carlbom, I., and J. Paciorek, "Planar Geometric Projections and Viewing Transformations," *Computing Surveys*, 10(4), December 1978, 465–502.

CARL85 Carlbom, I., I. Chakravarty, and D. Vanderschel, "A Hierarchical Data Structure for Representing the Spatial Decomposition of 3-D Objects," *CG & A*, 5(4), April 1985, 24–31.

CARL87 Carlbom, I., "An Algorithm for Geometric Set Operations Using Cellular Subdivision Techniques," *CG & A*, 7(5), May 1987, 44–55.

CARP84 Carpenter, L., "The A-buffer, an Antialiased Hidden Surface Method," *SIGGRAPH 84*, 103–108.

CATM72 Catmull, E., "A System for Computer Generated Movies," in *Proc. ACM Annual Conference*, ACM, New York, NY, August 1972, 422–431.

CATM74a Catmull, E., and R. Rom, "A Class of Local Interpolating Splines," in Barnhill, R., and R. Riesenfeld, eds., *Computer Aided Geometric Design*, Academic Press, San Francisco, 1974, 317–326.

CATM74b Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, December 1974.

CATM75 Catmull, E., "Computer Display of Curved Surfaces," in *Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structures*, May 1975. Also in FREE80, 309–315.

CATM78a Catmull, E., "The Problems of Computer-Assisted Animation," *SIGGRAPH 78*, 348–353.

CATM78b Catmull, E., "A Hidden-Surface Algorithm with Anti-Aliasing," *SIGGRAPH 78*, 6–11. Also in BEAT82, 462–467.

CATM79 Catmull, E., "A Tutorial on Compensation Tables," *SIGGRAPH 79*, 279–285.

CATM80 Catmull, E., and A.R. Smith, "3-D Transformations of Images in Scanline Order," *SIGGRAPH 80*, 279–285.

CGA82 Special Issue on Modeling the Human Body for Animation, *CG & A*, 2(11) N. Badler, ed., November 1982, 1–81.

CHAP72 Chapanis, A., and R. Kinkade, "Design of Controls," in Van Cott, H., and R. Kinkade, eds., *Human Engineering Guide to Equipment Design*, U.S. Government Printing Office, 1972.

CHAS81 Chasen, S.H., "Historical Highlights of Interactive Computer Graphics," *Mechanical Engineering*, 103, ASME, November 1981, 32–41.

CHEN88 Chen, M., J. Mountford, and A. Sellen, "A Study in Interactive 3-D Rotation Using 2-D Control Devices," *SIGGRAPH 88*, 121–129.

CHIN89 Chin, N., and S. Feiner, "Near Real-Time Shadow Generation Using BSP Trees," *SIGGRAPH 89*, 99–106.

CHIN90 Chin, N., *Near Real-Time Object-Precision Shadow Generation Using BSP Trees*, M.S. Thesis, Department of Computer Science, Columbia University, New York, 1990.

CHRI75 Christ, R., "Review and Analysis of Color Coding for Visual Display," *Human Factors*, 17(6), December 1975, 542–570.

CHUN89 Chung, J.C., *et al.*, "Exploring Virtual Worlds with Head-Mounted Displays," *Proc. SPIE Meeting on Non-Holographic True 3-Dimensional Display Technologies*, 1083, Los Angeles, Jan. 15-20, 1989.

CLAR76 Clark, J.H., "Hierarchical Geometric Models for Visible Surface Algorithms," *CACM*, 19(10), October 1976, 547–554. Also in BEAT82, 296–303.

CLAR79 Clark, J., "A Fast Scan-Line Algorithm for Rendering Parametric Surfaces," abstract in *SIGGRAPH 79*, 174. Also in Whitted, T., and R. Cook, eds., *Image Rendering Tricks, Course Notes 16 for SIGGRAPH 86*, Dallas, TX, August 1986. Also in JOY88, 88–93.

CLAR80 Clark, J., and M. Hannah, "Distributed Processing in a High-Performance Smart Image Memory," *Lambda (VLSI Design)*, 1(3), Q4 1980, 40–45.

CLAR82 Clark, J., "The Geometry Engine: A VLSI Geometry System for Graphics," *SIGGRAPH 82*, 127–133.

CLEA83 Cleary, J., B. Wyvill, G. Birtwistle, and R. Vatti, "Design and Analysis of a Parallel Ray Tracing Computer," *Proceedings of Graphics Interface '83*, May 1983, 33–34.

CLEV83 Cleveland, W., and R. McGill, "A Color-Caused Optical Illusion on a Statistical Graph," *The American Statistician*, 37(2), May 1983, 101–105.

CLEV84 Cleveland, W., and R. McGill, "Graphical Perception: Theory, Experimentation and Application to the Development of Graphical Methods," *Journal of the American Statistical Association*, 79(387), September 1984, 531–554.

CLEV85 Cleveland, W., and R. McGill, "Graphical Perception and Graphical Methods for Analyzing Scientific Data," *Science*, 229, August 30, 1985, 828–833.

COHE80 Cohen, E., T. Lyche, and R. Riesenfeld, "Discrete B-Splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics," *CGIP*, 14(2), October 1980, 87–111.

COHE83 Cohen, E., "Some Mathematical Tools for a Modeler's Workbench," *CG & A*, 3(7), October 1983, 63–66.

COHE85 Cohen, M.F., and D.P. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments," *SIGGRAPH 85*, 31–40.

COHE86 Cohen, M.F., D.P. Greenberg, D.S. Immel, and P.J. Brock, "An Efficient Radiosity Approach for Realistic Image Synthesis," *CG & A*, 6(3), March 1986, 26–35.

COHE88 Cohen, M.F., S.E. Chen, J.R. Wallace, and D.P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation," *SIGGRAPH 88*, 75–84.

CONR85 Conrac Corporation, *Raster Graphics Handbook*, second edition, Van Nostrand Reinhold, New York, 1985.

COOK82 Cook, R., and K. Torrance, "A Reflectance Model for Computer Graphics," *ACM TOG*, 1(1), January 1982, 7–24.

COOK84a Cook, R.L., "Shade Trees," *SIGGRAPH 84*, 223–231.

COOK84b Cook, R.L., T. Porter, and L. Carpenter, "Distributed Ray Tracing," *SIGGRAPH 84*, 137–145.

COOK86 Cook, R.L., "Stochastic Sampling in Computer Graphics," *ACM TOG*, 5(1), January 1986, 51–72.

COOK87 Cook, R.L., L. Carpenter, and E. Catmull, "The Reyes Image Rendering Architecture," *SIGGRAPH 87*, 95–102.

COON67 Coons, S. A., *Surfaces for Computer Aided Design of Space Forms*, MIT Project Mac, TR-41, MIT, Cambridge, MA, June 1967.

COSS89 Cossey, G., *Prototyper*, SmethersBarnes, Portland, OR, 1989.

COWA83 Cowan, W., "An Inexpensive Scheme for Calibration of a Colour Monitor in Terms of CIE Standard Coordinates," *SIGGRAPH 83*, 315–321.

CROC84 Crocker, G.A., "Invisibility Coherence for Faster Scan-Line Hidden Surface Algorithms," *SIGGRAPH 84*, 95–102.

CROW77a Crow, F.C., "Shadow Algorithms for Computer Graphics," *SIGGRAPH 77*, 242–247. Also in BEAT82, 442–448.

CROW77b Crow, F.C., "The Aliasing Problem in Computer-Generated Shaded Images," *CACM*, 20(11), November 1977, 799–805.

CROW78 Crow, F., "The Use of Grayscale for Improved Raster Display of Vectors and Characters," *SIGGRAPH 78*, 1–5.

CROW81 Crow, F.C., "A Comparison of Antialiasing Techniques," *CG & A*, 1(1), January 1981, 40–48.

CROW84 Crow, F.C., "Summed-Area Tables for Texture Mapping," *SIGGRAPH 84*, 207–212.

CYRU78 Cyrus, M. and J. Beck, "Generalized Two- and Three-Dimensional Clipping," *Computers and Graphics*, 3(1), 1978, 23–28.

DALL80 Dallas, W.J., "Computer Generated Holograms," in *The Computer in Optical Research*, Frieden, B.R , ed., Springer-Verlag, New York, 1980, 291–366.

DASI89 Da Silva, D., *Raster Algorithms for 2D Primitives*, Master's Thesis, Computer Science Department, Brown University, Providence, RI, 1989.

DEBO78 de Boor, C., *A Practical Guide to Splines*, Applied Mathematical Sciences Volume 27, Springer-Verlag, New York, 1978.

DECA59 de Casteljau, F., *Outillage Méthodes Calcul*, André Citroën Automobiles SA, Paris, 1959.

DEER88 Deering, M., S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics," *SIGGRAPH 88*, 21–30.

DELA88 Delany, H.C., "Ray Tracing on a Connection Machine," in *Proceedings of the 1988 International Conference on Supercomputing*, July 4–8, 1988, St. Malo, France, 659–664.

DEME80 Demetrescu, S., *A VLSI-Based Real-Time Hidden-Surface Elimination Display System*, Master's Thesis, Department of Computer Science, California Institute of Technology, Pasadena, CA, May 1980.

DEME85 Demetrescu, S., "High Speed Image Rasterization Using Scan Line Access Memories," in *Proceedings of the 1985 Chapel Hill Conference on VLSI*, Rockville, MD, Computer Science Press, 221–243.

DEYO89 Deyo, R., and D. Ingebretson, "Notes on Real-Time Vehicle Simulation," in *Implementing and Interacting with Real-Time Microworlds, Course Notes 29 for SIGGRAPH 89*, Boston, MA, August 1989.

DIGI89 Digital Equipment Corporation, *DEC XUI Style Guide*, Digital Equipment Corporation, Maynard, MA, 1989.

DIPP84 Dippé, M. and J. Swensen, "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *SIGGRAPH 84*, 149–158.

DIPP85 Dippé, M., and E.H. Wold, "Antialiasing through Stochastic Sampling," *SIGGRAPH 85*, 69–78.

DOCT81 Doctor, L., and J. Torborg, "Display Techniques for Octree-Encoded Objects," *CG & A*, 1(3), July 1981, 29–38.

DONA88 Donato, N., and R. Rocchetti, "Techniques for Manipulating Arbitrary Regions," in *Course Notes 11 for SIGGRAPH 88*, Atlanta, GA, August, 1988.

DREB88 Drebin, R.A., L. Carpenter, and P. Hanrahan, "Volume Rendering," *SIGGRAPH 88, 65–74*.

DUFF79 Duff, T., "Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays," *SIGGRAPH 79*, 270–275.

DURB88 Durbeck, R., and S. Sherr, eds., *Output Hardcopy Devices*, Academic Press, New York, 1988.

DUVA90 Duvanenko, V., W.E. Robbins, R.S. Gyurcsik, "Improved Line Segment Clipping," *Dr. Dobb's Journal*, July 1990, 36–45, 98–100.

DVOŘ43 Dvořák, A., "There is a Better Typewriter Keyboard," *National Business Education Quarterly*, 12(2), 1943, 51–58.

ELLS89 Ellsworth, D., *Pixel-Planes 5 Rendering Control*, Tech. Rep. TR89-003, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1989.

EMER03 Emerson, R., "Essays: First Series—Self Reliance," in *The Complete Works of Ralph Waldo Emerson*, Houghton Mifflin, Boston, MA, 1903.

ENCA72 Encarnacao, J., and W. Giloi, "PRADIS—An Advanced Programming System for 3-D-Display," *SJCC*, AFIPS Press, Montvale, NJ, 1972, 985–998.

ENDE87 Enderle, G, K. Kansy, and G. Pfaff, *Computer Graphics Programming*, second edition, Springer-Verlag, Berlin, 1987.

ENGE68 Engelbart, D.C., and W.K. English, *A Research Center for Augmenting Human Intellect, FJCC* , Thompson Books, Washington, D.C., 395.

ENGL89 England, N., "Evolution of High Performance Graphics Systems," in *Proceedings of Graphics Interface '89*, Canadian Information Processsing Society (Morgan Kauffman in U.S.), 1989.

EVAN81 Evans, K., P. Tanner, and M. Wein, "Tablet-Based Valuators that Provide One, Two or Three Degrees of Freedom," *SIGGRAPH 81*, 91–97.

EVAN89 Evans & Sutherland Computer Corporation, *The Breadth of Visual Simulation Technology*, Evans & Sutherland Computer Corporation, Salt Lake City, UT, 1989.

EYLE88 Eyles, J., J. Austin, H. Fuchs, T. Greer, and J. Poulton, "Pixel-Planes 4: A Summary," in *Advances in Computer Graphics Hardware II* (1987 Eurographics Workshop on Graphics Hardware), Eurographics Seminars, 1988, 183-208.

FARI86 Farin, G., "Triangular Bernstein-Bézier Patches," *Computer Aided Geometric Design*, 3(2), August 1986, 83–127.

FARI88 Farin, G., *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, New York, 1988.

FAUX79 Faux, I. D., and M. J. Pratt, *Computational Geometry for Design and Manufacture*, Wiley, New York, 1979.

FEIB80 Feibush, E.A., M. Levoy, and R.L. Cook, "Synthetic Texturing Using Digital Filters," *SIGGRAPH 80*, 294–301.

FEIN82a Feiner, S., S. Nagy, and A. van Dam, "An Experimental System for Creating and Presenting Interactive Graphical Documents," *ACM TOG*, 1(1), January 1982, 59–77.

FEIN82b Feiner, S., D. Salesin, and T. Banchoff, "DIAL: A Diagrammatic Animation Language," *CG & A*, 2(7), September 1982, 43–54.

FEIN85 Feiner, S., "APEX: An Experiment in the Automated Creation of Pictorial Explanations," *CG & A*, 5(11), November 1985, 29–38.

FEIN88 Feiner, S., "A Grid-Based Approach to Automating Display Layout," in *Proceedings of Graphics Interface '88*, Edmonton, Canada, June 1988, 192–197.

FERG64 Ferguson, J., "Multivariate Curve Interpolation," *JACM*, 11(2), April 1964, 221–228.

FIEL86 Field, D., "Algorithms for Drawing Anti-Aliased Circles and Ellipses," *CGVIP*, 33(1), January 1986, 1–15.

FISH84 Fishkin, K.P., and B.A. Barsky, "A Family of New Algorithms for Soft Filling," *SIGGRAPH 84*, 235–244.

FISH86 Fisher, S.S., M. McGreevy, J. Humphries, and W. Robinett, "Virtual Environment Display System," in *Proceedings of the 1986 Chapel Hill Workshop on Interactive 3D Graphics*, Chapel Hill, NC, 1986, 77–87.

FITT54 Fitts, P., "The Information Capacity of the Human Motor System in Controlling Amplitude of Motion," *Journal of Experimental Psychology*, 47(6), June 1954, 381–391.

FIUM89 Fiume, E.L., *The Mathematical Structure of Raster Graphics*, Academic Press, San Diego, 1989.

FLEC87 Flecchia, M., and R. Bergeron, "Specifying Complex Dialogs in Algae," in *Proceedings of CHI + GI '87*, ACM, New York, 1987, 229–234.

FLOY75 Floyd, R., and Steinberg, L., "An Adaptive Algorithm for Spatial Gray Scale," in *Society for Information Display 1975 Symposium Digest of Technical Papers*, 1975, 36.

FOLE71 Foley, J., "An Approach to the Optimum Design of Computer Graphics Systems," *CACM*, 14 (6), June 1971, 380–390.

FOLE74 Foley, J., and V. Wallace, "The Art of Natural Man–Machine Communication," *Proceedings IEEE*, 62(4), April 1974, 462–470.

FOLE76 Foley, J., "A Tutorial on Satellite Graphics Systems," *IEEE Computer* 9(8), August 1976, 14–21.

FOLE82 Foley, J., and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.

FOLE84 Foley, J., V. Wallace, and P. Chan, "The Human Factors of Computer Graphics Interaction Techniques," *CG & A*, 4(11), November 1984, 13–48.

FOLE87a Foley, J., "Interfaces for Advanced Computing," *Scientific American*, 257(4), October 1987, 126–135.

FOLE87b Foley, J., W. Kim, and C. Gibbs, "Algorithms to Transform the Formal Specification of a User Computer Interface," in *Proceedings INTERACT '87, 2nd IFIP Conference on Human-Computer Interaction*, Elsevier Science Publishers, Amsterdam, 1987, 1001-1006.

FOLE87c Foley, J., and W. Kim, "ICL—The Image Composition Language," *CG & A*, 7(11), November 1987, 26–35.

FOLE88 Foley, J., W. Kim, and S. Kovacevic, "A Knowledge Base for User Interface Management System," in *Proceedings of CHI '88–1988 SIGCHI Computer-Human Interaction Conference*, ACM, New York, 1988, 67-72.

FOLE89 Foley, J., W. Kim, S. Kovacevic, and K. Murray, "Defining Interfaces at a High Level of Abstraction," *IEEE Software*, 6(1), January 1989, 25–32.

FORR79 Forrest, A. R., "On the Rendering of Surfaces," *SIGGRAPH 79*, 253–259.

FORR80 Forrest, A. R., "The Twisted Cubic Curve: A Computer-Aided Geometric Design Approach," *Computer Aided Design*, 12(4), July 1980, 165–172.

FORR85 Forrest, A. R., "Antialiasing in Practice", in Earnshaw, R. A., ed., *Fundamental Algorithms for Computer Graphics*, NATO ASI Series F: Computer and Systems Sciences, Vol. 17, Springer-Verlag, New York, 1985, 113–134.

FORS88 Forsey, D.R., and R.H. Bartels, "Hierarchical B-spline Refinement," *SIGGRAPH 88*, 205–212.

FOUR82 Fournier, A., D. Fussell, and L. Carpenter, "Computer Rendering of Stochastic Models, " *CACM*, 25(6), June 1982, 371–384.

FOUR86 Fournier, A., and W.T. Reeves, "A Simple Model of Ocean Waves," *SIGGRAPH 86*, 75–84.

FOUR88 Fournier, A. and D. Fussell, "On the Power of the Frame Buffer," *ACM TOG*, 7(2), April 1988, 103–128.

FRAN81 Franklin, W.R., "An Exact Hidden Sphere Algorithm that Operates in Linear Time," *CGIP*, 15(4), April 1981, 364–379.

FREE80 Freeman, H. ed., *Tutorial and Selected Readings in Interactive Computer Graphics*, IEEE Comp. Soc. Press, Silver Spring, MD, 1980.

FRIE85 Frieder, G., D. Gordon, and R. Reynolds, "Back-to-Front Display of Voxel-Based Objects," *CG & A*, 5(1), January 1985, 52–60.

FROM84 Fromme, F., "Improving Color CAD Systems for Users: Some Suggestions from Human Factors Studies," *IEEE Design and Test of Computers*, 1(1), February 1984, 18–27.

FUCH77a Fuchs, H., J. Duran, and B. Johnson, "A System for Automatic Acquisition of Three-Dimensional Data," in *Proceedings of the 1977 NCC*, AFIPS Press, 1977, 49–53.

FUCH77b Fuchs, H., "Distributing a Visible Surface Algorithm over Multiple Processors," *Proceedings of the ACM Annual Conference*, Seattle, WA, October 1977, 449–451.

FUCH79 Fuchs, H., and B. Johnson, "An Expandable Multiprocessor Architecture for Video Graphics," *Proceedings of the 6th ACM-IEEE Symposium on Computer Architecture*, Philadelphia, PA, April 1979, 58–67.

FUCH80 Fuchs, H., Z.M. Kedem, and B.F. Naylor, "On Visible Surface Generation by A Priori Tree Structures," *SIGGRAPH 80*, 124–133.

FUCH81 Fuchs, H. and J. Poulton, "Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine," *VLSI Design*, 2(3), Q3 1981, 20–28.

FUCH82 Fuchs, H., S.M. Pizer, E.R. Heinz, S.H. Bloomberg, L. Tsai, and D.C. Strickland, "Design of and Image Editing with a Space-Filling Three-Dimensional Display Based on a Standard Raster Graphics System," *Proceedings of SPIE*, 367, August 1982, 117–127.

FUCH83 Fuchs, H., G.D. Abram, and E.D. Grant, "Near Real-Time Shaded Display of Rigid Objects," *SIGGRAPH 83*, 65–72.

FUCH85 Fuchs, H., J. Goldfeather, J. Hultquist, S. Spach, J. Austin, F. Brooks, J. Eyles, and J. Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes," *SIGGRAPH 85*, 111–120.

FUCH89 Fuchs, H., J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *SIGGRAPH 89*, 79–88.

FUJI85 Fujimura, K., and Kunii, T. L., "A Hierarchical Space Indexing Method," in Kunii, T. L., ed., *Computer Graphics: Visual Technology and Art , Proceedings of Computer Graphics Tokyo '85 Conference*, Springer-Verlag, 1985, 21–34.

FUSS82 Fussell, D., and B. D. Rathi, "A VLSI-Oriented Architecture for Real-Time Raster Display of Shaded Polygons," in *Proceedings of Graphics Interface '82*, Toronto, May 1982, 373–380.

GAIN84 Gaines, B., and M. Shaw, *The Art of Computer Conversation*, Prentice-Hall International, Englewood Cliffs, NJ, 1984.

GALI69 Galimberti, R., and U. Montanari, "An Algorithm for Hidden Line Elimination," *CACM*, 12(4), April 1969, 206–211.

GARD84 Gardner, G.Y., "Simulation of Natural Scenes Using Textured Quadric Surfaces," *SIGGRAPH 84*, 11–20.

GARD85 Gardner, G.Y., "Visual Simulation of Clouds," *SIGGRAPH 85*, 297–303.

GARG82 Gargantini, I., "Linear Octtrees for Fast Processing of Three-Dimensional Objects," *CGIP*, 20(4), December 1982, 365–374.

GARG86 Gargantini, I., T. Walsh, and O. Wu, "Viewing Transformations of Voxel-Based Objects via Linear Octrees," *CG & A*, 6(10), October 1986, 12–21.

GARR80 Garrett, M., *A Unified Non-Procedural Environment for Designing and Implementing Graphical Interfaces to Relational Data Base Management Systems*, Ph.D. dissertation, Technical Report GWU-EE/CS-80-13, Department of Electrical Engineering and Computer Science, The George Washington University, Washington, DC, 1980.

GARR82 Garrett, M., and J. Foley, "Graphics Programming Using a Database System with Dependency Declarations," *ACM TOG*, 1(2), April 1982, 109–128.

GHAR88 Gharachorloo, N., S. Gupta, E. Hokenek, P. Balasubramanian, B. Bogholtz, C. Mathieu, and C. Zoulas, "Subnanosecond Pixel Rendering with Million Transistor Chips," *SIGGRAPH 88*, 41–49.

GHAR89 Gharachorloo, N., S. Gupta, R.F. Sproull, and I.E. Sutherland, "A Characterization of Ten Rasterization Techniques," *SIGGRAPH 89*, 355–368.

GILO78 Giloi, W.K., *Interactive Computer Graphics—Data Structures, Algorithms, Languages*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

GINS83 Ginsberg, C.M., and D. Maxwell, "Graphical Marionette," in *Proceedings of the SIGGRAPH/SIGART Interdisciplinary Workshop on Motion: Representation and Perception*, Toronto, April 4–6, 1983, 172–179.

GIRA85 Girard, M., and A.A. Maciejewski, "Computational Modeling for the Computer Animation of Legged Figures," *SIGGRAPH 85*, 263–270.

GIRA87 Girard, M., "Interactive Design of 3D Computer-Animated Legged Animal Motion," *CG & A*, 7(6), June 1987, 39–51.

GLAS84 Glassner, A.S., "Space Subdivision for Fast Ray Tracing," *CG & A*, 4(10), October 1984, 15–22.

GLAS86 Glassner, A.S., "Adaptive Precision in Texture Mapping," *SIGGRAPH 86*, 297–306.

GLAS88 Glassner, A., "Spacetime Raytracing for Animation," *CG & A*, 8(2), March 1988, 60–70.

GLAS89 Glassner, A.S., ed., *An Introduction to Ray Tracing*, Academic Press, London, 1989.

GOLD71 Goldstein, R.A., and R. Nagel, "3-D Visual Simulation," *Simulation*, 16(1), January 1971, 25–31.

GOLD76 Goldberg, A., and Kay, A., *SMALLTALK-72 Instruction Manual*, Learning Research Group, Xerox Palo Alto Research Center, Palo Alto, CA, March 1976.

GOLD80 Goldstein, H., *Classical Mechanics*, Addison-Wesley, Reading, MA, 1980.

GOLD83 Goldberg, A., and D. Robson, *SmallTalk 80: The Language and Its Implementation*, Addison-Wesley, Reading, MA, 1983.

GOLD84 Goldwasser, S.M., "A Generalized Object Display Processor Architecture," in *Proceedings of the 11th Annual International Symposium on Computer Architecture*, Ann Arbor, MI, June 5–7, 1984, *SIGARCH Newsletter*, 12(3), June 1984, 38–47.

GOLD86 Goldfeather, J., J.P.M. Hultquist, and H. Fuchs, "Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System," *SIGGRAPH 86*, 107–116.

GOLD87 Goldsmith, J., and J. Salmon, "Automatic Creation of Object Hierarchies for Ray Tracing," *CG & A*, 7(5), May 1987, 14–20.

GOLD88 Goldwasser, S.M., R.A. Reynolds, D.A. Talton, and E.S. Walsh, "Techniques for the Rapid Display and Manipulation of 3-D Biomedical Data," *Comp. Med. Imag. and Graphics*, 12(1), 1988, 1–24.

GOLD89 Goldfeather, J., S. Molnar, G. Turk, and H. Fuchs, "Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning," *CG & A*, 9(3), May 1989, 20–28.

GONZ87 Gonzalez, R., and P. Wintz, *Digital Image Processing*, second edition, Addison-Wesley, Reading, MA, 1987.

GORA84 Goral, C.M., K.E. Torrance, D.P. Greenberg, and B. Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces," *SIGGRAPH 84*, 213–222.

GORI87 Goris, A., B. Fredrickson, and H.L. Baeverstad, Jr., "A Configurable Pixel Cache for Fast Image Generation," *CG & A*, 7(3), March 1987, 24–32.

GOSL89 Gosling, J., personal communication, March 1989.

GOSS88 Gossard, D., R. Zuffante, and H. Sakurai, "Representing Dimensions, Tolerances, and Features in MCAE Systems," *CG & A*, 8(2), March 1988, 51–59.

GOUR71 Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Trans. on Computers*, C-20(6), June 1971, 623–629. Also in FREE80, 302–308.

GREE85a Green, M., "The University of Alberta User Interface Management System," *SIGGRAPH 85*, 205–213.

GREE85b Green, M., *The Design of Graphical User Interfaces*, Technical Report CSRI-170, Department of Computer Science, University of Toronto, Toronto, 1985.

GREE85c Greene, R., "The Drawing Prism: A Versatile Graphic Input Device," *SIGGRAPH 85*, 103–110.

GREE86 Greene, N., "Environment Mapping and Other Applications of World Projections," *CG & A*, 6(11), November 1986, 21–29.

GREE87a Green, M., "A Survey of Three Dialog Models," *ACM TOG*, 5(3), July 1987, 244–275.

GREE87b Greenstein, J. and L. Arnaut, "Human Factors Aspects of Manual Computer Input Devices," in Salvendy, G., ed., *Handbook of Human Factors*, Wiley, New York, 1987, 1450–1489.

GREG66 Gregory, R.L., *Eye and Brain— The Psychology of Seeing*, McGraw-Hill, New York, 1966.

GREG70 Gregory, R.L., *The Intelligent Eye*, McGraw-Hill, London, 1970.

GRIM89 Grimes, J., L. Kohn, and R. Bharadhwaj, "The Intel i860 64-Bit Processor: A General-Purpose CPU with 3D Graphics Capabilities," *CG & A*, 9(4), July 1989, 85–94.

GSPC77 Graphics Standards Planning Committee. "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH." *Computer Graphics*, 11(3), Fall 1977.

GSPC79 Graphics Standards Planning Committee. "Status Report of the Graphics Standards Planning Committee." *Computer Graphics*, 13(3), August 1979.

GTCO82 GTCO Corporation, *DIGI-PAD 5 User's Manual*, GTCO Corporation, Rockville, MD, 1982.

GUPT81a Gupta, S., and R.E. Sproull, "Filtering Edges for Gray-Scale Displays," *SIGGRAPH 81*, 1–5.

GUPT81b Gupta, S., R. Sproull, and I. Sutherland, "A VLSI Architecture for Updating Raster-Scan Displays," *SIGGRAPH 81*, 71–78.

GUPT86 Gupta, S., D.F. Bantz, P.N. Sholtz, C.J. Evangelisti, and W.R. DeOrazio, *YODA: An Advanced Display for Personal Computers*, Computer Science Research Report RC11618 (–52213), IBM Thomas J. Watson Research Center, Yorktown Heights, NY, October 1986.

GURW81 Gurwitz, R., R. Fleming and A. van Dam, "MIDAS: A Microprocessor Instructional Display and Animation System," *IEEE Transactions on Education*, February, 1981.

HAEU76 Haeusing, M., "Color Coding of Information on Electronic Displays," in *Proceedings of the Sixth Congress of the International Ergonomics Association*, 1976, 210–217.

HAGE86 Hagen, M., *Varieties of Realism*, Cambridge University Press, Cambridge, England, 1986.

HAGE88 Hagen, R.E., *An Algorithm for Incremental Anti-Aliased Lines and Curves*, Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, January 1988.

HAIN86 Haines, E.A., and D.P. Greenberg, "The Light Buffer: A Shadow-Testing Accelerator," *CG & A*, 6(9), September 1986, 6–16.

HAIN89 Haines, E., "Essential Ray Tracing Algorithms," in Glassner, A.S., ed., *An Introduction to Ray Tracing*, Academic Press, London, 1989, 33–77.

HALA68 Halas, J., and R. Manvell, *The Technique of Film Animation*, Hastings House, New York, 1968.

HALA73 Halas, J., ed., *Visual Scripting*, Hastings House, New York, 1973.

HALA82 Halasz, F., and T. Moran, "Analogy Considered Harmful," in *Proceedings of the Human Factors in Computer Systems Conference*, ACM, New York, 1982, 383–386.

HALL83 Hall, R.A., and D.P. Greenberg, "A Testbed for Realistic Image Synthesis," *CG & A*, 3(8), November 1983, 10–20.

HALL86 Hall, R., "Hybrid Techniques for Rapid Image Synthesis," in Whitted, T., and R. Cook, eds., *Image Rendering Tricks, Course Notes 16 for SIGGRAPH 86*, Dallas, TX, August 1986.

HALL89 Hall, R., *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, New York, 1989.

HAMI53 Hamilton, W.R., *Lectures on Quaternions: Containing a Systematic Statement of a New Mathematical Method; of Which the Principles Were Communicated in 1843 to the Royal Irish Academy; and Which Has Since Formed the Subject of Successive Courses of Lectures, Delivered in 1848 and Subsequent Years, in the Halls of Trinity College, Dublin: With Numerous Illustrative Examples*, Hodges and Smith, Dublin, 1853.

HAML77 Hamlin, G., Jr., and C.W. Gear, "Raster-Scan Hidden Surface Algorithm Techniques," *SIGGRAPH 77*, 206–213. Also in FREE80, 264–271.

HANA80 Hanau, P., and D. Lenorovitz, "Prototyping and Simulation Tools for User/Computer Dialogue Design," *SIGGRAPH 80*, 271–278.

HANR83 Hanrahan, P., "Ray Tracing Algebraic Surfaces," *SIGGRAPH 83*, 83–90.

HANR89 Hanrahan, P., "A Survey of Ray-Surface Intersection Algorithms," in Glassner, A.S., ed., *An Introduction to Ray Tracing*, Academic Press, London, 1989, 79–119.

HANS71 Hansen, W., "User Engineering Principles for Interactive Systems," in *FJCC 1971*, AFIPS Press, Montvale, NJ, 1971, 523–532.

HARR88 Harrington, S.J., and R.R. Buckley, *Interpress: The Source Book*, Brady, New York, 1988.

HART89 Hartson, R., and D. Hix, "Human–Computer Interface Development: Concepts and Systems," *ACM Computing Surveys*, 21(1), March 1989, 5–92.

HATF89 Hatfield, D., *Anti-Aliased, Transparent, and Diffuse Curves*, IBM Technical Computing Systems Graphics Report 0001, International Business Machines, Cambridge, MA, 1989.

HAYE83 Hayes, P., and Szekely, P., "Graceful Interaction Through the COUSIN Command Interface," *International Journal of Man–Machine Studies*, 19(3), September 1983, 285–305.

HAYE84 Hayes, P., "Executable Interface Definitions Using Form-Based Interface Abstractions," in *Advances in Computer-Human Interaction*, Hartson, H.R., ed., Ablex, Norwood, NJ, 1984.

HAYE85 Hayes, P., P. Szekely, and R. Lerner, "Design Alternatives for User Interface Management Systems Based on Experience with COUSIN," in *CHI '85 Proceedings*, ACM, New York, 1985, 169–175.

HECK82 Heckbert, P., "Color Image Quantization for Frame Buffer Display," *SIGGRAPH 82*, 297–307.

HECK84 Heckbert, P.S., and P. Hanrahan, "Beam Tracing Polygonal Objects," *SIGGRAPH 84*, 119–127.

HECK86a Heckbert, P.S., "Filtering by Repeated Integration," *SIGGRAPH 86*, 315–321.

HECK86b Heckbert, P.S., "Survey of Texture Mapping," *CG & A*, 6(11), November 1986, 56–67.

HEDG82 Hedgley, D.R., Jr., *A General Solution to the Hidden-Line Problem*, NASA Reference Publication 1085, NASA Scientific and Technical Information Branch, 1982.

HEME82 Hemenway, K., "Psychological Issues in the Use of Icons in Command Menus," in *Proceedings Human Factors in Computer Systems Conference*, ACM, New York, 1982 20–23.

HERO76 Herot, C., "Graphical Input Through Machine Recognition of Sketches," *SIGGRAPH 76*, 97–102.

HERO78 Herot, C., and G. Weinzapfel, "One-Point Touch Input of Vector Information for Computer Displays," *SIGGRAPH 78*, 210–216.

HERZ80 Herzog, B., "In Memoriam of Steven Anson Coons," *Computer Graphics*, 13(4), February 1980, 228–231.

HILL83 Hill, F.S., Jr., S. Walker, Jr., and F. Gao, "Interactive Image Query System Using Progressive Transmission," *SIGGRAPH 83*, 323–333.

HILL87 Hill, R., "Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction—The Sassafras UIMS," *ACM TOG*, 5(3), July 1986, 179–210.

HIRS70 Hirsch, R., "Effects of Standard vs. Alphabetical Keyboard Formats on Typing Performance," *Journal of Applied Psychology*, 54, December 1970, 484–490

HOBB89 Hobby, J. D., "Rasterizing Curves of Constant Width," *JACM*, 36(2), April 1989, 209–229.

HODG85 Hodges, L., and D. McAllister, "Stereo and Alternating-Pair Techniques for Display of Computer-Generated Images," *CG & A*, 5(9), September 1985, 38–45.

HOFF61 Hoffman, K., and R. Kunze, *Linear Algebra*, Prentice–Hall, Englewood Cliffs, NJ, 1961.

HOLL80 Holladay, T. M., "An Optimum Algorithm for Halftone Generation for Displays and Hard Copies," *Proceedings of the Society for Information Display*, 21(2), 1980, 185–192.

HOPG86a Hopgood, F., D. Duce, J. Gallop, and D. Sutcliffe, *Introduction to the Graphical Kernel System (GKS)*, second edition, Academic Press, London, 1986.

HOPG86b Hopgood, F., D. Duce, E. Fielding, K. Robinson, and A. Williams, eds., *Methodology of Window Management*, Springer-Verlag, New York, 1986.

HORN79 Horn, B.K.P., and R.W. Sjoberg, "Calculating the Reflectance Map," *Applied Optics*, 18(11), June 1979, 1770–1779.

HUBS82 Hubschman, H., and S.W. Zucker, "Frame-to-Frame Coherence and the Hidden Surface Computation: Constraints for a Convex World," *ACM TOG*, 1(2), April 1982, 129–162.

HUDS86 Hudson, S., and R. King, "A Generator of Direct Manipulation Office Systems," *ACM Transactions on Office Information Systems*, 4(2), April 1986, 132–163.

HUDS87 Hudson, S., "UIMS Support for Direct Manipulation Interfaces," *ACM SIGGRAPH Workshop on Software Tools for User Interface Management*, in *Computer Graphics*, 21(2), April 1987, 120–124.

HUDS88 Hudson, S., and R. King, "Semantic Feedback in the Higgens UIMS," *IEEE Transactions on Software Engineering*, 14(8), August 1988, 1188–1206.

HUGH89 Hughes, J., *Integer and Floating-Point Z-Buffer Resolution*, Department of Computer Science Technical Report, Brown University, Providence, RI, 1989.

HULL87 Hull, R., and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues," *ACM Computing Surveys*, 19(3), September 1987, 201–260.

HUNT78 Hunter, G.M., *Efficient Computation and Data Structures for Graphics*, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

HUNT79 Hunter, G.M. and K. Steiglitz, "Operations on Images Using Quad Trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, 1(2), April 1979, 145–153.

HUNT81 Hunter, G.M., *Geometrees for Interactive Visualization of Geology: An Evaluation*, System Science Department, Schlumberger-Doll Research, Ridgefield, CT, 1981.

HUNT87 Hunt, R.W., *The Reproduction of Colour*, fourth edition, Fountain Press, Tolworth, England, 1987.

HURL89 Hurley, D., and J. Sibert, "Modeling User Interface-Application Interactions," *IEEE Software*, 6(1), January 1989, 71–77.

HUTC86 Hutchins, E., J. Hollan, and D. Norman, "Direct Manipulation Interfaces,' in Norman, D., and S. Draper, eds., *User Centered System Design*, Erlbaum, Hillsdale, NJ, 1986, 87–124.

IES87 Illuminating Engineering Society, Nomenclature Committee, *ANSI/IES RP-16-1986: American National Standard: Nomenclature and Definitions for Illuminating Engineering*, Illuminating Engineering Society of North America, New York, 1987.

IMME86 Immel, D.S., M.F. Cohen, and D.P. Greenberg, "A Radiosity Method for Non-Diffuse Environments," *SIGGRAPH 86*, 133–142.

INFA85 Infante, C., "On the Resolution of Raster-Scanned CRT Displays," *Proceedings of the Society for Information Display*, 26(1), 1985, 23–36.

INGA81 Ingalls, D., "The SmallTalk Graphics Kernel," *BYTE*, 6(8), August 1981.

INTE85 Interaction Systems, Inc., *TK-1000 Touch System*, Interaction Systems, Inc., Newtonville, MA, 1985.

INTE88 International Standards Organization, *International Standard Information Processing Systems—Computer Graphics—Graphical Kernel System for Three Dimensions (GKS-3D) Functional Description*, ISO Document Number 8805:1988(E), American National Standards Institute, New York, 1988.

INTE89 Intel Corporation, *i860 Microprocessor Family Product Briefs*, Intel Corporation, Santa Clara, CA, 1989.

IRAN71 Irani, K., and V. Wallace, "On Network Linguistics and the Conversational Design of Queueing Networks," *JACM*, 18, October 1971, 616–629.

ISO International Standards Organization, *Information Processing Text and Office Systems Standard Page Description Language (SPDL)*, ISO Document Number JTC1 SC18/WG8N561, American National Standards Institute, New York.

JACK64 Jacks, E., "A Laboratory for the Study of Man–Machine Communication," in *FJCC 64*, AFIPS, Montvale, NJ, 1964, 343–350.

JACK80 Jackins, C., and S.L. Tanimoto, "Oct-Trees and Their Use in Representing Three-Dimensional Objects," *CGIP*, 14(3), November 1980, 249–270.

JACO83 Jacob, R., "Using Formal Specifications in the Design of the User-Computer Interface," *CACM*, 26(4), April 1983, 259–264.

JACO85 Jacob, R., "A State Transition Diagram Language for Visual Programming," *IEEE Computer*, 18(8), August 1985, 51–59.

JACO86 Jacob, R., "A Specification Language for Direct-Manipulation User Interfaces," *ACM TOG*, 5(4), October 1986, 283–317.

JANS85 Jansen, F.W., "A CSG List Priority Hidden Surface Algorithm," in C. Vandoni, ed., *Proceedings of Eurographics 85*, North-Holland, Amsterdam, 1985, 51–62.

JANS87 Jansen, F.W., *Solid Modelling with Faceted Primitives*, Ph.D. Thesis, Department of Industrial Design, Delft University of Technology, Netherlands, September 1987.

JARV76a Jarvis, J.F., C.N. Judice, and W.H. Ninke, "A Survey of Techniques for the Image Display of Continuous Tone Pictures on Bilevel Displays," *CGIP*, 5(1), March 1976, 13–40.

JARV76b Jarvis, J.F., and C.S. Roberts, "A New Technique for Displaying Continuous Tone Images on a Bilevel Display," *IEEE Trans.*, COMM-24(8), August 1976, 891–898.

JENK89 Jenkins, R.A., "New Approaches in Parallel Computing," *Computers in Physics*, 3(1), January–February 1989, 24–32.

JEVA89 Jevans, D.A., "A Review of Multi-Computer Ray Tracing," *Ray Tracing News*, 3(1), May 1989, 8–15.

JOBL78 Joblove, G.H., and D. Greenberg, "Color Spaces for Computer Graphics," *SIGGRAPH 78*, 20–27.

JOHN78 Johnson, S., and M. Lesk, "Language Development Tools," *Bell System Technical Journal*, 57(6,7), July–August 1978, 2155–2176.

JOHN82 Johnson, S.A., "Clinical Varifocal Mirror Display System at the University of Utah," in *Proceedings of SPIE*, 367, August 1982, 145–148.

JONE26 Jones, L., and E. Lowry, "Retinal Sensibility to Saturation Differences," *Journal of the Optical Society of America*, 13(25), 1926.

JOVA86 Jovanović, B., *Visual Programming of Functional Transformations in a Dynamic Process Visualization System*, Report GWU-IIST-86-22, Department of Computer Science, George Washington University, Washington, DC, 1986.

JOY86 Joy, K.I., and M.N. Bhetanabhotla, "Ray Tracing Parametric Surface Patches Utilizing Numerical Techniques and Ray Coherence," *SIGGRAPH 86*, 279–285.

JOY88 Joy, K., C. Grant, N. Max, and L. Hatfield, *Tutorial: Computer Graphics: Image Synthesis*, IEEE Computer Society, Washington, DC, 1988.

JUDD75 Judd, D., and G. Wyszecki, *Color in Business, Science, and Industry*, Wiley, New York, 1975.

JUDI74 Judice, J.N., J.F. Jarvis, and W. Ninke, "Using Ordered Dither to Display Continuous Tone Pictures on an AC Plasma Panel," *Proceedings of the Society for Information Display*, Q4 1974, 161–169.

KAJI82 Kajiya, J.T., "Ray Tracing Parametric Patches," *SIGGRAPH 82*, 245–254.

KAJI83 Kajiya, J., "New Techniques for Ray Tracing Procedurally Defined Objects," *SIGGRAPH 83*, 91–102.

KAJI84 Kajiya, J., and B. Von Herzen, "Ray Tracing Volume Densities," *SIGGRAPH 84*, 165–173.

KAJI85 Kajiya, J.T., "Anisotropic Reflection Models," *SIGGRAPH 85*, 15–21.

KAJI86 Kajiya, J.T., "The Rendering Equation," *SIGGRAPH 86*, 143–150.

KAPL85 Kaplan, G., and E. Lerner, "Realism in Synthetic Speech," *IEEE Spectrum*, 22(4), April 1985, 32–37.

KAPP85 Kappel, M.R., "An Ellipse-Drawing Algorithm for Raster Displays," in Earnshaw, R., ed. *Fundamental Algorithms for Computer Graphics*, NATO ASI Series, Springer-Verlag, Berlin, 1985, 257–280.

KASI82 Kasik, D., "A User Interface Management System," *SIGGRAPH 82*, 99–106.

KAUF88a Kaufmann, H. E., "User's Guide to the Compositor," Computer Graphics Group Documentation, Brown University, Providence, RI, May 1988.

KAUF88b Kaufman, A., and R. Bakalash, "Memory and Processing Architecture for 3D Voxel-Based Imagery," *CG & A*, 8(6), November 1988, 10–23.

KAWA82 Kawaguchi, Y., "A Morphological Study of the Form of Nature," *SIGGRAPH 82*, 223–232.

KAWA88 Kawaguchi, Y., film, *ACM SIGGRAPH 88 Electronic Theater and Video Review*, 26, 1988.

KAY79a Kay, D.S., *Transparency, Refraction and Ray Tracing for Computer Synthesized Images*, M.S. Thesis, Program of Computer Graphics, Cornell University, Ithaca, NY, January 1979.

KAY79b Kay, D.S., and D. Greenberg, "Transparency for Computer Synthesized Images," *SIGGRAPH 79*, 158–164.

KAY86 Kay, T.L., and J.T. Kajiya, "Ray Tracing Complex Scenes," *SIGGRAPH 86*, 269–278.

KEDE84 Kedem, G., and J. L. Ellis, "The Raycasting Machine," in *Proceedings of the 1984 International Conference on Computer Design*, October 1984, 533–538.

KELL76 Kelly, K., and D. Judd, *COLOR—Universal Language and Dictionary of Names*, National Bureau of Standards Spec. Publ. 440, 003-003-01705-1, U.S. Government

Printing Office, Washington, DC, 1976.

KELL88 Kelley, A.D., M.C. Malin, and G.M. Nielson, "Terrain Simulation Using a Model of Stream Erosion," *SIGGRAPH 88,* 263–268.

KIER85 Kieras, D., and P. Polson, "An Approach to the Formal Analysis of User Complexity," *International Journal of Man-Machine Studies,* 22(4), April 1985, 365–394.

KLEM71 Klemmer, E., "Keyboard Entry," *Applied Ergonomics,* 2(1), 1971, 2–6.

KLIN71 Klinger, A., "Patterns And Search Statistics," in Rustagi, J., ed., *Optimizing Methods in Statistics,* Academic Press, New York, 1971, 303–337.

KNOW77 Knowlton, K., and L. Cherry, "ATOMS—A Three-D Opaque Molecule System for Color Pictures of Space-Filling or Ball-and-Stick Models," *Computers and Chemistry,* 1, 1977, 161–166.

KNOW80 Knowlton, K., "Progressive Transmission of Gray-Scale and Binary Pictures by Simple, Efficient, and Loss-less Encoding Schemes," *Proc. of IEEE,* 68(7), 1980, 885–896.

KNUT69 Knuth, D.E., *The Art of Computer Programming, Volume 2: Seminumerical Algorithms,* Addison-Wesley, Reading, MA, 1969.

KNUT87 Knuth, D., "Digital Halftones by Dot Diffusion," *ACM TOG,* 6(4), October 1987, 245–273.

KOBA87 Kobayashi, H., T. Nakamura, and Y. Shigei, "Parallel Processing of an Object Synthesis Using Ray Tracing," *Visual Computer,* 3(1), February 1987, 13–22.

KOCA87 Koçak, H., Bisshopp, F., Laidlaw, D., and T. Banchoff, "Topology and Mechanics with Computer Graphics: Linear Hamiltonian Systems in Four Dimensions," *Advances in Applied Mathematics,* 1986, 282–308.

KOCH84 Kochanek, D., and R. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control," *SIGGRAPH 84,* 33–41.

KOIV88 Koivunen, M., and M. Mäntylä, "HutWindows: An Improved Architecture for a User Interface Management System," *CG & A,* 8(1), January 1988, 43–52.

KORE82 Korein, J.U., and N.I. Badler, "Techniques for Generating the Goal-Directed Motion of Articulated Structures," *CG & A,* 2(11), November 1982, 71–81.

KORE83 Korein, J., and N. Badler, "Temporal Anti-Aliasing in Computer Generated Animation," *SIGGRAPH 83,* 377–388.

KREB79 Krebs, M., and J. Wolf, "Design Principles for the Use of Color in Displays," *Proceedings of the Society for Information Display,* 20, 1979, 10–15.

KRUE83 Krueger, M., *Artificial Reality,* Addison-Wesley, Reading, MA, 1983.

KURL88 Kurlander, D., and S. Feiner, "Editable Graphical Histories," in *Proc. 1988 IEEE Workshop on Visual Languages,* October 10–12, 1988, Pittsburgh, PA, 129–132.

KURL90 Kurlander, D., and S. Feiner, "A Visual Language for Browsing, Undoing, and Redoing Graphical Interface Commands," in Chang, S., ed., *Visual Languages and Visual Programming,* Plenum Press, New York, 1990, 257–275.

LAID86 Laidlaw, D.H., W.B. Trumbore, and J.F. Hughes, "Constructive Solid Geometry for Polyhedral Objects," *SIGGRAPH 86,* 161–170.

LAND85 Landauer, T., and D. Nachbar, "Selection from Alphabetic and Numeric Menu Trees Using a Touch-Sensitive Screen: Breadth, Depth, and Width," in *Proceedings CHI '85 Human Factors in Computing Systems Conference,* ACM, New York, 1985, 73–78.

LANE79 Lane, J., and L. Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *CGIP,* 11(3), November 1979, 290–297.

LANE80a Lane, J., and R. Riesenfeld, "A Theoretical Development for the Computer Generation of Piecewise Polynomial Surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* PAMI-2(1), January 1980, 35–46.

LANE80b Lane, J., L. Carpenter, T. Whitted, and J. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *CACM*, 23(1), January 1980, 23–34. Also in BEAT82, 468–479.

LANT84 Lantz, K., and W. Nowicki, "Structured Graphics for Distributed Systems," *ACM TOG*, 3(1), January 1984, 23–51.

LANT87 Lantz, K., P. Tanner, C. Binding, K. Huang, and A. Dwelly, "Reference Models, Window Systems, and Concurrency," in Olsen, D., ed., "ACM SIGGRAPH Workshop on Software Tools for User Interface Management," *Computer Graphics*, 21(2), April 1987, 87–97.

LASS87 Lasseter, J., "Principles of Traditional Animation Applied to 3D Computer Animation," *SIGGRAPH 87*, 35–44.

LAYB79 Laybourne, K., *The Animation Book*, Crown, New York, 1979.

LEAF74 Leaf, C., *The Owl Who Married a Goose*, film, National Film Board of Canada, 1974.

LEAF77 Leaf, C., *The Metamorphosis of Mr. Samsa*, film, National Film Board of Canada, 1977.

LEE85a Lee, S., W. Buxton, and K. Smith, "A Multi-touch Three Dimensional Touch-sensitive Tablet," in *Proceedings of CHI'85 Human Factors in Computing Systems Conference*, ACM, New York, 1985, 21–25.

LEE85b Lee, M.E., R.A. Redner, and S.P. Uselton, "Statistically Optimized Sampling for Distributed Ray Tracing," *SIGGRAPH 85*, 61–67.

LEVI76 Levin, J., "A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadric Surfaces," *CACM*, 19(10), October 1976, 555–563.

LEVI84 Levinthal, A., and T. Porter, "Chap—a SIMD Graphics Processor," *SIGGRAPH 84*, 77–82.

LEVO78 Levoy, M., *Computer Assisted Cartoon Animation*, Master's Thesis, Department of Architecture, Cornell University, Ithaca, NY, August 1978.

LEVO82 Levoy, M., "Area Flooding Algorithms," in *Two-Dimensional Computer Animation, Course Notes 9 for SIGGRAPH 82*, Boston, MA, July 26–30, 1982.

LEVO89 Levoy, M., "Design for a Real-Time High-Quality Volume Rendering Workstation," in *Proceedings of the Volume Visualization Workshop*, Department of Computer Science, University of North Carolina at Chapel Hill, May 18-19, 1989, 85–90.

LIAN83 Liang, Y-D., and B.A. Barsky, "An Analysis and Algorithm for Polygon Clipping," *CACM*, 26(11), November 1983, 868–877, and Corrigendum, *CACM*, 27(2), February 1984, 151.

LIAN84 Liang, Y-D., and Barsky, B., "A New Concept and Method for Line Clipping," *ACM TOG*, 3(1), January 1984, 1–22.

LIEB78 Lieberman, H., "How to Color in a Coloring Book," *SIGGRAPH 78*, 111–116.

LIEN87 Lien, S.L., M. Shantz, and V. Pratt, "Adaptive Forward Differencing for Rendering Curves and Surfaces," *SIGGRAPH 87*, 111–118.

LIND68 Lindenmayer, A, "Mathematical Models for Cellular Interactions in Development, Parts I and II," *J. Theor. Biol.*, 18, 1968, 280–315.

LINT89 Linton, M., J. Vlissides, and P. Calder, "Composing User Interfaces with InterViews," *IEEE Computer*, 22(2), February 1989, 8–22.

LIPS79 Lipscomb, J.S., *Three-Dimensional Cues for a Molecular Computer Graphics System*, Ph.D. Thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1979.

LOUT70 Loutrel, P.P., "A Solution to the Hidden-Line Problem for Computer-Drawn Polyhedra," *IEEE Trans. on Computers*, EC-19(3), March 1970, 205–213. Also in FREE80, 221–229.

LUCA84 Lucasfilm, Ltd., *The Adventures of André and Wally B.*, film, August 1984.

MACH78 Machover, C., "A Brief Personal History of Computer Graphics," *Computer*, 11(11), November 1978, 38–45.

MACK86 Mackinlay, J., "Automating the Design of Graphical Presentation of Relational Information," *ACM TOG*, 5(2), April 1986, 110–141.

MAGI68 Mathematical Applications Group, Inc., "3-D Simulated Graphics Offered by Service Bureau," *Datamation*, 13(1), February 1968, 69.

MAGN85 Magnenat-Thalmann, N., and Thalmann, D., *Computer Animation: Theory and Practice*, Springer-Verlag, Tokyo, 1985.

MAHL72 Mahl, R., "Visible Surface Algorithms for Quadric Patches," *IEEE Trans. on Computers*, C-21(1), January 1972, 1–4.

MAHN73 Mahnkopf, P., and J.L. Encarnação, *FLAVIS—A Hidden Line Algorithm for Displaying Spatial Constructs Given by Point Sets*, Technischer Bericht Nr. 148, Heinrich Hertz Institut, Berlin, 1973.

MAMM89 Mammen, A., "Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique," *CG & A*, 9(4), July 1989, 43–55.

MAND77 Mandelbrot, B., *Fractals: Form, Chance and Dimension*, W.H. Freeman, San Francisco, CA, 1977.

MAND82 Mandelbrot, B., Technical Correspondence, *CACM*, 25(8), August 1982, 581–583.

MÄNT88 Mäntylä, M. *Introduction to Solid Modeling*, Computer Science Press, Rockville, MD, 1988.

MARC80 Marcus, A., "Computer-Assisted Chart Making from the Graphic Designer's Perspective," *SIGGRAPH 80*, 247–253.

MARC82 Marcus, A., "Color: A Tool for Computer Graphics Communication," in Greenberg, D., A. Marcus, A. Schmidt, and V. Gorter, *The Computer Image*, Addison-Wesley, Reading, MA, 1982, 76–90.

MARC84 Marcus, A., "Corporate Identity for Iconic Interface Design: The Graphic Design Perspective," *CG & A*, 4(12), December 1984, 24–32.

MARK80 Markowsky, G., and M.A. Wesley, "Fleshing Out Wire Frames," *IBM Journal of Research and Development*, 24(5), September 1980, 582–597.

MARS85 Marsden, J., and A. Weinstein, *Calculus I, II, and III*, second edition, Springer Verlag, New York, 1985.

MART89 Martin, G., "The Utility of Speech Input in User–Computer Interfaces," *International Journal of Man–Machine Studies*, 30(4), April 1989, 355–376.

MASS85 Massachusetts Computer Corporation (MASSCOMP), *Graphics Application Programming Manual*, Order No. M-SP40-AP, MASSCOMP, Westford, MA, 1985.

MAUL89 Maulsby, D., and I. Witten, "Inducing Programs in a Direct Manipulation Environment," in *Proceedings CHI 1989*, ACM, New York, 1989, 57–62.

MAX79 Max, N.L., "ATOMLLL: - ATOMS with Shading and Highlights," *SIGGRAPH 79*, 165–173.

MAX81 Max, N., *Carla's Island*, animation, *ACM SIGGRAPH 81 Video Review*, 5, 1981.

MAX82 Max, N., "SIGGRAPH '84 Call for Omnimax Films," Computer Graphics, 16(4), December 1982, 208–214.

MAX84 Max, N.L., "Atoms with Transparency and Shadows," *CVGIP*, 27(1), July 1984, 46–63.

MAX86 Max, N., "Atmospheric Illumination and Shadows," *SIGGRAPH 86*, 117–124.

MAXW46 Maxwell, E. A., *Methods of Plane Projective Geometry Based on the Use of General*

Homogeneous Coordinates, Cambridge University Press, Cambridge, England, 1946.

MAXW51 Maxwell, E. A., *General Homogeneous Coordinates in Space of Three Dimensions*, Cambridge University Press, Cambridge, England, 1951.

MAYH90 Mayhew, D., *Principles and Guidelines in User Interface Design*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

MCIL83 McIlroy, M.D., "Best Approximate Circles on Integer Grids," *ACM TOG*, 2(4), October 1983, 237–263.

MCLE88 McLeod, J., "HP Delivers Photo Realism on an Interactive System," *Electronics*, 61(6), March 17, 1988, 95–97.

MCMI87 McMillan, L., "Graphics at 820 MFLOPS," *ESD: The Electronic Systems Design Magazine*, 17(9), September 1987, 87–95.

MEAG80 Meagher, D., *Octree Encoding: A New Technique for the Representation, Manipulation, and Display of Arbitrary 3-D Objects by Computer*, Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, NY, October 1980.

MEAG82a Meagher, D., "Geometric Modeling Using Octree Encoding," *CGIP*, 19(2), June 1982, 129–147.

MEAG82b Meagher, D., "Efficient Synthetic Image Generation of Arbitrary 3-D Objects," in *Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing*, IEEE Computer Socitey Press, Washington, DC, 1982.

MEAG84 Meagher, D., "The Solids Engine: A Processor for Interactive Solid Modeling," in *Proceedings of NICOGRAPH '84*, Tokyo, November 1984.

MEAG85 Meagher, D., "Applying Solids Processing to Medical Planning," in *Proceedings of NCGA '85*, Dallas, 1985, 101–109.

MEGA89 Megatek Corporation, *Sigma 70 Advanced Graphics Workstations*, Megatek Corporation, San Diego, CA, 1989.

MEIE83 Meier, B., *Brim*, Computer Graphics Group Documentation, Computer Science Department, Brown University, Providence, RI, 1983.

MEIE88 Meier, B., "ACE: A Color Expert System for User Interface Design," in *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 117–128, 1988.

MEYE80 Meyer, G.W., and D.P. Greenberg, "Perceptual Color Spaces for Computer Graphics," *SIGGRAPH 80*, 254–261.

MEYE83 Meyer, G., *Colorimetry and Computer Graphics*, Program of Computer Graphics, Cornell University, Ithaca, NY, 1983.

MEYE88 Meyer, G., and Greenberg, D., "Color-defective Vision and Computer Graphic Displays," *CG & A*, 8(5), September 1988, 28–40.

MICH71 Michaels, S., "QWERTY Versus Alphabetical Keyboards as a Function of Typing Skill," *Human Factors*, 13(5), October 1971, 419–426.

MICR89 Microsoft Corporation, *Presentation Manager*, Microsoft Corporation, Bellevue, WA, 1989.

MILL87 Miller, J.R., "Geometric Approaches to Nonplanar Quadric Surface Intersection Curves," *ACM TOG*, 6(4), October 1987, 274–307.

MILL88a Miller, G.S.P., "The Motion Dynamics of Snakes and Worms," *SIGGRAPH 88*, 169–178.

MILL88b Miller, G.S.P., "The Motion Dynamics of Snakes and Worms," lecture at ACM SIGGRAPH '88.

MILL88c Miller, P., and M. Szczur, "Transportable Application Environment (TAE) Plus

Experiences in 'Object'ively Modernizing a User Interface Environment," in *Proceedings of OOPSLA '88*, 58–70.

MILL89 Miller, J.R., "Architectural Issues in Solid Modelers," *CG & A*, 9(5), September 1989, 72–87.

MINS84 Minsky, M., "Manipulating Simulated Objects with Real-World Gestures Using a Force and Position Sensitive Screen," *SIGGRAPH 84*, 195–203.

MITC87 Mitchell, D.P., "Generating Antialiased Images at Low Sampling Densities," *SIGGRAPH 87*, 65–72.

MITC88 Mitchell, D.P., and A.N. Netravali, "Reconstruction Filters in Computer Graphics," *SIGGRAPH 88*, 221–228.

MOLN88 Molnar, S., "Combining Z-Buffer Engines for Higher-Speed Rendering," 1988 Eurographics Workshop on Graphics Hardware, Sophia-Antipolis, France, September, 1988. To appear in Kuijk, A.A.M., ed., *Advances in Computer Graphics Hardware III*, Proceedings of 1988 Eurographics Workshop on Graphics Hardware, Eurographics Seminars, Springer-Verlag, Berlin, 1989.

MORR86 Morris, J., M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S.H. Rosenthal, and F.D. Smith, "Andrew: A Distributed Personal Computing Environment," *CACM*, 29(3), March 1986, 184–201.

MORT85 Mortenson, M., *Geometric Modeling*, Wiley, New York, 1985.

MUNS76 Munsell Color Company, *Book of Color*, Munsell Color Company, Baltimore, MD, 1976.

MURC85 Murch, G., "Using Color Effectively: Designing to Human Specifications," *Technical Communications*, Q4 1985, Tektronix Corporation, Beaverton, OR, 14–20.

MUSG89 Musgrave, F.K., "Prisms and Rainbows: A Dispersion Model for Computer Graphics," in *Proceedings of Graphics Interface '89*, London, Ontario, June 19–23, 1989, 227–234.

MYER68 Myer, T., and I. Sutherland, "On the Design of Display Processors," *CACM*, 11(6), June 1968, 410–414.

MYER75 Myers, A.J., *An Efficient Visible Surface Program*, Report to the National Science Foundation, Computer Graphics Research Group, Ohio State University, Columbus, OH, July 1975.

MYER84 Myers, B., "The User Interface for Sapphire," *CG & A*, 4(12), December 1984, 13–23.

MYER85 Myers, B., "The Importance of Percent-Done Progress Indicators for Computer-Human Interfaces," in *Proceedings CHI '85*, ACM, New York, 1985, 11–17.

MYER86 Myers, B., "Creating Highly-Interactive and Graphical User Interfaces by Demonstration," *SIGGRAPH 86*, 249–257.

MYER88 Myers, B., *Creating User Interfaces by Demonstration*, Academic Press, New York, 1988.

MYER89 Myers, B., "User-Interface Tools: Introduction and Survey," *IEEE Software*, 6(1), January 1989, 15–23.

NAIM87 Naiman, A., and A. Fournier, "Rectangular Convolution for Fast Filtering of Characters," *SIGGRAPH 87*, 233–242.

NARU87 Naruse, T., M. Yoshida, T. Takahashi, and S. Naito, "SIGHT—a Dedicated Computer Graphics Machine," *Computer Graphics Forum*, 6(4), December 1987, 327–334.

NAVA89 Navazo, I., "Extended Octree Representation of General Solids with Plane Faces: Model Structure and Algorithms," *Computers and Graphics*, 13(1), January 1989, 5–16.

NAYL90 Naylor, B.F., "Binary Space Partitioning Trees as an Alternative Representation of

Polytopes," *CAD*, 22(4), May 1990, 250–253.

NEMO86 Nemoto, K., and T. Omachi, "An Adaptive Subdivision by Sliding Boundary Surfaces for Fast Ray Tracing," in *Proceedings of Graphics Interface '86*, 1986, 43–48.

NEWE72 Newell, M.E., R.G. Newell, and T.L. Sancha, "A Solution to the Hidden Surface Problem," in *Proceedings of the ACM National Conference 1972*, 443–450. Also in FREE80, 236–243.

NEWE74 Newell, M.E., *The Utilization of Procedure Models in Digital Image Synthesis*, Ph.D. Thesis, Technical Report UTEC-CSc-76-218, NTIS AD/A039 008/LL, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.

NEWM68 Newman, W., "A System for Interactive Graphical Programming," *SJCC*, Thompson Books, Washington, DC, 1968, 47–54.

NEWM71 Newman, W. M., "Display Procedures," *CACM*, 14(10), 1971, 651–660.

NEWM73 Newman, W., and R. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, 1973.

NEWM79 Newman, W., and R. Sproull, *Principles of Interactive Computer Graphics*, 2nd ed., McGraw-Hill, New York, 1979.

NICH87 Nicholl, T.M., D.T. Lee, and R.A. Nicholl, "An Efficient New Algorithm for 2-D Line Clipping: Its Development and Analysis," *SIGGRAPH 87*, 253–262.

NICO77 Nicodemus, F.E., J.C. Richmond, J.J. Hsia, I.W. Ginsberg, and T. Limperis, *Geometrical Considerations and Nomenclature for Reflectance*, NBS Monograph 160, U.S. Department of Commerce, Washington DC, October 1977.

NIEL86 Nielson, G., and D. Olsen, Jr., "Direct Manipulation Techniques for 3D Objects Using 2D Locator Devices," in *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, ACM, New York, 1987, 175–182.

NISH83a Nishimura, H., H. Ohno, T. Kawata, I. Shirakawa, and K. Omura, "LINKS-1: A Parallel Pipelined Multimicrocomputer System for Image Creation," in *Proceedings of the Tenth International Symposium on Computer Architecture, ACM SIGARCH Newsletter*, 11(3), 1983, 387–394.

NISH83b Nishita, T. and E. Nakamae, "Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources," *Proc. IEEE Computer Society International Computer Software and Applications Conference (COMPSAC)*, IEEE Computer Society, Washington, DC, November 1983, 237–241.

NISH85a Nishita, T., and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection," *SIGGRAPH 85*, 23–30.

NISH85b Nishita, T., I. Okamura, and E. Nakamae, "Shading Models for Point and Linear Sources," *ACM TOG*, 4(2), April 1985, 124–146.

NISH86 Nishita, T., and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Illuminated by Sky Light," *SIGGRAPH 86*, 125–132.

NISH87 Nishita, T., Y. Miyawaki, and E. Nakamae, "A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources," *SIGGRAPH 87*, 303–310.

NOLL67 Noll, M., "A Computer Technique for Displaying N-dimensional Hyperobjects," *CACM*, 10(8), August 1967, 469–473.

NORM88 Norman, D., *The Psychology of Everyday Things*, Basic Books, New York, 1988.

OKIN84 Okino, N., Y. Kakazu, and M. Morimoto, "Extended Depth-Buffer Algorithms for Hidden Surface Visualization," *CG & A*, 4(5), May 1984, 79–88.

OLIV85 Oliver, M., "Display Algorithms for Quadtrees and Octtrees and their Hardware Realisation," in Kessener, L., F. Peters, and M. van Lierop, eds., *Data Structures for Raster Graphics*, Springer-Verlag, Berlin, 1986, 9–37.

OLSE83 Olsen, D., and E. Dempsey, "SYNGRAPH: A Graphical User Interface Generator," *SIGGRAPH 83*, 43–50.

OLSE84a Olsen, D., "Pushdown Automata for User Interface Management," *ACM TOG*, 3(3), July 1984, 177–203.

OLSE84b Olsen, D., W. Buxton, R. Ehrich, D. Kasik, J. Rhyne, and J. Sibert, "A Context for User Interface Management," *CG & A*, 4(12), December 1984, 33–42.

OLSE86 Olsen, D., "MIKE: The Menu Interaction Kontrol Environment," *ACM TOG*, 5(4), October 1986, 318–344.

OLSE87 Olsen, D., ed., ACM SIGGRAPH Workshop on Software Tools for User Interface Management, *Computer Graphics*, 21(2), April 1987, 71–147.

OLSE88 Olsen, D., "Macros by Example in a Graphical UIMS," *CG & A*, 8(1), January 1988, 68–78.

OLSE89 Olsen, D., "A Programming Language Basis for User Interface Management," in *Proceedings CHI '89*, ACM, New York, 1989, 171–176.

OPEN89a Open Software Foundation, *OSF/MOTIF™ Manual*, Open Software Foundation, Cambridge, MA, 1989.

OPEN89b Open Software Foundation, *OSF/MOTIF™ Style Guide*, Open Software Foundation, Cambridge, MA, 1989.

OSTW31 Ostwald, W., *Colour Science*, Winsor & Winsor, London, 1931.

PAET86 Paeth, A.W., "A Fast Algorithm for General Raster Rotation," in *Proceedings Graphics Interface '86*, Canadian Information Processing Society, 1986, 77–81.

PAET89 Paeth, A. W., "Fast Algorithms for Color Correction," *Proceedings of the Society for Information Display*, 30(3), Q3 1989, 169–175, reprinted as Technical Report CS-89-42, Department of Computer Science, University of Waterloo, Waterloo, Canada, 1989.

PAIN89 Painter, J., and K. Sloan, "Antialiased Ray Tracing by Adaptive Progressive Refinement," *SIGGRAPH 89*, 281–288.

PALA88 Palay, A., W. Hansen, M. Kazar, M. Sherman, M. Wadlow, T. Neuendorffer, Z. Stern, M. Bader, and T. Peters, "The Andrew Toolkit: An Overview," in *Proceedings 1988 Winter USENIX*, February 1988, 9–21.

PARK80 Parke, F., "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems," *SIGGRAPH 80*, 48–56.

PARK82 Parke, F.I., "Parameterized Models for Facial Animation," *CG & A*, 2(11), November 1982, 61–68.

PARK88 Parker, R., *Looking Good in Print: A Guide to Basic Design for Desktop Publishing*, Ventana Press, Chapel Hill, NC, 1988.

PAVL81 Pavlidis, T. "Contour Filling in Raster Graphics," *CGIP*, 10(2), June 1979, 126–141.

PEAC85 Peachey, D.R., "Solid Texturing of Complex Surfaces," *SIGGRAPH 85*, 279–286.

PEAR86 Pearson, G., and M. Weiser, "Of Moles and Men: The Design of Foot Controls for Workstations," in *Proceedings CHI '86*, ACM, New York, 1986, 333–339.

PEAR88 Pearson, G., and M. Weiser, "Exploratory Evaluation of a Planar Foot-operated Cursor Positioning Device," in *Proceedings CHI '88*, ACM, New York, 1988, 13–18.

PEIT86 Peitgen, H.-O., and P.H. Richter, *The Beauty of Fractals: Images of Complex Dynamical Systems*, Springer-Verlag, Berlin, 1986.

PEIT88 Peitgen, H.-O., and D. Saupe, eds., *The Science of Fractal Images*, Springer-Verlag, New York, 1988.

PERL85 Perlin, K., "An Image Synthesizer," *SIGGRAPH 85*, 287–296.

PERL89 Perlin, K., and E. Hoffert, "Hypertexture," *SIGGRAPH 89*, 253–262.

PERR85 Perry, T., and P. Wallach, "Computer Displays: New Choices, New Tradeoffs," *IEEE Spectrum*, 22(7), July 1985, 52–59.

PERR89 Perry, T., and J. Voelcker, "Of Mice and Menus: Designing the User-Friendly Interface," *IEEE Spectrum*, 26(9), September 1989, 46–51.

PERS85 Personics Corporation, *View Control System*, Concord, MA, 1985.

PETE86 Peterson, J.W., R.G. Bogart, and S.W. Thomas, *The Utah Raster Toolkit*, University of Utah, Department of Computer Science, Salt Lake City, UT, 1986.

PHIG88 PHIGS+ Committee, Andries van Dam, chair, "PHIGS+ Functional Description, Revision 3.0," *Computer Graphics*, 22(3), July 1988, 125–218.

PIKE83 Pike, R., "Graphics in Overlapping Bitmap Layers," *ACM TOG*, 17(3), July 83, 331–356.

PIKE84 Pike, R., "Bitmap Graphics," in *Course Notes 4 for SIGGRAPH 84*, Minneapolis, MN, July 23–27,1984.

PINK83 Pinkham, R., M. Novak, and K. Guttag, "Video RAM Excels at Fast Graphics," *Electronic Design*, 31(17), Aug. 18, 1983, 161–182.

PITT67 Pitteway, M.L.V., "Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter," *Computer J.*, 10(3), November 1967, 282–289.

PITT80 Pitteway, M.L.V., and D. J. Watkinson, "Bresenham's Algorithm with Grey-Scale," *CACM*, 23(11), November 1980, 625–626.

PITT87 Pitteway, M.L.V., "Soft Edging Fonts," Computer Graphics Technology and Systems, in *Proceedings of the Conference Held at Computer Graphics '87*, London, October 1987, Advanced computing series, 9, Online Publications, London, 1987.

PIXA86 Pixar Corporation, *Luxo, Jr.*, film, Pixar Corporation, San Rafael, CA, 1986.

PIXA88 Pixar Corporation, *The RenderMan Interface*, Version 3.0, Pixar Corporation, San Rafael, CA, May 1988.

PLAT81 Platt, S.M., and N.I. Badler, "Animating Facial Expressions," *SIGGRAPH 81*, 245–252.

PLAT88 Platt, J.C., and A.H. Barr, "Constraint Methods for Flexible Models," *SIGGRAPH 88*, 279–288.

PORT79 Porter, T., "The Shaded Surface Display of Large Molecules," *SIGGRAPH 79*, 234–236.

PORT84 Porter, T., and T. Duff, "Compositing Digital Images," *SIGGRAPH 84*, 253–259.

POSC89 Posch, K.C., and W.D. Fellner, "The Circle-Brush Algorithm," *ACM TOG*, 8(1), January 1989, 1–24.

POTM82 Potmesil, M., and I. Chakravarty, "Synthetic Image Generation with a Lens and Aperture Camera Model," *ACM TOG*, 1(2), April 1982, 85–108.

POTM83 Potmesil, M., and I. Chakravarty, "Modeling Motion Blur in Computer-Generated Images," *SIGGRAPH 83*, 389–399.

POTM89 Potmesil, M., and E. Hoffert, "Pixel Machine: A Parallel Image Computer," *SIGGRAPH 89*, 69–78.

POTT88 Potter, R., L. Weldon, and B. Shneiderman, "Improving the Accuracy of Touch Screens: An Experimental Evaluation of Three Strategies," in *Proceedings CHI '88*, ACM, New York, 27–32.

PRAT84 Pratt, M., "Solid Modeling and the Interface Between Design and Manufacture," *CG & A*, 4(7), July 1984, 52–59.

PRAT85 Pratt, V., "Techniques for Conic Splines," *SIGGRAPH 85*, 151–159.

PREP85 Preparata, F. P., and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

PRES88 Press, W.H., B.P. Flannery, S.A. Teukolskym, and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, England, 1988.

PRIN71 Prince, D., *Interactive Graphics for Computer Aided Design*, Addison-Wesley, Reading, MA, 1971.

PRIT77 Pritchard, D.H., "U.S. Color Television Fundamentals—A Review," *IEEE Transactions on Consumer Electronics*, CE-23(4), November 1977, 467–478.

PRUS88 Prusinkiewicz, P., A. Lindenmayer, and J. Hanan, "Developmental Models of Herbaceous Plants for Computer Imagery Purposes," *SIGGRAPH 88*, 141–150.

PUTN86 Putnam, L.K., and P.A. Subrahmanyam, "Boolean Operations on n-Dimensional Objects," *CG & A*, 6(6), June 1986, 43–51.

QUIN82 Quinlan, K.M., and J.R. Woodwark, "A Spatially-Segmented Solids Database—Justification and Design," in *Proc. CAD '82 Conf.*, Fifth International Conference and Exhibit on Computers in Design Engineering, Mar. 30–Apr 1, 1982, Butterworth, Guildford, Great Britain, 1982, 126–132.

RATL72 Ratliff, F., "Contour and Contrast," *Scientific American*, 226(6), June 1972, 91–101. Also in BEAT82, 364–375.

REDD78 Reddy, D., and S. Rubin, *Representation of Three-Dimensional Objects*, CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1978.

REEV81 Reeves, W.T., "Inbetweening for Computer Animation Utilizing Moving Point Constraints," *SIGGRAPH 81*, 263–269.

REEV83 Reeves, W.T., "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects," *SIGGRAPH 83*, 359–376.

REEV85 Reeves, W.T., and R. Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Particle Systems," *SIGGRAPH 85*, 313–322.

REEV87 Reeves, W.T., D.H. Salesin, and R.L. Cook, "Rendering Antialiased Shadows with Depth Maps," *SIGGRAPH 87*, 283–291.

REFF88 de Reffye, P., C. Edelin, J. Françon, M. Jaeger, and C. Puech, "Plant Models Faithful to Botanical Structure and Development," *SIGGRAPH 88*, 151–158.

REIS82 Reisner, P., "Further Developments Toward Using Formal Grammar as a Design Tool," in *Proceedings of the Human Factors in Computer Systems Conference*, ACM, New York, 1982, 304–308.

REQU77 Requicha, A.A.G., *Mathematical Models of Rigid Solids*, Tech. Memo 28, Production Automation Project, University of Rochester, Rochester, NY, 1977.

REQU80 Requicha, A.A.G., "Representations for Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys*, 12(4), December 1980, 437–464.

REQU82 Requicha, A.A.G., and H.B. Voelcker, "Solid Modeling: A Historical Summary and Contemporary Assessment," *CG & A*, 2(2), March 1982, 9–24.

REQU83 Requicha, A.A.G., and H.B. Voelcker, "Solid Modeling: Current Status and Research Directions," *CG & A*, 3(7), October 1983, 25–37.

REQU84 Requicha, A.A.G., "Representation of Tolerances in Solid Modeling: Issues and Alternative Approaches," in Pickett, M., and J. Boyse, eds., *Solid Modeling by Computers*, Plenum Press, New York, 1984, 3–22.

REQU85 Requicha, A.A.G., and H.B. Voelcker, "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms," *Proc. IEEE*, 73(1), January 1985, 30–44.

REYN82 Reynolds, C.W., "Computer Animation with Scripts and Actors," *SIGGRAPH 82*, 289–296.

REYN87 Reynolds, C.W., "Flocks, Herds and Schools: A Distributed Behavioral Model," *SIGGRAPH 87*, 25–34.

RHOD89 Rhoden, D., and C. Wilcox, "Hardware Acceleration for Window Systems," *SIGGRAPH 89*, 61–67.

RHYN87 Rhyne, J., "Dialogue Management for Gestural Interfaces," *Proceedings ACM SIGGRAPH Workshop on Tools for User Interface Management*, in *Computer Graphics*, 21(2), April 1987, 137–145.

ROBE63 Roberts, L.G., *Machine Perception of Three Dimensional Solids*, Lincoln Laboratory, TR 315, MIT, Cambridge, MA, May 1963. Also in Tippet, J.T., *et al.*, eds., *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge, MA, 1964, 159–197.

ROBE65 Roberts, L.G., *Homogeneous Matrix Representations and Manipulation of* N-*Dimensional Constructs*, Document MS 1405, Lincoln Laboratory, MIT, Cambridge, MA, 1965.

ROGE85 Rogers, D.F., *Procedural Elements for Computer Graphics*, McGraw-Hill, New York, 1985.

ROGO83 Rogowitz, B., "The Human Visual System: A Guide for the Display Technologist," *Proceedings Society for Information Display*, 24(3), 1983.

ROMN69 Romney, G.W., G.S. Watkins, and D.C. Evans, "Real Time Display of Computer Generated Half-Tone Perspective Pictures," in *Proceedings 1968 IFIP Congress*, North Holland Publishing Co., 1969, 973–978.

ROSE83 Rosenthal, D., "Managing Graphical Resources," *Computer Graphics*, 17(1), January 1983, 38–45.

ROSE85 Rose, C., B. Hacker, R. Anders, K. Wittney, M. Metzler, S. Chernicoff, C. Espinosa, A. Averill, B. Davis, and B. Howard, *Inside Macintosh*, I, Addison-Wesley, Reading, MA, 1985, I-35–I-213.

ROSS86 Rossignac, J.R., and A.A.G. Requicha, "Depth-Buffering Display Techniques for Constructive Solid Geometry," *CG & A*, 6(9), September 1986, 29–39.

ROSS89 Rossignac, J., and H. Voelcker, "Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection, and Shading Algorithms," *ACM TOG*, 8(1), January 1989, 51–87.

ROTH76 Rothstein, J., and C.F.R. Weiman, "Parallel and Sequential Specification of a Context Sensitive Language for Straight Lines on Grids," *CGIP*, 5(1), March 1976, 106–124.

ROTH82 Roth, S., "Ray Casting for Modeling Solids," *CGIP*, 18(2), February 1982, 109–144.

RUBE83 Rubel, A., "Graphic Based Applications—Tools to Fill the Software Gap," *Digital Design*, 3(7), July 1983, 17–30.

RUBE84 Rubenstein, R., and H. Hersh, *The Human Factor—Designing Computer Systems for People*, Digital Press, Burlington, MA, 1984.

RUBE88 Rubenstein, R., *Digital Typography*, Addison-Wesley, Reading, MA, 1988.

RUBI80 Rubin, S.M., and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *SIGGRAPH 80*, 110–116.

RUSH86 Rushmeier, H.E., *Extending the Radiosity Method to Transmitting and Specularly Reflecting Surfaces*, M.S. Thesis, Mechanical Engineering Department, Cornell University, Ithaca, NY, 1986.

RUSH87 Rushmeier, H., and K. Torrance, "The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium," *SIGGRAPH 87*, 293–302.

SABE88 Sabella, P., "A Rendering Algorithm for Visualizing 3D Scalar Fields," *SIGGRAPH 88*, 51–58.

SALE85 Salesin, D., and R. Barzel, *Two-Bit Graphics*, Computer Graphics Project, Computer

Division, Lucasfilm, Ltd., San Rafael, CA, 1985; also in CG & A, 6(6), June 1986, 36–42.

SALM96 Salmon, G., *A Treatise on Conic Sections*, Longmans, Green, & Co., 10th edition, London 1896.

SALV87 Salvendy, G., ed., *Handbook of Human Factors*, Wiley, New York, 1987.

SAME84 Samet, H., "The Quadtree and Related Hierarchical Data Structures," *ACM Comp. Surv.*, 16(2), June 1984, 187–260.

SAME88a Samet, H., and R. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics, Part I: Fundamentals," *CG & A*, 8(3), May 1988, 48–68.

SAME88b Samet, H. and R. Webber, "Hierarchical Data Structures and Algorithms for Computer Graphics, Part II: Applications," *CG & A*, 8(4), July 1988, 59–75.

SAME89a Samet, H., "Neighbor Finding in Images Represented by Octrees," *CGVIP*, 46(3), June 1989, 367–386.

SAME89b Samet, H., "Implementing Ray Tracing with Octrees and Neighbor Finding," *Computers and Graphics*, 13(4), 1989, 445–460.

SAME90a Samet, H., *Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

SAME90b Samet, H., *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA, 1990.

SARR83 Sarraga, R.F., "Algebraic Methods for Intersections of Quadric Surfaces in GMSOLID," *CVGIP*, 22(2), May 1983, 222–238.

SCHA83 Schachter, B., *Computer Image Generation*, Wiley, New York, 1983.

SCHE86 Scheifler, R., and J. Gettys, "The X Window System," *ACM TOG*, 5(2), April 1986, 79–109.

SCHE88a Scheifler, R.W., J. Gettys, and R. Newman, *X Window System*, Digital Press, 1988.

SCHE88b Scherson, I.D., and E. Caspary, "Multiprocessing for Ray-Tracing: A Hierarchical Self-balancing Approach," *Visual Computer*, 4(4), October 1988, 188–196.

SCHM83 Schmid, C., *Statistical Graphics: Design Principles and Practice*, Wiley, New York, 1983.

SCHM86 Schmucker, K., "MacApp: An Application Framework," *Byte*, 11(8), August 1986, 189–193.

SCHU69 Schumacker, R., B. Brand, M. Gilliland, and W. Sharp, *Study for Applying Computer-Generated Images to Visual Simulation*, Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX, September 1969.

SCHU80 Schumacker, R., "A New Visual System Architecture," in *Proceedings of the Second Interservice/Industry Training Equipment Conference*, Salt Lake City, UT, 16-20 November 1980.

SCHU85 Schulert, A., G. Rogers, and J. Hamilton, "ADM—A Dialog Manager," in *CHI '85 Proceedings*, San Francisco, CA, Apr 14–18, 1985, 177–183.

SCHW82 Schweitzer, D., and E. Cobb, "Scanline Rendering of Parametric Surfaces," *SIGGRAPH 82*, 265–271.

SCHW87 Schwarz, M., W. Cowan, and J. Beatty, "An Experimental Comparison of RGB, YIQ, LAB, HSV, and Opponent Color Models," *ACM TOG*, 6(2), April 1987, 123–158.

SECH82 Sechrest, S., and D.P. Greenberg, "A Visible Polygon Reconstruction Algorithm," *ACM TOG*, 1(1), January 1982, 25–42.

SEDE84 Sederberg, T.W., and D.C. Anderson, "Ray Tracing of Steiner Patches," *SIGGRAPH 84*, 159–164.

SEDE86 Sederberg, T.W., and S.R. Parry, "Free-Form Deformation of Solid Geometric Models," *SIGGRAPH 86*, 151–160.

SEDG88 Sedgewick, R., *Algorithms*, second edition, Addison-Wesley, Reading, MA, 1988.

SELF79 Selfridge, P., and K. Sloan, *Raster Image File Format (RIFF): An Approach to Problems in Image Management*, TR61, Department of Computer Science, University of Rochester, Rochester, NY, 1979.

SELI89 Seligmann, D. and S. Feiner, "Specifying Composite Illustrations with Communicative Goals," *Proceedings of ACM UIST '89*, ACM, New York, 1989, 1–9.

SEQU89 Séquin, C.H. and E.K. Smyrl, "Parameterized Ray Tracing," *SIGGRAPH 89*, 307–314.

SHAN87 Shantz, M., and S. Lien, "Shading Bicubic Patches," *SIGGRAPH 87*, 189–196.

SHAN89 Shantz, M. and S. Chang, "Rendering Trimmed NURBS with Adaptive Forward Differencing," *SIGGRAPH 89*, 189–198.

SHAO88 Shao, M.Z., Q.S. Peng, and Y.D. Liang, "A New Radiosity Approach by Procedural Refinements for Realistic Image Synthesis," *SIGGRAPH 88*, 93–101.

SHAW91 Shaw, C.D., M. Green, and J. Schaeffer, "A VLSI Architecture for Image Composition," 1988 Eurographics Workshop on Graphics Hardware, Sophia-Antipolis, France, September, 1988. In Kuijk, A.A.M., ed., *Advances in Computer Graphics Hardware III*, Eurographics Seminars, Springer-Verlag, Berlin, 1991, 183–199.

SHER79 Sherr, S., *Electronic Displays*, Wiley, New York, 1979.

SHIN87 Shinya, M., T. Takahashi, and S. Naito, "Principles and Applications of Pencil Tracing," *SIGGRAPH 87*, 45–54.

SHIR86 Shires, G., "A New VLSI Graphics Coprocessor—The Intel 82786," *CG & A*, 6(10), October 1986, 49–55.

SHNE83 Shneiderman, B., "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer*, 16(8), August 1983, 57–69.

SHNE86 Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA, 1986.

SHOE85 Shoemake, K., "Animating Rotation with Quaternion Curves," *SIGGRAPH 85*, 245–254.

SHOU79 Shoup, R.G., "Color Table Animation," *SIGGRAPH 79*, 8–13.

SHOW89 Marketing Department, *Brochure*, Showscan Film Corp., Culver City, CA, 1989.

SIBE86 Sibert, J., W. Hurley, and T. Bleser, "An Object-Oriented User Interface Management System," *SIGGRAPH 86*, 259–268.

SIEG81 Siegel, R., and J. Howell, *Thermal Radiation Heat Transfer*, second edition, Hemisphere, Washington, DC, 1981.

SIG85 *Introduction to Image Processing, Course Notes 26 for SIGGRAPH 85*, San Francisco, California, July 1985.

SILL89 Sillion, F., and C. Puech, "A General Two-Pass Method Integrating Specular and Diffuse Reflection," *SIGGRAPH 89*, 335–344.

SIMP85 Simpson, C., M. McCauley, E. Roland, J. Ruth, and B. Williges, "System Design for Speech Recognition and Generation," *Human Factors*, 27(2), April 1985, 115–142.

SIMP87 Simpson, C., M. McCauley, E. Roland, J. Ruth, and B. Williges, "Speech Control and Displays," in Salvendy, G., ed., *Handbook of Human Factors*, Wiley, New York, 1987, 1490–1525.

SIOC89 Siochi, A., and H. R. Hartson, "Task-Oriented Representation of Asynchronous User Interfaces," in *Proceedings CHI '89*, ACM, New York, 183–188.

SKLA90 Sklar, D., "Implementation Issues for SPHIGS (Simple PHIGS)," Technical Report, Computer Science Department, Brown University, Providence, RI, August 1990.

SLOA79 Sloan, K.R., and S.L. Tanimoto, "Progressive Refinement of Raster Images," *IEEE Transactions on Computers*, C-28(11), November 1979, 871–874.

SMIT78 Smith, A.R., "Color Gamut Transform Pairs," *SIGGRAPH 78*, 12–19.

SMIT79 Smith, A.R., "Tint Fill," *SIGGRAPH 79*, 276–283.

SMIT82 Smith, D., R. Kimball, B. Verplank, and E. Harslem, "Designing the Star User Interface," *Byte*, 7(4), April 1982, 242–282.

SMIT84 Smith, A.R., "Plants, Fractals and Formal Languages," *SIGGRAPH 84*, 1–10.

SMIT87 Smith, A.R., "Planar 2-pass Texture Mapping and Warping," *SIGGRAPH 87*, 263–272.

SMIT88 Smith, D.N., "Building Interfaces Interactively," in *Proceedings ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 1988, 144–151.

SMIT89 Smith, A.R., "Geometry vs. Imaging," in *Proceedings of NCGA '89*, Philadelphia, PA, April 1989, 359–366.

SNOW83 Snowberry, K., S. Parkinson, and N. Sisson, "Computer Display Menus," *Ergonomics*, 26(7), July 1983, 699–712.

SNYD85 Snyder, H., "Image Quality: Measures and Visual Performance," in TANN85, 70–90.

SNYD87 Snyder, J.M. and A.H. Barr, "Ray Tracing Complex Models Containing Surface Tessellations," *SIGGRAPH 87*, 119–128.

SPAR78 Sparrow, E.M., and R.D. Cess, *Radiation Heat Transfer*, Hemisphere, Washington, DC, 1978.

SPRO82 Sproull, R.F., "Using Program Transformations to Derive Line-Drawing Algorithms," *ACM TOG*, 1(4), October 1982, 259–273.

SRIH81 Srihari, S., "Representation of Three-Dimensional Digital Images," *ACM Computing Surveys*, 13(4), December 1981, 399–424.

STAU78 Staudhammer, J., "On Display of Space Filling Atomic Models in Real Time," *SIGGRAPH 78*, 167–172.

STEI89 Steinhart, J., ed., *Introduction to Window Management, Course Notes 11 for SIGGRAPH 89*, Boston, MA, August 1989.

STER83 Stern, G., "Bbop—A System for 3D Keyframe Figure Animation," in *Introduction to Computer Animation, Course Notes 7 for SIGGRAPH 83*, New York, July 1983, 240–243.

STIN78 Stiny, G., and J. Gips, *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts*, University of California Press, Berkeley, 1978.

STON88 Stone, M., W. Cowan, and J. Beatty, "Color Gamut Mapping and the Printing of Digital Color Images," *ACM TOG*, 7(3), October 1988, 249–292.

STOV82 Stover, H., "True Three-Dimensional Display of Computer Data," in *Proceedings of SPIE*, 367, August 1982, 141–144.

STRA88 Strauss, P., *BAGS: The Brown Animation Generation System*, Ph.D. Thesis, Technical Report CS-88-22, Computer Science Department, Brown University, Providence, RI, May 1988.

SUKA88 Sukaviriya, P., "Dynamic Construction of Animated Help from Application Context," in *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, ACM, New York, 1988, 190–202.

SUKA90 Sukaviriya, P., and L. Moran, "User Interface for Asia," in Neilsen, J., ed., *Designing User Interfaces for International Use*, Elsevier, Amsterdam, 1990.

SUN86a Sun Microsystems, *Programmer's Reference Manual for the Sun Window System*, Sun Microsystems, Mountain View, CA, 1986.

SUN86b Sun Microsystems, *SunView™ Programmer's Guide*, Sun Microsystems, Mountain View, CA, 1986.

SUN87 Sun Microsystems, *NeWS™ Technical Overview*, Sun Microsystems, Mountain View, CA, 1987.

SUN89 Sun Microsystems, *OPEN LOOK Graphical User Interface*, Sun Microsystems, Mountain View, CA, 1989.

SUNF86 Sun Flex Corporation, *Touchpen*, Sun Flex, Novato, CA, 1986.

SUTH63 Sutherland, I.E., "Sketchpad: A Man–Machine Graphical Communication System," in *SJCC*, Spartan Books, Baltimore, MD, 1963.

SUTH65 Sutherland, I.E., "The Ultimate Display," in *Proceedings of the 1965 IFIP Congress*, 2, 1965, 506–508.

SUTH68 Sutherland, I.E., "A Head-Mounted Three Dimensional Display," in *FJCC 1968*, Thompson Books, Washington, DC, 757–764.

SUTH74a Sutherland, I.E., R.F. Sproull, and R.A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *ACM Computing Surveys*, 6(1), March 1974, 1–55. Also in BEAT82, 387–441.

SUTH74b Sutherland, I.E., and Hodgman, G.W., "Reentrant Polygon Clipping," *CACM*, 17(1), January 1974, 32–42.

SUTT78 Sutton, J., and R. Sprague, "A Survey of Business Applications," in *Proceedings American Institute for Decision Sciences 10th Annual Conference, Part II*, Atlanta, GA, 1978, 278.

SUYD86 Suydham, B., "Lexidata Does Instant Windows," *Computer Graphics World*, 9(2), February 1986, 57–58.

SWAN86 Swanson, R., and L. Thayer, "A Fast Shaded-Polygon Renderer," *SIGGRAPH 86*, 95–101.

SYMB85 Symbolics, Inc., *S-Dynamics*, Symbolics, Inc., Cambridge, MA, 1985.

TAMM82 Tamminen, M. and R. Sulonen, "The EXCELL Method for Efficient Geometric Access to Data," in *Proc. 19th ACM IEEE Design Automation Conf.*, Las Vegas, June 14–16, 1982, 345–351.

TAMM84 Tamminen, M., and H. Samet, "Efficient Octree Conversion by Connectivity Labeling," *SIGGRAPH 84*, 43–51.

TANA86 Tanaka, A.M., M. Kameyama, S. Kazama, and O. Watanabe, "A Rotation Method for Raster Images Using Skew Transformation," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, June 1986, 272–277.

TANI77 Tanimoto, S.L., "A Graph-Theoretic Real-Time Visible Surface Editing Technique," *SIGGRAPH 77*, 223–228.

TANN85 Tannas, L. Jr., ed., *Flat-Panel Displays and CRTs*, Van Nostrand Reinhold, New York, 1985.

TEIT64 Teitelman, W., "Real-Time Recognition of Hand-Drawn Characters," in *FJCC 1964*, *AFIPS Conf. Proc.*, 24, Spartan Books, Baltimore, MD, 559.

TEIT86 Teitelman, W., "Ten Years of Window Systems—A Retrospective View," in Hopgood, F.R.A., *et al.*, eds., *Methodology of Window Management*, Springer-Verlag, New York, 1986, 35–46.

TERZ88 Terzopoulos, D., and K. Fleischer, "Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture," *SIGGRAPH 88*, 269–278.

TESL81 Tesler, L., "The Smalltalk Environment," *Byte*, 6(8), August 1981, 90–147.

TEXA89 Texas Instruments, Inc., *TMS34020 and TMS34082 User's Guide*, Texas Instruments, Dallas, TX, March 1989.

THIB87 Thibault, W.C., and B.F. Naylor, "Set Operations on Polyhedra Using Binary Space Partitioning Trees," *SIGGRAPH 87*, 153–162.

THOM84 Thomas, S.W., *Modeling Volumes Bounded by B-Spline Surfaces*, Ph.D. Thesis, Technical Report UUCS-84-009, Department of Computer Science, University of Utah, Salt Lake City, UT, June 1984.

THOM86 Thomas, S.W., "Dispersive Refraction in Ray Tracing," *The Visual Computer*, 2(1), January 1986, 3–8.

THOR79 Thornton, R.W., "The Number Wheel: A Tablet-Based Valuator for Three-Dimensional Positioning," *SIGGRAPH 79*, 102–107.

TILB76 Tilbrook, D., *A Newspaper Page Layout System*, M.Sc. Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1976. Also see *ACM SIGGRAPH Video Tape Review*, 1, May 1980.

TILL83 Tiller, W., "Rational B-Splines for Curve and Surface Representation," *CG & A*, 3(6), September 1983, 61–69.

TILO80 Tilove, R.B., "Set Membership Classification: A Unified Approach to Geometric Intersection Problems," *IEEE Trans. on Computers*, C-29(10), October 1980, 847–883.

TORB87 Torborg, J., "A Parallel Processor Architecture for Graphics Arithmetic Operations," *SIGGRAPH 87*, 197–204.

TORR66 Torrance, K.E., E.M. Sparrow, and R.C. Birkebak, "Polarization, Directional Distribution, and Off-Specular Peak Phenomena in Light Reflected from Roughened Surfaces," *J. Opt. Soc. Am.*, 56(7), July 1966, 916–925.

TORR67 Torrance, K., and E.M. Sparrow, "Theory for Off-Specular Reflection from Roughened Surfaces," *J. Opt. Soc. Am.*, 57(9), September 1967, 1105–1114.

TOTH85 Toth, D.L., "On Ray Tracing Parametric Surfaces," *SIGGRAPH 85*, 171–179.

TOUL70 Touloukian, Y.S., and D.P. DeWitt, eds., *Thermophysical Properties of Matter: The TPRC Data Series, Vol. 7 (Thermal Radiative Properties: Metallic Elements and Alloys)*, Plenum, New York, 1970.

TOUL72a Touloukian, Y.S., and D.P. DeWitt, eds., *Thermophysical Properties of Matter: The TPRC Data Series,Vol. 8 (Thermal Radiative Properties: Nonmetallic Solids)*, Plenum, New York, 1972.

TOUL72b Touloukian, Y.S., D.P. DeWitt, and R.S. Hernicz, eds., *Thermophysical Properties of Matter: The TPRC Data Series, Vol. 9 (Thermal Radiative Properties: Coatings)*, Plenum, New York, 1972.

TRAU67 Traub, A.C., "Stereoscopic Display Using Rapid Varifocal Mirror Oscillations," *Applied Optics*, 6(6), June 1967, 1085–1087.

TRIC87 Tricoles, G., "Computer Generated Holograms: an Historical Review," *Applied Optics*, 26(20), October 1987, 4351–4360.

TROW75 Trowbridge, T.S., and K.P. Reitz, "Average Irregularity Representation of a Rough Surface for Ray Reflection," *J. Opt. Soc. Am.*, 65(5), May 1975, 531–536.

TUFT83 Tufte, E., *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, CT, 1983.

TURK82 Turkowski, K., "Anti-Aliasing Through the Use of Coordinate Transformations," *ACM TOG*, 1(3), July 1982, 215–234.

TURN84 Turner, J.A., *A Set-Operation Algorithm for Two and Three-Dimensional Geometric Objects*, Architecture and Planning Research Laboratory, College of Architecture, University of Michigan, Ann Arbor, MI, August 1984.

ULIC87 Ulichney, R., *Digital Halftoning*, MIT Press, Cambridge, MA, 1987.

UPSO88 Upson, C., and M. Keeler, "V-BUFFER: Visible Volume Rendering," *SIGGRAPH 88*, 59–64.

UPST89 Upstill, S., *The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics*, Addison-Wesley, Reading, MA, 1989.

VANA84 Van Aken, J. R., "An Efficient Ellipse-Drawing Algorithm," *CG&A*, 4(9), September 1984, 24–35.

VANA85 Van Aken, J.R., and M. Novak, "Curve-Drawing Algorithms for Raster Displays," *ACM TOG*, 4(2), April 1985, 147–169.

VANA89 Van Aken, J., personal communication, January 1989.

VANC72 Van Cott, H., and R. Kinkade, *Human Engineering Guide to Equipment Design*, 008-051-00050-0, U.S. Government Printing Office, Washington, DC, 1972.

VAND74 van Dam, A., G.M. Stabler, and R.J. Harrington, "Intelligent Satellites for Interactive Graphics," *Proceedings of the IEEE*, 62(4), April 1974, 483–492.

VERB84 Verbeck, C.P., and D.P. Greenberg, "A Comprehensive Light-Source Description for Computer Graphics," *CG & A*, 4(7), July 1984, 66–75.

VERS84 Versatron Corporation, *Footmouse*, Versatron Corporation, Healdsburg, CA, 1984.

VITT84 Vitter, J., "US&R: A New Framework for Redoing," *IEEE Software*, 1(4), October 1984, 39–52.

VOSS85a Voss, R., "Random Fractal Forgeries," in Earnshaw, R.A., ed., *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, Berlin, 1985; NATO ASI series F, volume 17, 805–835.

VOSS85b Vossler, D., "Sweep-to-CSG Conversion Using Pattern Recognition Techniques," *CG & A*, 5(8), August 1985, 61–68.

VOSS87 Voss, R., "Fractals in Nature: Characterization, Measurement, and Simulation," in *Course Notes 15 for SIGGRAPH 87*, Anaheim, CA, July 1987.

WALD64 Wald, G., "The Receptors for Human Color Vision," *Science*, 145, 1964, 1007–1017.

WALL87 Wallace, J.R., M.F. Cohen, and D.P. Greenberg, "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods," *SIGGRAPH 87*, 311–320.

WALL89 Wallace, J.R., K.A. Elmquist, and E.A. Haines, "A Ray Tracing Algorithm for Progressive Radiosity," *SIGGRAPH 89*, 315–324.

WAN88 Wan, S., K. Wong, and P. Prusinkiewicz, "An Algorithm for Multidimensional Data Clustering," *ACM Transactions on Mathematical Software*, 14(2), June 1988, 153–162.

WARD85 Ward, J., and B. Blesser, "Interactive Recognition of Handprinted Characters for Computer Input," *CG & A*, 5(9), September 1985, 24–37.

WARD88 Ward, G.J., F.M. Rubinstein, and R.D. Clear, "A Ray Tracing Solution for Diffuse Interreflection," *SIGGRAPH 88*, 85–92.

WARE87 Ware, C., and J. Mikaelian, "An Evaluation of an Eye Tracker as a Device for Computer Input," in *Proceedings of CHI + GI 1987*, ACM, New York, 183–188.

WARE88 Ware, C., and J. Beatty, "Using Color Dimensions to Display Data Dimensions," *Human Factors*, 20(2), April 1988, 127–42.

WARN69 Warnock, J., *A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures*, Technical Report TR 4-15, NTIS AD-753 671, Computer Science Department, University of Utah, Salt Lake City, UT, June 1969.

WARN83 Warn, D.R., "Lighting Controls for Synthetic Images," *SIGGRAPH 83*, 13–21.

WASS85 Wasserman, A., "Extending Transition Diagrams for the Specification of Human-Computer Interaction," *IEEE Transactions on Software Engineering*, SE-11(8), August

1985, 699–713.

WATE87 Waters, K., "A Muscle Model for Animating Three-Dimensional Facial Expressions," *SIGGRAPH 87*, 17–24.

WATK70 Watkins, G.S., *A Real Time Visible Surface Algorithm*, Ph.D. Thesis, Technical Report UTEC-CSc-70-101, NTIS AD-762 004, Computer Science Department, University of Utah, Salt Lake City, UT, June 1970.

WEGH84 Weghorst, H., G. Hooper, and D.P. Greenberg, "Improved Computational Methods for Ray Tracing," *ACM TOG*, 3(1), January 1984, 52–69.

WEIL77 Weiler, K. and P. Atherton, "Hidden Surface Removal Using Polygon Area Sorting," *SIGGRAPH 77*, 214–222.

WEIL80 Weiler, K., "Polygon Comparison Using a Graph Representation," *SIGGRAPH 80*, 10–18.

WEIL85 Weiler, K., "Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments," *CG & A*, 5(1), January 1985, 21–40.

WEIL86 Weil, J., "The Synthesis of Cloth Objects," *SIGGRAPH 86*, 49–54.

WEIL87 Weil, J., "Animating Cloth Objects," personal communication, 1987.

WEIL88 Weiler, K., "The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Modeling," in Wozny, M. J., H. McLaughlin, and J. Encarnação, eds., *Geometric Modeling for CAD Applications, IFIP WG5.2 Working Conference, Rensselaerville, NY, 12–14 May 1986*, North-Holland, 1988, 3–36.

WEIM80 Weiman, C.F.R., "Continuous Anti-Aliased Rotation and Zoom of Raster Images," *SIGGRAPH 80*, 286–293.

WEIN81 Weinberg, R., "Parallel Processing Image Synthesis and Anti-Aliasing," *SIGGRAPH 81*, 55–61.

WEIN87 Weingarten, N., personal communication, 1987.

WEIN88 Weinand, A., E. Gamma, and R. Marty, "ET++—An Object Oriented Application Framework in C++," *OOPSLA 1988 Proceedings*, ACM-SIGPLAN Notices, 23(11), November 1988, 46–57.

WEIS66 Weiss, R.A., "BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces," *JACM*, 13(2), April 1966, 194–204. Also in FREE80, 203–213.

WELL76 Weller, D., and R. Williams, "Graphic and Relational Data Base Support for Problem Solving," *SIGGRAPH 76*, 183–189.

WELL89 Wellner, P., "Statemaster: A UIMS Based on Statecharts for Prototyping and Target Implementation," in *Proceedings of CHI '89*, ACM, New York, 177–182.

WERT39 Wertheimer, M., "Laws of Organization in Perceptual Forms," in Ellis, W.D., ed., *A Source Book of Gestalt Psychology*, Harcourt Brace, New York, 1939.

WESL81 Wesley, M.A., and G. Markowsky, "Fleshing Out Projections," *IBM Journal of Research and Development*, 25(6), November 1981, 934–954.

WEST89 Westover, L., "Interactive Volume Rendering," in *Proceedings of Volume Visualization Workshop*, Department of Computer Science, University of North Carolina at Chapel Hill, May 18-19, 1989, 9–16.

WHEL82 Whelan, D., "A Rectangular Area Filling Display System Architecture," *SIGGRAPH 82*, 147–153.

WHIT80 Whitted, T., "An Improved Illumination Model for Shaded Display," *CACM*, 23(6), June 1980, 343–349.

WHIT82 Whitted, T., and S. Weimer, "A Software Testbed for the Development of 3D Raster

Graphics Systems," *ACM TOG*, 1(1), January 1982, 43–58.

WHIT83 Whitted, T., "Anti-Aliased Line Drawing Using Brush Extrusion," *SIGGRAPH 83*, 151–156.

WHIT84 Whitton, M., "Memory Design for Raster Graphics Displays," *CG & A*, 4(3), March 1984, 48–65.

WHIT85 Whitted, T., "The Hacker's Guide to Making Pretty Pictures," in *Image Rendering Tricks, Course Notes 12 for SIGGRAPH 85*, New York, July 1985.

WILH87 Wilhelms, J., "Using Dynamic Analysis for Realistic Animation of Articulated Bodies," *CG & A*, 7(6), June 1987, 12–27.

WILL72 Williamson, H., "Algorithm 420 Hidden-Line Plotting Program," *CACM*, 15(2), February 1972, 100–103.

WILL78 Williams, L., "Casting Curved Shadows on Curved Surfaces," *SIGGRAPH 78*, 270–274.

WILL83 Williams, L., "Pyramidal Parametrics," *SIGGRAPH 83*, 1–11.

WITK87 Witkin, A., K. Fleischer, and A. Barr, "Energy Constraints on Parameterized Models," *SIGGRAPH 87*, 225–232.

WITK88 Witkin, A., and M. Kass, "Spacetime Constraints," *SIGGRAPH 88*, 159–168.

WOLB88 Wolberg, G., *Geometric Transformation Techniques for Digital Images: A Survey*, Technical Report CUCS-390-88, Department of Computer Science, Columbia University, New York, December 1988. To appear as Wolberg, G., *Digital Image Warping*, IEEE Computer Society, Washington, DC, 1990.

WOLB89 Wolberg, G., and T.E. Boult, *Separable Image Warping with Spatial Lookup Tables*, SIGGRAPH 89, 369–378.

WOLB90 Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

WOLF87 Wolf, C., and P. Morel-Samuels, "The Use of Hand-Drawn Gestures for Text Editing," *International Journal of Man-Machine Studies*, 27(1), July 1987, 91-102.

WOLF90 Wolff, L., and D. Kurlander, "Ray Tracing with Polarization Parameters," *CG&A*, 10(6), November 1990, 44–55.

WOO85 Woo, T., "A Combinatorial Analysis of Boundary Data Structure Schemata," *CG & A*, 5(3), March 1985, 19–27.

WOOD70 Woods, W., "Transition Network Grammars for Natural Language Analysis," *CACM*, 13 (10), October 1970, 591–606.

WOOD76 Woodsford, P. A., "The HRD-1 Laser Display System," *SIGGRAPH 76*, 68–73.

WOON71 Woon, P.Y., and H. Freeman, "A Procedure for Generating Visible-Line Projections of Solids Bounded by Quadric Surfaces," in *IFIP 1971*, North-Holland Pub. Co., Amsterdam, 1971, pp. 1120–1125. Also in FREE80, 230–235.

WRIG73 Wright, T.J., "A Two Space Solution to the Hidden Line Problem for Plotting Functions of Two Variables," *IEEE Trans. on Computers*, 22(1), January 1973, 28–33. Also in FREE80, 284–289.

WU89 Wu, X., and J.G. Rokne, "Double-Step Generation of Ellipses," *CG & A*, 9(3), May 1989, 56–69.

WYLI67 Wylie, C., G.W. Romney, D.C. Evans, and A.C. Erdahl, "Halftone Perspective Drawings by Computer," *FJCC 67*, Thompson Books, Washington, DC, 1967, 49–58.

WYSZ82 Wyszecki, G., and W. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*, second edition, Wiley, New York, 1982.

WYVI86 Wyvill, G., C. McPheeters, and B. Wyvill, "Data Structures for Soft Objects," *The*

Visual Computer, 2(4), April 1986, 227–234.

WYVI88 Wyvill, B., "The Great Train Rubbery," *ACM SIGGRAPH 88 Electronic Theater and Video Review*, 26, 1988.

YEAG86 Yeager, L., and C. Upson, "Combining Physical and Visual Simulation—Creation of the Planet Jupiter for the Film '2010'," *SIGGRAPH 86*, 85–93.

ZDON90 Zdonik, S.B., and D. Maier, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann, San Mateo, CA, 1990.

ZELT82 Zeltzer, D., "Motor Control Techniques for Figure Animation," *CG & A*, 2(11), November 1982, 53–59.

ZIMM87 Zimmerman, T., J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill, "A Hand Gesture Interface Device," in *Proceedings of the CHI + GI* 1987 Conference, ACM, New York, 189–192.

索 引

索引中的页码为英文原书页码，与书中页边标注的页码一致。

A

Abort command（中止命令），409~410
A-buffer algorithm（A缓存算法），671, 693~695
Accelerator keys（快捷键），367~368, 411~412, 436
ACE（A Color Expert）（彩色专家系统），601
Achromatic light（消色差光），563~573
Action（作用，动作），1066, 1068~1070
Action, routines（动作例程），457
Active values（活动值），466
Active zone（活动区域），560
Active-edge table（活动边表），97, 681, 886
Active-surface table（活动面表），684
Actors（演员），1073
Adaptive subdivision（自适应细分），见Curved surfaces, Spatial partitioning
Additive color primaries（附加色彩基（值）），585
Address space（地址空间），166, 177
  single（单一），177~179
Addressability（寻址能力），146, 170
Adjoint（伴随的），217, 1108
Aerial perspective（空中透视），610
Affine combination（仿射组合），1086
Affine space（仿射空间），1083~1108, 1085
Affine transformation（仿射变换），见Transformation, affine
Aliasing（走样），612, 627~628, 671。另见Antialiasing
  artifact, 14
  in scan conversion（扫描转换中的），1008
  in scan converting conics（扫描转换），957
  sliver polygons（狭长多边形），96
  temporal（时域），1058, 1078
Alignment（排列、校直、准线）
  for patterns（图案），101
α (angle between $\vec{R}$ and $\vec{V}$)（$\vec{R}$和$\vec{V}$间的夹角），729, 813
Alpha_1（造型系统），540~541, 547
α-channel（α通道），见Compositing, α-channel
Ambient light（环境光），564, 612, 722~723
Ambient reflection coefficient (k_a)（环境反射系数），723
American National Standards Institute（美国国家标准协会(ANSI)），见ANSI
Ampex digital optics (ADO), 829
Anchor of pattern（图案固定），101
Andrew（安德鲁MORR86(有关的视窗管理技术)），441, 452
Animation（动画），5, 1057~1081
  basic rules（基本规则），1077~1078
  cartoon -character（卡通人物（角色）），1077
  control（控制），1070~1078
  conventional（常规），1058~1059
  on fields（区域），1079
  flip-book（翻动书页），1081
  graphical languages（图形语言），1067
  key-frame（关键帧），1058, 1059
  linear-list notations（线性表表示法），1065
  look-up table（查找表），180~181, 1064~1065
  physically based（基于物理的），1076
  staging of（分解），1078
  on twos（每两帧动画）1080
Animation control（动画控制）
  constraint-based（基于约束的），1071
  explicit（显式的），1070
  key-frame（关键帧），1070
  procedural（过程），1070
  tracking live action（真实运动跟踪），1073
ANSI (American National Standards Organization)（美国国家标准化组织），16, 285
Antialiased brushes（反走样笔刷），974
Antialiasing（反走样），14, 132~142, 598, 612, 617~646, 817~818, 909, 965~976。另见Filter, Area sampling
  circles（圆的），969~971
  conics（圆锥曲线的），971~974
  general curves（一般曲线），974
  Gupta-Sproull line algorithm（Gupta-Sproull 线算法），137~142
  Gupta-Sproull techniques（Gupta-Sproull技术），969, 975~976
  lines（直线的），967~969, 975
  multiple lines（多条直线的），968
  polygons（多边形的），975
  rectangles（矩形的），975
  temporal（时域的），819, 1079
  text（文本的），977
  2-bit（2位），968
Application-dependent data（依赖于应用的数据），844

Ardent Titan（SGI图形工作站使用的一种技术），890~891
Area sampling（区域采样）
  unweighted（未加权区域采样），132~135
  weighted（加权区域采样），135~142
Area subdivision algorithm, Warnock（Warnock的区域细分算法），550
Area subdivision algorithms（区域细分算法），686~695
Artificial reality（人造现实），21, 357另见Virtual world
ASAS（一种动画系统使用的语言），1066, 1073
Atmosphere（大气、空气、环境），1044
Atmospheric attenuation（大气衰减），727~728
AT&T Pixel Machine（AT&T像素处理器），889, 911~912
Attribute bundle, PHIGS（属性包），332
Attribute（属性），18, 19
  cosmetic（装饰），945
  geometric（几何），945
  inheritance（继承性），318~321
  nongeometric（非几何），298
  output primitives（输出图元），30~40
  SPHIGS, 298, 318~321
  SRGP, 38
Audio（音频）。见Voice input-output
Autocompletion, command, 362
Axial ray（轴向射线），787

B

B(radiosity)（辐射度），793
B-rep（边界表示），见Boundary representation
Back distance (B)（后距离），241
Back-face culling（背面消除），663~664, 807
Background, color attribute（背景，颜色属性），35
Backing store（后备存储，备用存储），996, 997
Backus-Naur form（Backus-Naur范式），461
Bandwidth（带宽），158, 180
Bandwidth limiting（带宽限制），629
Baseline , text（基线，文字），39
Basis（基），
  standard（标准），1093
  of a vector space（向量空间的），1092
Basis matrix（基矩阵），483, 489, 493, 505, 510
Batch, screen updates（屏幕更新批量），328
BBN Butterfly（一种并行处理的方案），911
Beam（光束），784
Beam current（电子束电流），163
Beam tracing（光束跟踪），787
Bernstein polynomials（Bernstein多项式），489~490, 494
β(angle between $\bar{N}$ and $\bar{H}$)（$\bar{N}$和$\bar{H}$之间的角），731, 813
β-splines（β样条），见Splines
Bézier curves（Bézier曲线），见Splines, Bézier curves
Bias parameter, β-spline（偏移参数，β样条），505~506
Bicubic surface（双三次曲面），473
Bicubic surfaces, drawing（画双三次曲面），523~528
Bidirectional reflectivity(ρ)（双向反射率），763
  diffuse（双向漫反射率），763
  specular（双向镜面反射率），763
Bilevel CRT（二值阴极射线管），见Bilevel display
Bilevel display（二值显示器），12, 564, 568, 570
Bilinear（双线性），1095
Binary space-partitioning (BSP)tree（二元空间划分树），
  regularized Boolean set operations（正则的布尔集合运算），546, 556~557
  shadow algorithm（阴影算法），751
  for solid modeling（实体造型），555~557, 559
  visible-surface determination（可见面的判定），675~680
Binding（绑定），见Hardware binding
Binocular disparity（双眼视差），616
Bintree（二叉树），552, 784
BitBlt, 69, 986~992。另见CopyPixel, PixBlt
  implementation（实现），132
  managing windows（管理窗口），996~998
  pseudocode（伪代码），988
Bitmap（位图），1, 13。另见Pixmap
  characters（字符），127
  offscreen（屏外）。见Canvas, offscreen
  pattern（图案），34~36
  scaling of（缩放），851
Blanking（留空），12
Blending functions, curves（调配函数），483, 485, 493~494, 497
Blobby objects（滴状物体），1047
Bloom, phosphor（光晕，磷光物质），158
Böhm algorithm, for cubic curves（三次曲线的Böhm算法），510
Boldface（粗体，黑体），见Character
Boolean operations on polygons（多边形的布尔运算），937
Boolean operations on solids（实体的布尔运算），见Regularized boolean set operations
Bottlenecking（瓶颈），831, 832~833
Boundary fill（边界填充），见Filling algorithms, boundary fill
Boundary representation（边界表示），542~548, 559, 560
  adjacency relationships（相邻关系），546
  non-polyhedral（非多面体），547~548
  regularized Boolean set operations（正则布尔集合运算），546~547

Bounding box（边界盒），660。另见Extent
Box filter（盒式过滤器），见Filter, box
Bresenham, circle scan conversion（Bresenham，圆扫描转换），见Scan conversion, midpoint circle
Bresenham, line scan conversion（Bresenham，线扫描转换），见Scan conversion, midpoint line
Brewster angle（Brewster角），771
Brightness, of light（光的辉度），563, 584
BRIM, 849
Brush（画刷），另见Antialiased brush
  orientation（方向），104
  shape（形状），104
BSP tree（BSP树），见Binary-partitioning tree
B-spline（B样条），见Curved surfaces, Curves
B-spline curves（B样条曲线），见Splines, B-spline curves
B-spline surfaces（B样条曲面），见Splines, B-spline surfaces
Buffer（缓存，缓冲区），见Refresh buffer
Bump mapping（凹凸映射），1043
Button mask（按钮标志），47~48
By-example specification（运用实例说明），464

C

Cabinet projection（斜二侧投影），见 Projection
CAD。参见Computer-aided design
Callback procedures（回调过程），450~452
Calligraphic display（书法显示器）。另见Random-scan display
Camera viewing（照相机观察），249
Camera interpolation（照相机插值），见Interpolation, camera
Cancel command（"取消"命令），409~410
Canvas（画布），53~54
  offscreen（屏外），69
Cartography（制图学），6
CAT scan（CAT扫描）。见Computed tomography
Catenary（悬链线），1041
Cathode-ray tube (CRT)（阴极射线管），155~160, 564, 570,641,856。另见Flat tension mask,Flicker, Focus
  shadow-mask（阴罩），158
Catmull subpixel area-subdivision algorithm（Catmull子像素区域细分算法），693
Catmull-Rom splines（Catmull-Rom 样条）。参见Splines, Catmull-Rom,
Cavalier projection（斜等侧投影），参见Projection
Cel, 1059
Cell decomposition（单元分解），548~549, 558
Center of projection (COP)（投影中心），230
Center of window (COW)（窗口中心），238
Central Structure Storage（中央结构存储器），293~295。另见Structure network
Channel（信道，通道），844
Character（字符）
  alignment（排列），129
  boldface（黑体），131
  descender（下标），39, 129
  font（字体），129
  italic（斜体），131
  recognition（识别），373
  roman（罗马），129
  typeface（字型），127, 129
  width（宽度），129
Charge-coupled device (CCD)（电耦合设备），195~196
Choice logical device（选择逻辑设备），42, 188, 195, 352, 436, 452
Choice set（选择集），361
Chord, locator button state（和弦，定位器按钮状态），44
Chroma, color（色度），574
Chromaticity, color（色度），580~583, 589
Chromaticity coordinates（色度坐标），580, 586, 589, 600
Chromaticity diagram（色度图），见CIE chromaticity diagram
CIE（Commission Internationale de l'Éclairage）（国际照明委员会），579
CIE chromaticity diagram（CIE色度图）579~584
CIE color model（CIE颜色模型），585
CIE LUV uniform color space（CIE LUV均匀颜色空间），584, 594
CIE primaries（CIE基色），579~580
Circle, implicit equation of（隐式方程的圆，圆周），1097
Circles（圆），见Scan conversion, general circles
Classical adjoint（经典伴随）见Adjoint, classical
Click and drag interaction（点击与拖动交互操作），386
Client（客户机），440
Clip rectangle（裁剪矩形），55
Clipping（裁剪），77, 924
  analytical（分析），110
  characters（字符），127
  Cohen-Sutherland line algorithm（Cohen-Sutherland直线算法），113
  Cyrus-Beck-Liang-Barsky algorithm（Cyrus-Beck-Liang - Barsky算法），925
  depth（深度），870
  endpoints（端点），112
  to general windows（一般窗口），995
  in homogeneous coordinates（齐次坐标中的），272, 870
  Liang-Barsky line algorithm（梁友栋-Barsky直线算法），121~124

Liang-Barsky polygon algorithm（梁友栋-Barsky多边形算法），929, 930~937, 1006
lines（线），111~124, 925
Nichol-Lee-Nichol line algorithm（Nichol-Lee-Nichol线算法），115, 117, 925~928, 1006
polygon（折线），124~127,924,929
in PostScript（在PostScript中），1002
to rectangular windows（矩形窗口），924
Sutherland-Hodgman polygon-clipping algorithm（Sutherland-Hodgman多边形裁剪算法），124~129, 274, 929
text string（文本串），127
3D（三维），271~274, 659, 869~870, 878~879
3D Cohen-Sutherland line algorithm（三维Cohen-Sutherland线算法），271
3D Cyrus-Beck（三维Cyrus-Beck算法），271
3D Liang-Barsky（三维梁友栋-Barsky算法），271
trivial acceptance, trivial rejection（简单接受，简单拒绝），868, 878
2D primitives in a raster world（光栅世界的二维图元），110~127
2D raster graphics（二维光栅图形学），55
vertical polygon edges（垂直多边形边），935
Weiler polygon algorithm（Weiler多边形算法），929, 937~945, 1006
Clipping plane（裁剪平面），
back（yon）（后），240
front（hither）（前），240
Closure, visual（封闭性，视觉），418~419
Cloth（布料），1041
Clouds（云），1044~1045
fractal-based（基于分形的云），1044
Cluster（串，簇），675
Cluster priority（组优先级），676
Clustered-dot ordered dither（聚点有序抖动），568, 570, 572
CMY color model（CMY颜色模型），584, 587~589, 600
CMYK color model（CMYK颜色模型），588
Coding（编码）
information（信息），404
redundant（冗余），422, 424
visual（可视），387. 422~425
Coding categories（编码分类），422
Cohen-Sutherland line-clipping algorithm（Cohen-Sutherland线剪裁算法），见Clipping, Cohen-Sutherland line algorithm
Coherence（相关性），91, 657
area coherence（区域相关性），657, 686
depth coherence（深度相关性），657, 684
edge coherence（边相关性），91, 657, 680
face coherence（面相关性），657
frame coherence（帧相关性），657, 664, 715
implied edge coherence（隐含边相关性），657
invisibility coherence（不可见相关性），685
object coherence（对象相关性），657, 751, 784
scan-line coherence（扫描线相关性），91, 657, 680, 715
span coherence（跨度相关性，跨段相关性），657, 681
spatial coherence（空间相关性），881
Color（颜色），见Chromaticity, Dominant wavelength, Hue, Luminance, Metamer
Color blind（色盲），见Color deficient
Color coding（颜色编码），422~424, 602
Color deficient（色盲），422, 424, 601~602
Color gamuts（颜色域），583~584
Color harmony（色彩融合），601
Color interpolation（颜色插值），598~599
Color map（色彩图），见Video look-up table
Color matching functions（颜色匹配函数），578~580
Color models（颜色模型），584~595。另见CIE, CMY, HLS, HSB, HSV, HVC, RGB, YIQ
Color table（颜色表），32。另见Video look-up table
Color usage rules（颜色使用规则），602
Colorimeter（色度计），582, 586
Colorimetry（色度学；比色计），575
Column-preserving map（列保留映射），828
Comb function（梳状函数），636~637
Command language（指令语言），348, 402, 403
Command mode（命令方式），416~417
Commission Internationale de l'Éclairage（国际照明委员会），见CIE
Commutativity, matrix operations（可交换性，矩阵运算），209~210
Complementary colors（补色），582~583, 590
Composite architecture（组合结构），见Image-composition architecture
Composite interaction task（组合交互作业），见Interaction tasks
Composite modeling transformation matrix (CMTM)（合成建模变换矩阵），316
Composite video（复合视频），180
Compositing（复合，合成）
α-channel（α通道），835~840
hardware（硬件），840~842
image（图像），815, 835~843, 850
Composition（合成），见Transformation composition
matrix（矩阵），205
Compositor（合成器），901, 906
Computed tomography（CT），816, 1035, 1039
Computer-aided design (CAD)（计算机辅助设计），5, 7, 471, 514, 516
Conceptual design, of user interface（用户界面的概念性设计），394~395, 404, 429, 430

Conductor（导体，导线），767，770~771
Cone filter（锥形滤波器），136
Cone receptors in eye（锥形视觉接收器），576
Cone tracing（锥跟踪），786~787，另见Ray tracing
Cones（锥体），733
Conics（圆锥曲线），见Scan conversion，general conics
Connection Machine（连接器），见Thinking Machines Connection Machine
Connectivity of regions（区域连通性），979
Consistency（一致性，连贯性）
  user interface（用户界面），404~405
  visual（视觉），425~426
Constraint（约束），360，378，454，1040
  dynamic（动态），1040，1076
  energy（能量），1040
  in line drawing（画线中的），384
Constraint-based modeling（基于约束的建模），1040
Constructive planar geometry（构造的平面几何法），938
Constructive solid geometry（CSG）（构造实体几何），557~558，559，560，672，901
Containment tree（包含树），943，1007
Context，PostScript（上下文，PostScript），1000
Context switching（上下文切换），907~908
Context-sensitivity，user interface（上下文相关用户界面），409，412，413，417，457，459
Contiguous partitioning（邻近划分），887~888。另见Parallel rasterization architectures，image-parallel
Continuity（连续性），见Curves
  curved surface（曲面），480~482
Contouring，intensity（轮廓线，亮度），569
Control grid of CRT（控制网格阴极射线管的），155，565
Control points（控制点），见Curves，Curved surfaces
Control to display ratio(C/D ratio)（控制-显示比率），351，352，375
Convergence of electron beam（电子束的聚焦性），156
Conversion between color models（颜色模式间转换），584~596
Conversion between curve representations（曲线表示间的转换），510~511
Convex hull（凸包），488，490~491，492，494，496，509
Convolution（卷积），629~633。另见Filter
  graphical（图形的），632~633
  precomputed tables（预计算表），695
Cook-Torrance，见 Illumination，Cook-Torrance
Coons'patch（孔斯曲面片），519
Coordinate system（坐标系），1092，1094
  application（应用），60，280
  camera（照相机），280
  device（设备），210，281
  eye（眼睛），280
  left-handed（左手），280
  local（局部），280，378
  logical device（逻辑设备），280
  modeling（模型），280
  normalized device（规格化设备），280~281
  normalized projection（规格化投影），241，278，280
  object（物体），280，359
  problem（问题），280
  raster（光栅），281
  right-handed（右手），214，280
  screen（屏幕），280~281，359，379
  in SRGP（在SRGP中），26
  (u，v，n)，238，280
  (u，v，VPN)，280
  view reference（视图参考），238，280
  world（世界），210，280，359
Coordinates（坐标）
  absolute（绝对），185
  in an affine space（仿射空间），1094
  relative（相对），185
  in a vector space（向量空间中），1093
Coordinate-system representation（坐标系表示），72
CopyPixel，56，986~992
Coral（珊瑚），1048
Core Graphics System（核心图形系统），15
Correct command（命令），409~410
Correlation.（相关），见Pick correlation
COUSIN，465
Cramer's rule（克莱姆法则），1106
Cross-hatching（交叉线），见Vector cross-hatching
Cross-product（向量的叉积），1104
CRT（阴极射线管），见Cathode ray tube
CSG，见Constructive solid geometry
CSS，见Central Structure Storage
CT scan（CT扫描），见Computed tomography
Cube architecture（立方体体系结构），914
Cuberille，549
Cubic curve（三次曲线），472。另见Splines
  drawing（绘制），511~514
Culling，structure network（剔除网络结构），340
Current command（当前指令），415
Current path，PostScript（当前路径，PostScript），1003
Current point（当前点），415
Current position (CP)（当前位置），171~174
Currently selected object (CSO)（当前选定的对象），416，417，459~460，463
Cursor，3D（游标），672
Curved surfaces（曲面），516~529。另见Surface patch
  adaptive subdivision（自适应分割），527
  display methods（显示方法），913

forward differences（前后差分），913
tesselation（网格化），807
Curves（曲线），另见Splines
parametric polynomial（参数多项式），472
parametric cubic（三次参数），478~516
Cyrus-Beck line clipping algorithm（Cyrus-Beck线裁剪算法），117~124

D

D(microfacet distribution function)（微面元分布函数），764
Beckmann, 764~765
Trowbridge-Reitz, 764
d_L(distance from point source to surface)（点源到曲面的距离），725
D/A conversion（数模转换），见Digital-to-analog conversion
Damage repair（破损修复），37。另见Regeneration strategy
Data flow（数据流），465
Data model（数据模型），466
Data tablet（数据输入板），见Tablet
Database（数据库），见Display model
Database amplification（数据库扩展），1011
DataGlove（数据手套），355~357, 1073
DC (direct current)（直流），625, 630, 637
DDA, 见Digital differential analyzer
Decision variable（判定变量）
for circle scan conversion（圆扫描转换），84
for ellipse scan conversion（椭圆扫描转换），88
for line scan conversion（线扫描转换），76
Deflection coils（偏转线圈），155
Deformable solids（可变形实体），1042
Degree elevation, parametric curves（升阶，参数曲线），507
Degrees, rectangular（角度，矩形），30
Demultiplex events（信号分离事件），447
Density emitter（密度发射器），1037
Dependence, linear（相关，线性），1091
Depth buffer（深度缓存），见z-Buffer
Depth clipping（深度裁剪），611
Depth cueing（深度提示），610, 727~728
Depth of field（域的深度），615, 774~775, 789
Depth-sort algorithm（深度排序算法），673, 715~716
Descender, character, 见Character, descender
Design objective, user interface（用户界面的设计目标），391~392, 405, 421, 458
Design rules, visual（设计规则），426~428
Desktop metaphor（桌面隐喻），347
Detectability, filter（可检测性，过滤器），333
Determinant（行列式），见Matrix, determinant of
Device-independence（设备无关性）
graphics（图形），15
interface（接口），67
DIAL（拨号），1067
Dialogue box（对话框），381~382, 411, 414, 425~426, 445, 452
Dicing（切割），811
Dielectric（非传光，非光导），766, 770~771
Diffuse reflection（漫反射），723~727
coefficient（k_d）（系数），724
Digital differential analyzer（数字微分分析器），74
Digital Equipment Corporation (DEC)（数字设备公司），430
Digital holography（数字全息术），918~919
Digital typography（数字印刷），见Typography, digital
Digital-to-analog conversion（数模转换），860
Digitize（数字化），350, 355。另见Scanning
Dimension, fractal（维数，分形），1020
Direct manipulation（直接操纵），2
Direct manipulation user interfaces（直接操纵用户界面），348, 397~398, 403, 412, 465
Direction of projection (DOP)（投影方向），231, 238
Direction of reflection ($\bar{R}$)（反射方向），729
Direct-view storage tube(DVST)（直视存储器），10, 161
Discrete Fourier transform（离散傅里叶变换），625
Discretely shaped β-splines（离散造影β样条），507
Dispersed-dot ordered dither（离散点有序抖动），570, 572
Dispersion（散射，离差），757
Display controller（显示控制器），11, 69
Display coprocessor（显示协处理器），见Graphics, display processor
Display devices, raster（显示设备，光栅），25
Display list（显示列表），9
Display list storage（显示列表存储），176
Display model（显示模型），862
distributed（分布式），881
Display procedure（显示过程），308
Display processing unit（显示处理单元（DPU）），861~866。另见Random-scan display processor, Display controller
Display program（显示程序），见Display list
Display traversal（显示遍历），294, 299, 308~331, 867~868, 877, 880~881
attribute inheritance（属性继承），318~321
immediate mode（瞬时模式），867
implementation（实现），334~338
modeling transformation（模型变换），315~317, 336
optimization（优化），336~337, 340~341
retained mode（保留模式），867
viewing transformation（观察变换），299~302
Distance formula, from point to line（距离公式，点到线的），1100

Distributed frame buffer（分布式帧缓存），见Parallel rasterization architectures, image-parallel
Distributed ray tracing（分布式光线跟踪），788~792
  jittering（抖动），788
  Poisson distribution（泊松分布），788
Dither matrix（抖动矩阵），569~572
Dithering（抖动），569, 599。另见Ordered dither, Random dither
Do what I mean (DWIM), 362
Do-it command（Do-it命令），417
Domain（域），
  frequency（频），623
  spatial（空），618, 623
  temporal（时），618
Dominant wavelength, color（主波长，颜色），575~576, 580, 582
Doré, 810
Dot product（点积），1094
  in region classification（区域分类中的），118
Dot size（点尺寸），146, 158
Double buffering（双缓冲），13, 177, 337~338, 858~859, 886
Double-hexcone（双六棱锥），592, 594
HLS color model（HLS颜色模型）
Dragging（拖动），386
DRAM (Dynamic random-access memory), 动态随机存储器），857~858, 871, 890
Drawing octants（画八分圆弧），953
DVST（直视存储管），见Direct-view storage tube
Dynamic constraints（动态约束），见Constraint, dynamic
Dynamic Digital Displays Voxel Processor（动态数字显示体素处理器），914
Dynamic random-access memory（动态随机存储器），见DRAM
Dynamic range, intensity（亮度变化范围），564, 566~567
Dynamics（动力学），615, 1071, 1074~1076
  geometric modeling（几何造型），311~314
  inverse（逆向），1076
  motion（运动），4, 1057
  update（更新），4, 1057

E

E_i (incident irradiance)（入射光的辐照度），762
Echo（回应）。见Feedback
Edge coherence（边相关性），96
Edge detection（边缘检测），820
Edge enhancement（边缘增强），820
Edge table（边表），681
Editing, of structure network（结构网络的编辑），324~327
Eigenvalue（特征值），1108
Eigenvector（特征向量），1108
8 by 8 display（8×8显示），889
8-connected region（8连通区域），979
Elasticity（弹性），1042
Electroluminescent (EL) display（电致发光显示），164
Electromagnetic energy（电磁能），575
Electron gun（电子枪），155, 159
Electrophoretic display（电泳显示），164
Element, structure network（元素，结构网络），293, 295, 324
Ellipse（椭圆）
  arc specification（弧定义），30
  specification（描述），951, 1007。另见Scan conversion, general ellipses
Elliptical weighted-average filter（椭圆加权平均滤波器），827~828
Energy distribution（能量分布），见Spectral energy distribution
Environment mapping（环境映射），见Reflection mapping
Ergonomics（人类工程学），见Human factors
Error diffusion, Floyd-Steinberg（误差扩散），572~573
Error measure（误差测量），73
Error rates（误差率），391, 403
  selection task（选择任务），360, 370
Error recovery（错误恢复），409~411
Errors, user（误差，用户），394, 409~411
$\eta_{i\lambda}$, $\eta_{t\lambda}$(indices of refraction)（折射率），757
Euler angles（欧拉角），1063
Euler operators（欧拉算子），544, 561
Euler's formula（欧拉公式），543
Evans & Sutherland Computer Corp.（Evans & Sutherland计算机公司），886, 919
Even-odd rule（奇偶规则），965
Event（事件），20
  input（输入），42
  SRGP input mode（SRGP输入模式），45~48
  window system（窗口系统），447~451
Event language（事件语言），462~463
Event mode, input devices（事件模式，输入设备），437~439
Event queue（事件队列），42, 437~440
Event routing（事件路由）
  dispatcher（发送方），450
  listener（接收方），449
  notifier（通告者），450
  real-estate based（基于不动产的），448
  user（用户），448~450
Event-driven interaction（事件驱动交互），42~48

Excitation purity, color（色值度，颜色），575~576, 582, 583, 591
Expert system（专家系统），413, 425, 466~467, 601
Explicit functions（显函数），478
Extensibility of user interface（用户界面的可扩展性），393, 403, 413
Extent（范围），660~663, 887
  minmax（最小最大），661
  text（文本），39
  3D object（三维物体），336
Extinction, coefficient of(k_λ)（消光系数），767
Extrusion textures（突出（挤压）纹理），1018
Eye（视线），564
Eye tracker（视线跟踪仪），351
Eye-hand coordination（眼手协调），350, 360

F

F_λ(Fresnel term)（菲涅尔项），764, 766~767
f_{att} light-source attenuation fador（光源衰减因子），725
Faces, modeling（人脸造型），1049
Factoring of user interface commands（用户界面命令参数化），415, 418
Fade operator（淡化操作符），839
Fast Fourier transform（快速傅里叶变换），626
Feedback（反馈），360, 362, 375, 393, 404~408, 414, 437
  input devices（输入设备），50~52
Feibush-Levoy-Cook algorithm（Feibush-Levoy-Cook算法），823~826, 852
Fields, video（场），1079
$F(u)$（f的傅里叶变换），625
$f(x)$, signal（通常表示信号），625
Filled primitives（填充图元），见Scan conversion, filled primitives
Filling（填充）
  circle（圆），99~100
  ellipse（椭圆），99~100
  pattern（图案），100~104
  polygon（多边形），92~99
  rectangle（矩形），91
Filling algorithms（填充算法），842, 979
  boundary fill（边界填充），979~980
  flood fill（泛填充），979~980
  seed fill（种子填充），980
  soft fill（软填充），979, 983~986
  tint fill（（颜色的）浓淡填充），979
Film recorder（胶片记录器），153~154, 570, 641
Filter（滤波器），136。另见Prefiltering, Postfiltering
  Bartlett（Bartlett滤波器），637
  box（盒式），636, 775
  convolution kernel（卷积核），631
  cut-off（截止），629
  finite support（有限支集），631
  FIR（有限脉冲响应），635
  Gaussian（高斯），635, 641
  highlighting/visibility（高亮度/可见性），332~333
  IIR（无限脉冲响应），635
  infinite support（无限支集），633
  negative lobes（负波瓣），634
  ringing（振荡），633
  support（支集），631
  triangle（三角），635
  windowed（窗口），633
Filtering（过滤，滤波），628~636, 695, 817
Finite element mesh（有限元网格），1041~1042
Fireworks（焰火），1031, 1055
Fitts' law（Fitts法则），371
Flaps（挡板），733
Flat tension mask CRT（平面拉伸荫罩CRT），160
Flicker, Focus, CRT（闪烁CRT焦点），12, 157
Flight simulator（飞行模拟器），4, 919
Flocking behavior（群体行为），1049
Flood fill（泛填充），见Filling algorithms, flood fill
Floodlight（泛光灯），733
Fluorescence, phosphor（荧光，荧光物质），157
Flux（通量），760。另见Irradiance
Flyback（快速回扫），见Vertical retrace
Focus, CRT（CRT焦点），156
Fog（雾），1031
Font（字体），见Character
Font cache（字体高速缓存），127
Foot switch（脚踏开关），352
Footmouse（脚踏鼠标），351
Footprint（足迹），106, 108
  interaction device（交互设备），350
Footprint processor（足迹处理器），888~889, 902。另见Parallel rasterization architectures, image-parallel
Foreshortened surface area（透视缩小曲面面积），761~762
Form fill-in（填充格式），398, 402~403
Form follows function (Bauhaus)（形式遵从功能），431
Forward differences（前向差分），511~514, 524~525
  adaptive（自适应），514
  in conic scan conversion（在圆锥曲线扫描转换中），955~957
4-connected region（四连通区域），979
Fourier analysis（傅里叶分析），625
Fourier transform（傅里叶变换），625。另见Inverse Fourier Transform
Fractal dimension（分形维数），见Dimension, fractal
Fractal models（分形模型），1020~1026

Frame buffer（帧缓存），14, 166~169, 178, 816, 856
distributed（分布式的），887~890
memory-access problem（内存存取问题），856, 873
Frame buffer synergy（帧缓存协作），816
Fresnel term（菲涅耳项），见F_λ
Front distance(F)（前距离），240, 265
Function key（功能键），195, 352, 372, 406, 412
Functional design, user interface（功能设计，用户界面），394~395, 404, 406, 429~430
Fundamental frequency（基频），623
Fusion frequency, critical（临界停闪频率），157

G

G(geometrical attenuation factor)（几何衰减因子），764~766
Gamma correction（Gamma校正），564~568, 600
Gaussian elimination（高斯消元法），1104, 1106
Gauss-Seidel iteration（Gauss-Seidel迭代），1109
General Electric（通用电气），919
General Electric NASA II（美国研发的一种飞行模拟器），899~900
Generalized cylinder（广义柱面），见Sweep
Generic commands（通用命令），405
Genesis effect（一个粒子系统作品名），840, 1032
GENESYS（一个动画系统名），1067
Genisco Space Graph（一个真三维显示的产品名），917
Genus of a polyhedron（多面体的方格），543
Geometric aliasing（几何走样），472
Geometric continuity（几何连续性），480~482
Geometric extents（几何范围），见Extents
Geometric modeling（几何造型），见Hierarchy, Structure network
interactive（交互式），315
object hierarchy（对象层次），288~291
Geometry Engine（几何引擎），878~879
Geometry matrix（几何矩阵），517, 520, 521
Geometry vector（几何向量），483~486, 488~489, 492, 510, 516~517
Geomod（一个几何造型系统的名字），547
Geomorphology（地貌学），1043
Gestalt rules（Gestalt规则），418~422
Gibbs phenomenon（Gibbs现象），633~634
GKS（计算机图形核心系统），16, 176, 436, 439
Global transformation matrix（全局变换矩阵），316
Glue operator（胶合操作），548
GMSOLID（一个体素造型系统的名字），560
Goniometric diagram（图），732
Good continuation, visual（好的连续性），418~419
Gouraud，见Shading, Gouraud
Gradient search（梯度搜索），1052
Graftals（描述植物结构与生成方法的名字），1027
Grammar-based models（基于文法的模型），1027~1031
Gram-Schmidt process（Gram-Schmidt过程），1102, 1112
Graphic alphabet（图表），425~426
Graphical languages（图形语言）。另见Animation, graphical languages
Graphics display processors（图形显示处理器），166~179
Graphics pipeline（图形流水线），866~871
Graphics subroutine packages（图形子程序包），25, 285
Graphics workstation（图形工作站），见Workstation
Grass, 1031
Gravity（重力），376, 378, 385~386, 388
Great arc（大弧），1063
Grid-line distance（栅格行间距离），946
Grids, positioning（栅格定位），360, 378, 427
Group technology（群组技术），539
Growth sequence（增长序列），569
Gupta-Sproull algorithms（Gupta-Sproull算法），见Antialiasing, Gupta-Sproull techniques
GWUIMS(George Washington University User Interface Management System)（一个用户界面管理系统），466

H

$\bar{H}$(halfway vector)（中间向量），731
Hair（毛发），1049
Halftone pattern（半色调模式），777
Halftoning（半色调），568~569, 897~898
Halfway vector（$\bar{H}$）（中间向量），731
Handles for user interaction（用户交互柄），386, 388
Hands, modeling of（手的建模），1049
Hardware binding design（硬件绑定设计），395, 404, 406, 429
Harmonic（谐波），623
Head mouse（头鼠标），351~352
Head-motion parallax（头运动视差），见Parallax
Head-mounted display（头盔显示器），357, 917~918
Help facility（帮助工具），412, 414, 457
Hermite curves（Hermite曲线），482, 483~488, 491, 515~516。另见Splines
Hexcone HSV color model（六棱锥HSV颜色模型），591
Hidden-line determination（隐藏线判定），见Visible-line determination
Hidden-surface elimination（隐藏面清除），见Visible-surface determination
Hierarchical B-spline refinement（层次B样条细化），见Splines, Hierarchical B-spline refinement
Hierarchical display list（分层显示列表），176, 291
Hierarchical menu selection（层次菜单选择），365~367

Hierarchical object selection（层次对象选择），362~364
Hierarchy（层次，层次结构），665。另见Display traversal, Structure network, Ray tracing
  automated generation（自动生成），707~710
  data（数据），344
  efficient traversal（有效遍历），707
  limitations in modeling（建模的限制），341
  object（对象），362, 397
  object modeling（对象建模），288~291
  procedure（过程），343
Highlighting, in geometric model（几何模型的加亮显示），332
Hither clipping plane（裁剪平面），见Clipping plane, *front*
HLS color model（HLS颜色模型），584, 592~595
Hologram（全息图），见Digital holography
Homogeneous coordinates（齐次坐标），204~208, 213~217
Homogenization（均匀化作用），1090
HSB color model（HSB颜色模型），585。另见HSV color model
HSV color model（HSV颜色模型），584, 590~592, 596~598
Hue, color（色调），574~575, 579, 584, 590
Human factors（人的因素），40, 391
Humans, animation（人类，动画），1076
HUTWindows, 459
HVC color model（HVC颜色模型），584, 594, 597
Hybrid parallelism（混合并行性），见Parallel rasterization architectures, hybrid-parallel
Hyperbola, spcification（双曲线描述），1007
Hypermedia（超媒体），5

I

i, $\sqrt{-1}$（虚数单位），625
$I_{dc\lambda}$ (depth cue color)（深度提示颜色），727
I_i (incident radiance)（入射辐射光亮度），762
I_p(point light source intensity)（点光源强度），724
I_r(reflected radiance)（反射辐射光亮度），763
Icon（图标），369, 395, 397~398
  design goals（设计目标），399
Iconic user interfaces（图标化用户界面），398~402
IFS, 见Iterated function system
Illuminant C（发光物C），581~582
Illumination（光照），另见Light source
  anisotropic（各向异性的），770~771
  Cook-Torrance（Cook-Torrance模型），764
  equation（方程），722
  global（全局），722, 775
  Hall model（Hall模型），785~786
  local（局部），775
  model（模型），721
  off-specular peak（偏离镜面的峰值），767
  Phong（Phong光照模型），729~730, 731, 769
  physically based models（基于物理的模型），760~771
  polarization state（偏振状态），771
  Torrance-Sparrow（Torrance-Sparrow模型），764
Image（图像）
  abstract（抽象），816
  blending（混合），835
  digitals（数字），817
  discrete（离散），817
  as primitive in PostScript（如同PostScript页面描述语言中的图元），1003
  rotation（旋转），851
  scaling（缩放），63, 821
  shearing（错切），822
  translation（平移），820
Image attributes（图像属性），849~850
Image compositing（图像合成），见Compositing, Image
Image composition language (ICL)（图像合成语言），842
Image irradiance（图像辐照度），763
Image parallelism（图像并行操作），见Parallel rasterization architectures, image-parallel
Image processing（图像处理），2, 820
Image storage（图像存储），815, 843~850
Image-assembly tree（图像-组装树），843
Image-composition architecture（图像合成体系结构），906~907。另见Parallel rasterization architectures, hybrid-parallel
Image-order rasterization（图像顺序光栅化），见Rasterization algorithms, scan line
Image-precision（图像精度），见Visible surface determination
Imaging model（成像模式），999
Implicit equations（隐式方程），478~528
Inbetweening（插值），1058
Incremental methods（增量方法），72
Independence, linear（线性无关），1091
Index of refraction, complex（$\tilde{\eta}_\lambda$）（折射率，复数），767
Input（输入），见Interaction handling
Input devices（输入设备），188~195, 348~359。另见Interaction tasks, Logical input device
Input pipeline（输入管线，输入流水线），68
Inside（内部），见Odd-parity rule, Winding rule, Filling algorithms
Instance, *object hierarchy*（实例，对象层次），291
Instance block（模块示例），325~327
Instance transformation（场景变换），见Display traversal
Integral equation method（积分方程法），792
Intel 82786 Graphics Coprocessor（Intel 82786图形协处理器），908

Intel i860 Microprocessor（Intel i860微处理器），864~866
Intensity（亮度），722~723, 760~763
  light（光），563~573
  of line as function of slope（直线的亮度是斜率的函数），80
  profile（轮廓，剖面），973
  radiant（辐照），761
  resolution（分辨率），568
Interaction（交互），见Logical input device
Interaction handling（交互处理），20
  sampling vs. event（采样与事件），42~44
  in SPHIGS（SPHIGS中的交互处理），328~331
  in SRGP（SRGP中的交互处理），40~52
Interaction tasks（交互任务），348~349, 358~381
  composite（复合），381~388
  position（定位），349, 358~361, 376~378, 381
  quantify（量化），349, 374~376
  select（选择），349, 361~373, 377, 381
  text（文本输入），349, 373~374
  3D（三维），376~381
  3D rotation（三维旋转），379~381
Interaction techniques（交互技术），348~349, 358~381, 394~395, 430, 451
  color specification（颜色规范），595~598
Interaction toolkits（交互工具箱），451~456
Interdot distance（点间距离），146
Interlaced scan（隔行扫描），179~180
Interleaved partitioning（交叉划分），888。另见Parallel rasterization architectures, image-parallel
International Standards Organization（国际标准化组织），见ISO
Internode（内节点），1028
Interobject communication（对象间通信），1019
Interobject reflections（对象间反射），758~761
Interpolation（插值），1051。另见Shading
  camera（照相机），1064, 1077
  color（颜色），598~599
  linear（线性），1063
  orientation（方位），1063
  spherical linear（球面线性），1063
Intersection formula（交集公式）
  between circle and line（圆和直线），1100
  between line and plane（直线和平面），1101
InterViews, 452
Inverse（逆），见Matrix, inverse of
Inverse Fourier transform（傅里叶逆变换），625
Invisible-surface table（不可见面表），685
Irradiance（辐照度），762
ISO (International Standards Organization)（国际标准化组织），16, 285
Isosurface（等值面），1035~1036, 1047
Italic（斜体），见Character, italic, Font
Item buffer（项缓存），782, 796
Iterated function system(IFS)（迭代函数系统），846~849
  modeling（建模），1026

J

Jaggies（锯齿状图形），14, 132, 628。另见Aliasing, Antialiasing
Joystick（游戏杆），191, 350~351, 355, 358, 360
Joyswitch（游戏开关），191, 360
JPL（Jet Propulsion Lab）（美国CIT的喷气推进实验室），607
Julia-Fatou set,（Julia-Fatou集合），1021
Just-noticeable color difference（可察觉的颜色差），578

K

k_a (ambient-reflection coefficient)（环境反射系数），723
k_d (diffuse-reflection coefficient)（漫反射系数），724
k_i (intrinsic color)（固有颜色值），722~723
k_s (specular-reflection coefficient)（镜面反射系数），729
k_t(transmission coefficient)（透射系数），754
Kerning（下标），129
Keyboards（键盘），194, 351, 358, 362, 372~373, 377, 404, 406
  chord（和弦），352
  logical input device（逻辑输入设备），46, 52, 188, 436
Key-frame（关键帧），1058
Kinematics（运动学），1074~1076
Knight, stained glass（骑士，彩色玻璃），1050
Knot, cubic curve（三次曲线结点），491, 496
Kochanek-Bartels spline（Kochanek-Bartels样条），见Splines, Kochanek-Bartels

L

$\bar{L}$(vector to light L)（到光源L的向量），723
Label structure element（标签，结构元素），324
Lambertian reflection（朗伯反射），723
Lambert's law（朗伯定律），724
Language（语言）
  form（形式），394
  meaning（含义），394
Lap-dissolve（重叠-溶解），841
Latency（延迟），874, 917
Lateral inhibition（横向抑制），735
Law of least astonishment（最小惊讶定律），404, 413
Lazy evaluation（惰性计算法），785
Lazy menus（下拉菜单），见Menus, drop-down
LCD（液晶显示器），641。见Liquid crystal display

Length, of a vector（向量长度），1095
Lerp（Linear interpolation的简写，线性插值），见Linear interpolation
Level surface（层次曲面，水平曲面），1035~1036, 1047。另见Isosurface
Lexical design（词法设计），395。另见Hardware binding design
Lexidata Lex90,（一个系统的名字），908
L-grammars（L文法），1027
Liang-Barsky line-clipping algorithm（梁友栋-Barsky裁剪算法），见Clipping, Liang-Barsky line algorithm
Light（光），见 Ambient light, Diffuse light, Illumination 等
Light-emitting diode (LED)（发光二极管），355
Light pen（光笔），193, 350
Light source（光源），另见Illumination
  attenuation（衰减），725~727
  attenuation factor（f_{att}）（衰减系数），725
  colored（有色光源），725~726
  cones（光锥，锥体），733
  directional（定向光），612, 725
  extended（扩展）613, 772~773, 789, 793, 799
  flaps（挡板），733
  foodlight（泛光灯），733
  goniometric diagram（测角图），732
  overflow（溢出），734
  point（点光源），612, 723
  sky（天空），799
  spotlight（聚光灯），733
  Warn controls（警告控制），731~734
Light valve display（光阀显示器），164
Lighting（发光光照），868~869, 878。另见Illumination
Lightness, color（明度，颜色），574~575, 579
Line（线，直线），另见Scan conversion
  contour（轮廓线），667
  haloed（光晕），667~668
  implicit equation（隐式方程），1112
  parametric equation（参数方程），1086
  style（型），109, 945
  thickness（宽度），945
Line drawing（线框图），9
Linear combination（线性组合），1085
Linear expression（线性表达式），895~896
Linear-list notations（线性表表示法），见Animation, linear-list notations
LinearSoftFill,（一个软填充算法），983, 984
Linguistic interaction task（语音交互操作），360, 374
LINKS-1（一种制作动画的系统），911
Liquid crystal display (LCD)（液晶显示器），161~163, 372
List-priority algorithms（列表优先级算法），672~680。另见Depth-sort algorithm, Binary space-partitioning tree
Local, control, cubic curres（三次曲线的局部控制），491, 493
Local transformation matrix（局部变换矩阵），304
Locator（定位器），20, 917
Locator logical device（定位逻辑设备），188, 436~439
  absolute（绝对），350
  continuous（连续），350~351
  direct（直接），350
  discrete（离散），350~351
  indirect（间接），350
  relative（相对），350
  3D（三维），328,355~357
  2D（二维），41~51
Logical input device（逻辑输入设备），41~42, 188, 349, 436~439。另见Keyboard, Locator
Logical visual organization（逻辑视觉机构），418~422
Logic-enhanced memory（逻辑增强内存），893~899。另见Parallel rasterization architectures, image-parallel
Look-up table (LUT)（查找表），845。另见Video look-up table
Low-pass filtering（低通滤波），629
Luminance, color（光强度），563, 575~576, 589, 590
Luminescence（发光），837, 839
Luminous efficiency function（光效率函数），576~577, 579
Luminous energy（发光能量），581
Luxo, Jr, 1052, 1077

M

m (RMS slope of microfacets)（RMS），765
Mach bands（马赫带效应），735~736, 738
Macintosh（美国Apple公司1984年推出的计算机系统），348, 353, 368, 371, 373, 397, 400, 411, 413, 425, 430, 441, 444
  operating system（操作系统），996
Macro command（宏指令），413
Mandelbrot set（Mandelbrot 集），1021~1022
Mandrill, smiling face of（山魈的笑脸），618
Man-machine dialogue（人机对话），见User-computer dialogue
Marble（大理石），1046
Marching cubes（移动立方体法），820, 1035, 1048
Marionette, graphical（图形的牵线木偶 ），1073
Marker, output primitive（标记，输出图元），28
Master, object hierarchy（母体，对象层次），291
Material percentage volume（体元中材料的百分比），1038
Material properties（材质属性），613, 723
Matrix（矩阵）

determinant of（的行列式），1103~1105
identity（单位），1103
inverse of（逆），1105~1106
multiplication of（乘法），1103
transpose of（转置），1105
Matrix addressing of display（显示器的矩阵寻址），162
Matrix Multiplier（矩阵乘法器），877~878
Matte volume（不光滑体），1038
McDonnell-Douglas, 919
Measure, logical input devices（度量，逻辑输入设备），42, 45~47
Measure of an interaction device（交互设备度量），436
Median-cut algorithm（中值分割算法）
color（颜色），600
Medical imaging（医学图像），816, 1037~1039
Megatek Sigma 70, 891
Memorization（记忆），413~414
Memory（内存）
recall（恢复），402, 414
recognition（识别），402, 414
Menus（菜单），402~404, 410~411, 420
appearing（呈现），367
drop-down（下拉），369
hierarchical（分级），364~367
order（命令），364
pie（饼图），370
pop-up（弹出），348, 367~368, 370
pull-down（下拉），49~50, 54~58, 367
pull-out（弹出），367, 369
static（静态），367
Metafile（元文件），333, 844
Metamer, color（条件等色），576, 579
Metaphor, user interface（隐喻，用户界面），394
Mickey UIMS（Mickey用户界面管理软件），455~456
Microfacets（微面元），764
Micropolygons（微多边形），811
Microsoft（微软公司），348, 368, 373, 382
MIKE UIMS（MICKY用户界面管理软件），465
MIMD（多指令多数据流），见Parallelism
MIP map（MIP图），826
Mitering（斜面接合），962
Mnemonic command names（助记符），413
Modality, attribute specification（模态，属性说明），30
Mode, input devices（模式，输入设备），44
Modeling（建模，造型），286~288
Modeling（建模，造型），见Geometric modeling
Modeling transformation（模型变换），见Transformation, modeling
Modes（模式）
context-sensitive syntax（上下文敏感语法，上下文相关语法），417
harmful（有害），414
useful（有用），414
user interface（用户界面），414~418
Mole interaction device（鼹鼠标交互设备），351
Molecular modeling（分子建模），607, 698, 1047~1048, 1057
Monochrome（单色），见Bilevel
Moon, non-Lambertian surface（月亮非朗伯曲面），763
Motion blur（运动模糊），615, 789
Mountains, fractal（分形山），1022~1024
Mouse（鼠标），15, 191, 350~351, 354, 357, 360~361, 364, 371, 373, 376, 380, 406, 411, 416。另见Locator
Mouse ahead（鼠标在前），438,
Moving-points path（运动点路径），1061
Multipass transformation（多重变换），821~822, 828~832
Multiple control points, curves（曲线多控制点），495~496, 499
Multiple instruction multiple data (MIMD)（多指令流多数据流），见Parallelism
Multiprocessing（多重处理），873~876
Munsell color-order system（Munsell颜色排序系统），574

N

$\bar{N}$(surface normal)（曲面法线），723
n (specular reflection exponent)（镜面反射指数），729
Name set, PHIGS（名称集），332
NASA（美国宇航局），607
NASA II（美国研发的一种飞行模拟器），见General Electric NASA II
National Television System Committee(NTSC)（美国国家电视系统委员会），180, 589~590
Natural cubic spline（自然三次样条），491
Natural language dialogue（自然语言对话），402~403
Natural phenomena（自然现象），1043
Necker cube illusion（Necker立方体幻觉），608
Negative orientation（负向），1105
Network（网络），见Structure network
Newell-Newell-Sancha algorithm（Newell-Newell-Sancha算法），见Depth-sort algorithm
NeWS window system（NeWS窗口系统），440~444, 452
Newton-Raphson iteration（Newton-Raphson迭代），699, 1109~1110
Newton' s method（牛顿法），1110
NeXT, 353, 400, 468
Nicholl-Lee-Nicholl algorithm（Nicholl-Lee-Nicholl算法），见Clipping, Nicholl-Lee-Nicholl line algorithm

Noise function（噪声函数），1016
Noise-based modeling（基于噪声的建模），1043
Noise-based texture mapping（基于噪声的纹理映像），1016~1018
Noll box（Noll盒），355
Nominative information, coding（主格信息，编码），422~424
Nonexterior rule（非外部规则），965
Nonspectral color（非光谱颜色），583
Normal（法线）
  to bicubic surface（双三次曲面），522~523
  to plane（平面），217, 476
  to quadric surface（二次曲面），529
  to surface（曲面），807
Nomalizing transformation（规格化变换）
  parallel（平行），260~267
  perspective（透视），268~271
Notches, in scan conversion（扫描转换中的凹槽），971
NTSC（国家电视系统委员会），见National Television System Committee
Number wheel interaction technique（数字轮交互技术），375
NURBS（非均匀有理B样条）见Splines Nonuniform rational B-splines (NURBS)
Nyquist rate（奈奎斯特频率），627~628, 788~790

O

O_{dA}(object diffuse color)（物体漫反射颜色），726
O_{sA} (object specular color)（物体镜面反射颜色），730
Object buffer（物体缓存），672
Object hypothesis（目标假设），608
Object modeling（物体建模），见Geometric modeling
Object parallelism（物体并行处理），见Parallel rasterization architectures, object-parallel
Object placement automatic（自动物体布局），1050
Object-order rasterization（物体序列光栅化），见Rasterization algorithms, z-Buffer
Object-precision（对象精度），见Visible surface determination
Octree（八叉树），550~555, 559~560
  linear notation（线性记号），554~555
  neighbour finding（查找邻节点），552~554
  PM octree（PM八叉树），555
  regularized Boolean set operations（正则布尔集合运算），552~553
  rotation（旋转），552
  visible-surface determination（可见面判定），695~698
Odd-parity rule, polygons（多边形奇数规则），34
Omnimax film format（Omnimax胶片格式），230
Opacity（不透明性），754
Opaque operator（不透明运算符），839
OPEN LOOK（Sun Microsystem公司提供的一种图形用户界面），452
Open Software Foundation (OSF)（开放软件基金会），430, 452
Ordered dither（有序抖动），568
Ordinal information, coding（次序信息，编码），422~424
Orientation of a basis（基底方向），1105
Orthogonal（正交的），1102
Orthonormal basis（正交基），1102
Oslo algorithm, cubic curves（三次曲线的Oslo算法），510
Ostwald color-order system（Ostwald颜色排序系统），574
Outcode, clipping（外码），113, 271
Output pipeline（输出流水线），68。另见Rendering
Output primitives（输出图元），18, 19
  area-defining（区域定义），107
  geometric modeling（几何建模），296~298
  raster graphics（光栅图形），26~30
  respecification（重新定义），61~63
Overhauser splines（Overhauser样条），见 Splines, Catmull-Rom样条

P

Page mode（页模式），858
Page-description languages（页描述语言），998~1006
Painter's algorithm（画家算法），674
Painting（着色）
  implementation（实现），45
  versus drawing（与绘图），61
PAL television standard（PAL制式电视标准），180
Palette（调色板），41
Pan-zoom movie（移动-放缩电影），1059, 1064~1065
Parabolas, specification（抛物线），1007
Parallax（视差），915
Parallel front-end architectures（并行前端体系结构），880~882
Parallel projection（平行投影）
  front（前向），232, 250
  oblique（斜），232
  orthographic（正交），232, 250
  side（侧面），232
  top（顶部），232, 251
Parallel rasterization architectures（并行光栅体系结构），887~899
  hybrid-parallel（混合并行操作），902~907
  image-parallel（图像并行处理），887~899
  object-parallel（物体并行处理），899~902
Parallelism（并行性），875~876

multiple instruction multiple data(MIMD)（多指令流多数据流），876
single instruction multiple data(SIMD)（单指令流多数据流），875~876
Parallelogram rule（平行四边形法则），1084
Parametric surfaces（参数曲面），471, 516~528
Palametric bivariate polynomial surface patches（参数双变量多项式曲面片），472。另见Splines
Parametric continuity（参数连续性），480~482
Parametric cubic curves（三次参数曲线），见Curves, parametric cubic
Parametric polynomial curves（参数多项式曲线），见Curves, parametric polynomial
Parametric representation, in line clipping（参数表示），118
Parametric velocity（参数速率），482
Paraxial ray（旁轴射线），787
Parity rule（奇偶规则），965
Particle systems（粒子系统），1031~1034
Partitioned frame buffer（分块帧缓存），见Parallel rasterization architectures, image-parallel
Patch（片，面片），见Splines, surface patch
Path tracing（路径跟踪），792
Pattern, output attribute（图案，输出属性），34~37
Pattern filling（图案填充），见 Filling, pattern
Pattern mapping（模式映射），见 Surface detail
Pattern recognition interaction technique（模式识别交互技术），370, 372
P-curves（P曲线），1067
Pel（图像单元），1
Pen, sonic（笔，声音），357
Pen polygon（笔触多边形），1010
Pen style（笔型，画笔类型），109
Pencil（画笔），787
test（测试），1058
tracing（光束跟踪），787
Perceived intensity of light（光敏强度），563
Perceptually uniform color space（感觉一致的颜色空间），584
Peridot（一种构造菜单和对话框的工具），456, 464
Perspective transformation（透视变换），见Transformation, perspective
Persistence, phosphor（余辉，荧光物质），157
Person-to-person dialogue（个人对话），393
Perspective foreshortening（透视缩小效应），231, 280
Perspective projection（透视投影）
one-point（一点），231
three-point（三点），232
two-point（两点），232, 247, 249
Perspective transformation（透视变换），657~660
Phantom vertices, cubic curves（三次曲线假设顶点），495
Phase（相位），977
angle（相位角），626
shift（相位移），624
PHIGS, 16。另见SPHIGS
PHIGS+, 16, 862
Phoenix Data Systems Insight（Phoenix数据系统理解），914
Phong（Phong光照模型），见Illumination, Phong, Shading, Phong
Phosphor（磷光物质，荧光物质），155, 157, 564, 583, 589
Phosphorescence（磷光），157
Photometer（光度计），564, 587
Photorealism（照片真实性），605
renderings（绘制），16
Physically based modeling（基于物理的建模），1039~1047
Pick（拾取）
correlation（关联拾取），48~50
correlation implementation（关联拾取实现），338
correlation object hierarchy（关联拾取对象层次），329~331
identifier（标识符），331
logical device（逻辑设备），188, 436~437
logical input device（逻辑输入设备），42, 330
point（点），113
windows（窗口），113, 339
Picture hierarchy（画面层次），见Hierarchy, object
Piecewise approximation（分段逼近），见Curves
Piecewise continuous polynomial（分段连续多项式），478
Pimple（小突起），977
Pipeline front-end architectures（流水线前端体系结构），877~880
Pipeline rasterization architectures（流水线光栅化体系结构），883~886
image-order（图像序），885~886
object-order（物体序），883~885
Pipelining（流水线），874
Pitch, shadow mask CRT（阴罩CRT的间距），159
Pitteway, 75
Pixar, 280
Pixar Image Computer（Pixar图像计算机），914
PixBlt（像素块传送指令），166, 863。另见BitBlt
Pixel（像素），1
geometry（几何），133
cache（高速缓存），885
replications（复制），105
Pixel-Planes, 894~897, 902
Pixel-Planes 5, 905

Pixel-stream editor（像素流编辑器），810~811
Pixmap（像素图），13，166。另见Bitmap，Canvas
pattern（图案），36
Planar geometric projection（平面几何投影），230
Plane（平面）
equation（平面方程），216，476，895~896
implicit equation（隐式方程），1098~1099
parametric equation（参数方程），1087
Plane-parallel copy（平面并行拷贝），998
Plane-serial copy（平面串行拷贝），998
Plants（植物），1027~1031
Plasma panel display（等离子平板显示器），163~164
Plastic slip（塑料滑块），1041
Plotter（绘图仪），148~154，564
drum（鼓式），149
electrostatic（静电），149
flatbed（平板），148
ink-jet（喷墨），152
Point light sonrce（点光源），见Light source，point
Point of evaluation（估值点），84
Polhemus，3D digitizer（Polhemus三维数字化仪），
355~357
Polling（轮询），见Sampling
Polygon（多边形）
area of（多边形面积），1112
interior test（内部测试），34
Polygon clipping（多边形裁剪），见Clipping，polygon
Polygon creation（多边形生成），383~384，388
Polygon mesh（多边形网格），471，472，473~476，871，
920~921
consistency（一致性），475~476
Polygon table（多边形表），681
Polyhedron（多面体），543
simple（简单多面体），543
SPHIGS，297
Polyline（折线），27。另见Scan conversion，polylines
Popularity algorithm，color（普及算法），600
Portability（可移植性），15
application programs（应用程序），285
Positioning interaction task（定位交互任务），见
Interaction task
Positive orientation（正向），1105
Postconditions（后置条件），465~467
Postfilterings（后置滤波），642，818
PostScript（一种页面描述语言），441~442，514，
923~924，963，999~1006，1081
Potentiometer（电位器），352，375，594
POWER IRIS，889~893
Preconditions（前提条件），465~467
Prefiltering（前置滤波），642
Primaries，color（基色），577，585，587~588
Primitive instancing（基本实体举例法），539
Primitives（图元），见Output primitives
Printer（打印机）
dot-matrix（点阵），148
laser（激光打印机），151
thermal-transfer（热传导），152~153
Priority，display（优先级，显示），302
Procedural models（过程模型），1018~1019
Processing mode，keyboard（处理模式，键盘），46
Progressive refinement（逐步细化），812。另见
Radiosity
Projection（投影），229
axonometric parallel（轴测平行），233，610
cabinet parallel（斜二测平行），235~236，258
cavalier parallel（斜等测平行），235~236，252，258
isometric parallel（等轴平行），233，252
oblique（斜），234，610
oblique parallel（斜平行），233
orthographic parallel（正平行），232
parallel（平行），230，243
perspectives（透视），230，611
Projection implementation（投影实现）
parallel（平行），260~267
perspectives（透视），268~271
Projection matrix（投影矩阵）
general（广义），258
orthographic（正交），256
perspective（透视），254~256
Projection plane（投影平面），230
Projection reference point (PRP)（投影参考点），238
Peojection textures（投影纹理），1018
Projector（投影线），230
Prompts（提示），411~412
Proximity，visual（近似性，视觉），418~419
Pruning，structure network（修剪，结构网络），340
Pseudorealism，rendering（伪真实感，绘制），见
Photorealism
Pulldown menu（下拉菜单），见Menus，pulldown
Pulse function（脉冲函数），629
Purity（纯度），见Excitation purity
Put that there（一个界面名），403
Putty（油灰，腻子），1042

Q

Quadratic interpolation（二次插值），497
Quadratic polynomials（二次多项式），504
Quadric surface（二次曲面），471，473，528，546，698
Quadrilateral mesh（四边形网格），见Polygon，mesh
Quadtrees（四叉树），550，697
for image storage（图像存储的），846

linear notation（线性表示），554~555
neighbor finding（邻节点查找），552~554
Quantity interaction task（定量交互任务），见Interaction task
Quantitative invisibility（不可见量），666
Quantization effects（量化效应），见Antialiasing
Quaternion（四元数），1063, 1081
Question-answer dialogue（问答对话），403
QuickDraw, 16, 963

R

$\bar{R}$ (direction of reflection)（反射方向），729
Race conditions（竞态条件），449
Radiance（面辐射强度），761
Radio button interaction technique（单选按钮交互技术），369, 426, 452
Radiosity (B)（辐射度），793
Radiosity methods（辐射度方法），722, 753, 775, 793~806
ambient term（环境项），801
color bleeding（渗色），795
delta form factors（△形状因子），798
extended form factors（扩展形状因子），806
z-buffer（z缓存），806
form factor (F_{i-j})（形状因子），794~799, 803~804
form factor reciprocity relationship（形状因子相互关系），794
gathering（聚集），801
Gauss-Seidel iteration（Gauss-Seidel迭代），795
Gouraud shading（Gouraud明暗处理），795, 806
hemicube（半立方体），796
mirror world（镜中世界），805~806
progressive refinement（逐步细化），800~804
radiosity equation（辐射度方程），793~795
ray tracing（光线跟踪），803, 804~806
reflection frustum（反射截头锥体），805
shooting（发射），801
specular reflection（镜面反射），804~806
substructuring（子结构），799~800
vertex radiosities（顶点辐射度），795
RAMDAC, 见Video look-up table
Random dither（随机抖动），568
Random scan（随机扫描），9, 12
Random-scan display processor（随机扫描显示处理器），184~187
Rapid recall（快速检索），392
Raster（光栅），11
display（显示器），166, 187, 851~861。另见Cathode ray Tube
Raster graphics package（光栅图形软件包），25
Raster image processor（光栅图像处理器），14, 71
Raster lines（光栅行），11
Raster operation（光栅操作），177~178。另见BitBlt, PixBlt, Write mode
Raster scan（光栅扫描），12
Raster Technologies Inc., 882
Rastering（光栅化），641
Rasterization（光栅化）606, 870~871, 882。另见Scan conversion, Shading, Visibility determination
polygon（多边形），883~885, 891~893, 896
scan-line algorithms（扫描线算法），870
speed of（速度），873
z-buffer algorithm（z缓存算法），870~871
RasterOp, 172~174, 368。另见Raster operation
Raster-scan generator（光栅扫描生成器），167~168
Ratio information, coding（比率信息，编码），422~424
Ray, eye（光线，视线），701
Ray casting（光线投射），701。另见Ray tracing in b-rep Boolean set operations, 546
Ray Casting Machine（光线投射设备），901
Ray tracing（光线跟踪），701~715, 753, 776~793, 804~806。另见Distributed ray tracing
adaptive supersampling（自适应超采样），714~715
adaptive tree-depth control（自适应树深度控制），783
antialiasing（反走样），714~715
area sampling（区域采样），786~787
beam tracing（光束跟踪），787, 793
bounding volume（包围体），705~706
computing intersections（求交），702~704, 705~706
cone tracing（锥跟踪），786~787
constructive solid geometry（构造立体几何学（CSG）），712~713
distributed ray tracing（分布式光线跟踪），788~792
efficiency（效率），704~712, 782~785
from light sources（从光源出发的），792
hierarchy（层次结构），706~712
Kay-Kajiya bounding volume（kay-kajiya包围体），705~708
light buffers（光缓存），783~784
mass properties（质量属性），719
numerical precision problems（数值精度问题），781
octrees（八叉树），711
pencil tracing（光束跟踪），787
polygons（多边形），703~704
primary rays（主光线），778
ray classification（光线分类），784~785
ray tree（光线树），778~779, 781
real time（实时），890
recursive（递归），722, 776~793
reflection mapping（反射映射），782
reflection rays（反射线），778
refraction rays（折射线），778
regularized Boolean set operations（正则布尔集合运

算），712~713
secondary rays（从属光线），778
shadow rays（阴影光线），777
shadows（阴影），776~777
slab（面对），705
space-time（空间－时间），1078
spatial partitioning（空间划分），710~712
spheres（球体），702~703
stochastic sampling（随机采样），788~791
surface normal（曲面法向量），703~704
Ray-tracing architectures（光线跟踪体系结构），910~912
Reaction Handler, The, 458
Realism（真实感）见Photorealism
Reconstruction（重构），619, 636~642
sample and hold（采样和保持），641
Rectangle write（矩形写），102
Recursive subdivision, curves and surfaces（曲线和曲面的递归划分），511, 512~514, 526~528
Redo command（Redo命令），409~410
Reference model（参考模型），945
Refinement, curves（曲线的细化），507
Reflected light（反射光），574
Reflection（反射），614。另见Diffuse reflection, Direction of reflection, Specular reflection
Reflection mapping（反射映射），758~761, 782
antialiasing（反走样），759
Refraction（折射），614
Refraction vector（$\bar{T}$）（折射向量），757
Refresh（刷新），10, 12, 155, 157~158, 856。另见Display traversal
Refresh rate（刷新率），155, 170
Regeneration strategy（更新策略），996
Region（区域），979
boundary-defined（边界定义的），980
interior-defined（内容点定义的），980
Region checks, clipping（区域检测，裁剪），113
Regularized Boolean set operations（正则布尔集合运算），535~539, 547~548, 561, 702。另见Constructive solid geometry
for binary space partitioning trees（二元空间划分树），556~557
for boundary representations（边界表示），546~547
compared with ordinary Boolean set operations（与一般布尔集合运算的比较），536~538
for octrees（八叉树），552~553
for sweeps（扫掠），541
ray tracing（光线跟踪），712~713
Relaxation techniques（渐近技术），1072
Renderers,design（绘制器，设计），810~812
Rendering（绘制），606
PostScripi, 999
Rendering, SPHIGS, 见Display traversal type（类型），323
Rendering equation（绘制方程），776, 792
Rendering pipeline（绘制流水线，绘制流程），806~812
z-buffer（z缓存），806~808
global illumination（全局光照），809
Gouraud shading（Gourand明暗处理），806~807
list-priority（列表优先级），808
local illumination（局部光照），806~809
Phong shading（Phong阴影处理），808
ray tracing（光线跟踪），810
RenderMan, 280, 281, 811
Repeat command（Repeat命令），416
Request mode（请求模式），436~439
Resampling（重采样），817, 832
Residual（残差），946~947
Resolution（分辨率），147, 153, 158, 160, 183, 860~861
interaction tasks（交互任务），359, 375
Response time（响应时间），见Latency
Retained-mode graphics（保留模式图形），见SPHIGS
Reyes, 811~812
RGB color model（RGB颜色模型），584~587, 596~597
ρ(bidirectional reflectivity（双向反射率），763
ρ_d(diffuse bidirectional reflectivity)（双向漫反射率），763
ρ_s(specular bidirectional reflectivity)（双向镜向反射率），763~764
RIFF, 849
Right justification（正确判断），129
Ringing（振荡），633
RIP, 见Raster image processor
Root, structure network（根，结构网络），299
Root finding（求根），1109
Rotation（旋转）
dynamic（动态），387
3D（三维），215
2D（二维），203, 206
Rotation interaction task（旋转交互任务），见Interaction task
Rothstein code（Rothstein码），821
Route sheet（路由表），1059
Row-preserving map（行保持映像），828
Rubberband（橡皮筋线）
circle drawing（画圆），383
ellipse drawing（画椭圆），383
feedback（反馈），51~52
line drawing（画线），382
rectangle drawing（画矩形），382

Run-length encoding（行程编码），845, 854

S

S_i(shadow coefficient)（阴影系数），745
Sample, SRGP input mode（样本，SRGP图形包输入模式），44~45
Sample and hold（采样和保持），641
Sample mode（样本模式），437
Sample-driven interaction（样本驱动交互），42~44
SampleLocator, 438, 439
SamplePick，437
SampleString，437
Sampling（采样），619
  area sampling（区域采样），621~623
  importance sampling（重要性采样），790~791
  point sampling（点采样），619~621
  sampling theory（采样理论），621, 623~628
  unweighted area sampling（未加权区域采样），621, 714
  weighted area sampling（加权区域采样），622
Saturation, color（饱和度，颜色），574~575, 579~580, 584, 590~591
Saturn's rings（土星环），1045
Scalar field（标量场），1035
Scaling（缩放）。另见Image scaling
  differential（不同的），202,210
  3D（三维），215
  2D（二维），202, 205~206
Scan conversion（扫描转换），14, 71, 870, 945~965。另见Rasterization
  antialiased lines（反走样直线），132~142
  characters（字符），127~132
  circles（圆），81~87
  conics in gray-scale（灰度等级圆锥），957
  ellipse（椭圆），88~91
  evaluation of algorithms（算法评价），946
  filled primitives（填充图元），964~965
  general circles（广义圆），949~951
  general conics（广义圆锥），951~961
  general ellipses（广义椭圆），951~961
  general lines（广义（直）线），948~949
  incremental line（增量线），73~74
  line（线），72~81
  midpoint circle（中点圆），83~87
  midpoint ellipse（中点椭圆），88~91
  midpoint line（中点线），74~81
  outline primitives（轮廓线图元），81
  polylines（折线），949
  rectangle（矩形），91
  text（文本），976~979
  text strings（文本字符串），130~132
  thick general curves（宽一般曲线），963~964
  thick primitives（宽图元），104~109, 961
  trapezoid（梯形），99
  triangle（三角形），99
  triangle mesh（三角形网格），99
Scan line（扫描线），751, 883~884
Scan Line Access Memory (SLAM)（扫描线存取存储器），897~899
Scan-line algorithm（扫描线算法），97, 680~686, 715, 737
  regularized Boolean set operations（正则布尔集合运算），684
Scan-line system（扫描线系统），见Pipeline rasterization architectures, image-order
Scanner（扫描仪），195~197, 374
Scefo, 1066~1067
Scene radiance（场景辐射光亮度），763
Schröder stairway illusion（楼梯幻觉），608~609
Scientific visualization（科学计算可视化），5, 1057
Scissoring（截剪），71, 110, 143, 870, 924, 995
Score conductor's（指挥的乐谱），1069
Screen, printing（屏幕），570
Screen angle, printing（屏幕角度），568
Scroll（滚动），182, 365
S-dynamics（一个系统名），1069
SECAM television standard（SECAM制式电视标准），182
Second-order differences, circle scan conversion（圆扫描转换二阶差分），85~87
Sectoring（分区），747
Segment（段），176, 442
Segment storage, local（段存储，本地），176
Segments, in GKS（图形核心系统中的段），16
Select interaction task（选择交互任务），见Interaction task
Selection（选择）
  by naming（通过命名选择），362
  by pointing（点取选择），362~364
Self-luminous object（自发光物体），574~575, 580
Self-occlusion（自遮挡），724
Self-similarity（自相似性），1020
Semantic design（语义设计），见Functional design
Sequence（序列），见Action
Sequencing design（顺序设计），394~395, 404, 406
Sequencing rules（顺序规则），393
Serpent UIMS（Serpent用户界面管理系统），466
Server-client window management system（客户机-服务器窗口管理系统），440~441
Set（集，集合）
  boundary points（边界点），535
  closed（闭），535

closure（闭包），535
interior（内部），535
open（开），535
regular（正则），536
regularization（正则化），535
Shade, color（色深，颜色），574
Shade trees（明暗处理树），810
Shader（明暗处理模块），810~811
Shading（明暗处理），868~869，870~871。另见Rasterization
constant（flat, faceted）（恒定），734~735
dot-product interpolation（点积插值），739
Gouraud(intensity interpolation)（Gouraud亮度插值），598~599，613，736~738
interpolated（插值的），613，735
model（明暗模型），721
Phong(normal interpolation)（Phong模型（法向插值）），738~739
polygon mesh（多边形网），735~739
problems with interpolated shading（插值明暗处理问题），739~740
Shadow map（阴影图），752
Shadows（阴影），614，745~753，910
fake（虚假），746
penumbra（半影），773
percentage-closer filtering（百分比比率更接近滤波），753
region of influence（影响区域），749
scan-line（扫描线），746
shadow polygon（阴影多边形），749
shadow volume（阴影体），749，772~773
shadow volume binary-space partitioning tree（阴影体的二元空间划分树），751
sphere of influence（影响球），749
two-pass z-buffer algorithm（两遍的z缓存算法），751~753
two-pass object-precision algorithm（两遍的对象精度算法），746~749
umbra（全影），772
Shapes（形状）
algebra（形状代数），992~995
boolean operations on（形状的布尔运算），993~995
data structure（数据结构），992
Shear（错切），
3D（三维），216
2D（二维），207
Shielding, viewports（屏蔽，视口），302
Shrinking raster（收缩光栅），157
Side effects（副作用），409
SIGGRAPH（Special Interest Group on Graphics的缩写），15，519
SIGHT（一个系统名），911
Signal（信号），618
continuous（连续），618
discrete（离散），618
frequency spectrum（频谱），623，628
Silhouette edge（轮廓边），698
Silicon Graphics POWER SERIES，879，889~893
SIMD（单指令多数据流），见Parallelism
Similarity, visual（相似性），418~421，424
Simulation（模拟，仿真），5，1057
Sinc function（sinc函数），632~635，966
Singer/Link，919
Single instruction multiple data（SIMD）（单指令多数据流），见Parallelism
Skeletal motion（骨骼的运动），1049
Skeleton（骨架，框架），1061
Sketchpad（略图，草稿，示意图），8，1040，1071，1076
Skin（皮肤），1014，1049
Skitter，378
Slerp，见Interpolation, spherical linear
Sliver polygons（狭长多边形），95
Slow-in/slow-out（渐入/渐出），1060
Smalltalk，996，1073
SmethersBarnes prototyper（SmetherBarnes原型），454，464
Smoke（烟），1031
Snakes（蛇），1048
Snell's law（Snell定律），756，778
Soft objects（软物体），1047
Solid angle（立体角），760~761
Solid modeling（实体造型），532~562，572。另见Geometric modeling
features（特征），561
point classification（点分类），556
robustness（健壮性），561
toleranced objects（带有容差的物体），561
untoleranced objects（无容差的物体），561
user interface（用户界面），561
Solid modeling representation（实体造型表示）
comparison（比较），559
conversion（转换），559~560
evaluated model（已求值模型），559~560
unevaluated model（未求值模型），559~560
Solid textures（立体纹理），1015~1018
Sorting, bucket（桶排序），886，903~904
Span（生成），883~884，886，980
of a set of vectors（向量集的生成），1085
Span calculation（生成计算），99，108
Spatial integration（空间积分），568，599
Spatial occupancy enumeration（空间位置枚举），549~550，558，559

Spatial partitioning（空间划分）, 664, 686, 710~712
adaptive（适合的）, 664
Spatial partitioning representations（空间划分表示）, 548~557
Spatial resolution（空间分辨率）, 568~569
Spatial subdivision（空间子分）, 见Spatial partitioning
Spatial task（空间任务）, 360, 374
Special orthogonal matrix（特殊正交矩阵）, 207, 216
Spectral energy distribution（光谱能量分布）, 575~576, 579~580, 582
Spectral sampling（光谱采样）, 773
Spectral-response functions（光谱响应函数）, 576~577
Spectroradiometers（光谱辐射计）, 576, 582, 586
Spectrum（谱, 频谱）, 575
Specular reflection（镜平反射）, 728~731
coefficient of(k_s)（镜面反射系数）, 729
color shift（色彩漂移）, 768
exponent(n)（指数）, 729
Speech recognition（语音识别）, 见Voice recognition
Speech synthesizer（语音合成）, 见Voice synthesizer
Speed of learning（学习的速度）, 391~392, 403, 418
Speed of use（使用的速度）, 391~392, 398, 403, 411
SPHIGS, 16, 285~346, 482。另见Hierarchy, structure network, Structure network, and Structure
interaction handling（交互处理）, 328~331
object modeling（物体建模）, 304~314
output attributes（输出属性）, 298
output primitives（输出图元）, 296~298
SPHIGS, screen updating（SPHIGS, 屏幕更新）, 见Viewing operations, 299~302
Splines（样条）, 482
Bézier curves（Bézier曲线）, 482, 488~491, 515~516
Bézier surfaces（Bézier曲面）, 521~522
B-spline curves（B样条曲线）, 491
B-spline surfaces（B样条曲面）, 522
β-splines（β样条）, 505~507, 515~516
Catmull-Rom（Catmull-Rom样条）, 504~505, 515~516
deformations（变形）, 1014
hierarchical（分层的）, 1012~1014
hierarchical B-spline refinement（层次式B样条细化）, 510
Kochanek-Bartels, 507, 515~516
nonuniform, nonrational B-spline curves（非均匀非有理B样条曲线）, 495~500, 515~516
nonuniform rational B-splines（NURBS）（非均匀有理B样条曲线）, 502, 547
nonuniform, rational cubic polynomial curve segments（非均匀有理三次多项式曲线段）, 501~504
use in animation（用于动画）, 1060
used for characters（用于字符的）, 131
Spot size（点尺寸）, 见Dot size
Spotlight（聚光灯）, 733
Sprite（精灵）, 180, 1065
Square wave（方波）, 626
Squash and stretch（挤压与拉伸）, 1077
SRAM（Static random-access memory）, 857
SRGP, 16, 25~66, 436
framebuffer control（帧缓存控制）, 52~60
interaction handling（交互处理）, 40~52
output attributes（输出属性）, 30~38
output primitives（输出图元）, 26~30
SRGPcopyPixel, 69
implementation（实现）, 132
Staging（分解）, 见Animation, staging of
Staircasing（楼梯状）, 132
Standard affine plane（标准仿射平面）, 1089
Standard affine, 3-space（三维空间的标准仿射）, 1089
Standard graphics pipeline（标准图形流水线）, 见Graphics pipeline
Standardized object（标准化物体）, 304
Standards（标准）, 15
graphics packages（图形软件包）, 285
Starter kit of commands（命令的初学者工具箱）, 392, 413
State variables（状态变量）, 457~464, 466
State-transition diagram（状态转换图）, 363, 382, 383, 384, 404, 457~464
Static random-access memory（静态随机访问存储器）, 见SRAM
Stellar GS2000（一个图形系统的名字）, 889, 910, 912~913
Steradian（单位立体角）, 760~761
Stereo（立体）,
display（立体显示）, 915~916
pair（对）, 616, 376
Stereopsis（立体观测）, 616~617
Stimulus-response (S-R)compatibility（刺激－响应兼容性）, 359, 378~380
Stochastic sampling（随机采样）, 788~791, 812
Storage（存储）, 见Structure network
Storage, primitives（图元存储）, 见Structure network
Storyboard（故事板）, 430, 1058
Stream erosion（溪流侵蚀（模型））, 1043
String logical device（串逻辑设备）, 436
Stroke（笔画）, 9
logical input device（逻辑输入设备）, 42
Structure（结构）, 293, 295。另见Hierarchy, structure network
editing（编辑）, 295, 324~327
elision（省略）, 340
example（示例）, 311~314
hierarchy（层次）, 308~314
modeling transformation（模型变换）, 304~308, 315, 336
pick correlation（关联拾取）, 329
referral（分派, 安排）, 341。另见Display traversal, Hierarchy

Structured display file（结构化显示文件，Hierarchical display list
Structures（结构），16
Stucco（（用灰泥）粉饰，这里指纹理），1055
Style guide,user interface（风格指导，用户界面），430
Subdividing cubic curves（三次曲线分割），507~510
Subpixel area-subdivision algorithms（子像素区域细分算法），693
Subspace,affine or linear（子空间，仿射或线性），1088
Subtractive color primaries（减性基色），587~588
Summed area table（求和面积表），826~827
SunView, 452~454
Superposition, principle of（超位置原理），966
Supersampling（超采样），620
  adaptive（自适应），643~644
  stochastic（随机），644~645, 788~791
Support, of filter（滤波器的支集），136, 631
Surface detail（曲面细节），741~745
  bump mapping（凹凸映射），744
  displacement mapping（位移映射），745
  solid texture（实体纹理），745
  surface-detail polygons（曲面细节多边形），741
  texture mapping（纹理映射），741~744, 800
  transparency mapping（透明映射），745
Surface modeling（曲面造型），471
Surface normal（曲面法线）
  in Boolean set operations（布尔集合运算中的），537~539。另见Normal, to surface
  for implicitly defined surfaces（隐式曲面的），1102
Surface patch（曲面片），另见Curved surfaces
  Blinn display algorithm（Blinn显示算法），698~699
  Catmull recursive-subdivision display algorithm（Catmull递归细分显示算法），698~699
  Clark display algorithm（Clark显示算法），700~701
  cracks（裂缝，断裂），527~528, 701, 737
  Lane-Carpenter display algorithm（Lane-Carpenter显示算法），700~701
  Whitted display algorithm（Whitted显示算法），698~699
Surface table（面表），684
Sutherland-Hodgman Polygon-Clipping algorithm（Sutherland-Hodgman多边形裁剪算法），见Clipping, Sutherland-Hodgman Polygon-Clipping algorithm
Sweep（扫掠），540~541
  general（一般），540
  rotational（旋转），540
  translational（平移），540
Symbol（符号）201, 363
Syntax（语法），
  free-form（自由式），417
  nofix（非前后缀），417
  postfix（前缀），416
  prefix（后缀），415, 416, 418
Synthetic camera（合成照相机），299
Systolic Array Graphics Engine (SAGE), 903~904

T

$\bar{T}$ (refraction vector)（折射向量），757
Tablet（输入板），188, 350~351, 353~354, 359, 372
  resistive（电阻的），189
  sonic（声音的），189
  stylus（触笔），188
Tactile continuity（触觉连续性），371
TAE-Plus, 430
Tangent vector（切向量），479~488
Tearing（撕裂），1042
Tektronix, 161
Template（模板），201
Template procedure,geometric modeling（模板过程，几何造型），307
Temporal aliasing（时域走样），见Aliasing, temporal
Tension parameter, β-splines（张力参量，β样条），505~506
Texas Instruments TMS34020（一个显示处理器名），862~864
Texel（纹理元），742。另见Character
  output primitive（输出图元），38~40
Text entry speed（正文输入速度），374
Text-extent（正文范围），129
Texture（纹理），611, 614
  surface（曲面），910
Texture mapping（纹理映射），见Surface detail
Texture space（纹理空间），1015
θ_i (angle between $\bar{L}$ and $\bar{N}$)（向量$\bar{L}$与$\bar{N}$的夹角），723
θ_i (angle of incidence)（入射角），756
θ_t (angle of refraction)（折射角），756
Thick lines（粗线，宽线），见Scan conversion of thick primitives
Thickness（粗度，厚度，宽度），105
ThingLab, 1072
Thinking Machines Connection Machine, 911
Thresholding（取阈值），820
Tiling（贴图），101
Time delay, input（时间延迟，输入），439
Tint, color（色浓，颜色），574
Tint fill（浓淡填充），见Filling algorithms, tint fill
Tone, color（色调，颜色），574
Top-down design,user interface（用户界面的自顶向下设计），404, 429
Torrance-Sparrow（Torrance-Sparrow表面模型），见Illumination, Torrance-Sparrow

Total internal reflection（全内反射）, 758, 778
Touch panel（触摸板）, 193, 350~351, 354~355, 359
Trackball（跟踪球）, 191, 350, 355, 379~380
Tracking system（跟踪系统）, 见Locator
Transformation（变换）, 201~226
  affine（仿射）, 207, 598, 1106
  construction of（的构造）, 1107
  of coordinate system（坐标系的）, 222~226
  geometric（几何）, 16, 877~878
  image（图像）, 182~183
  incremental（增量）, 213
  linear（线性）, 1106
  modeling（模型）, 280, 868
  of normal vectors（法向量的）, 217
  normalizing（规格化）, 259
  in object modeling（实体造型中的）, 291, 304~308, 315, 336
  perspective（透视）, 275
  of points in an affine space（仿射空间中的点变换）, 1107
  rigid body（刚体）, 207
  of vectors in a vector space（向量空间的向量变换）, 1107
  viewing（观察）, 869
  viewing in SPHIGS（SPHIGS中的观察变换）, 299~302
  window-to-viewport（窗口到视口）, 210~212
Transformation composition（变换合成）, 208~210, 217~222
Transformation matrix（变换矩阵）, 210。另见State-transition network
  augmented(ATN)（扩展）, 461~463
  augmented recursive（扩展递归）, 462~463
  recursive(RTN)（递归）, 460~463
Translation（平移）
  3D（三维）, 214
  2D（二维）, 201~202
Transmission coefficien(k_t)（透射系数）, 754, 778
Transparency（透明度, 透明性）, 614, 754~758
  approximation to changingk_t（对改变k_t的逼近）, 756
  critical angle（临界角）, 758
  filtered（滤镜透明法）, 755
  z-buffer implementations（z缓存实现）, 755~756
  interpolated（插值透明法）, 754~755
  nonrefractive（无折射）, 754~756
  refractive（折射）, 757
  screen-door, 755
  of surfaces（曲面）, 909~910
  total internal reflection（整体内部反射）, 758
Transpose（转置）, 见Matrix, transpose of
Traversal（遍历）, 见Display traversal
Trees, modeling of（树）, 1022, 1027~1031
Triangle, signed area of（带符号的三角形面积）, 1112
Triangle Processor and Normal Vector Shader（三角形处理器与法向量, 明暗处理模块）, 900
Triangle strip（三角带）, 见Polygon, mesh
Triangulation（三角剖分）, 143
Trigger（触发器）, 42, 436~439
Trimming curves（修剪曲线）, 528
Tristimulus theory, color（三色激励理论）, 576
Tristimulus values（三色激励值）, 582
Trivariate Bernstein polynomials（Bernstein多项式）, 见Bernstein polynomials, trivariate
Trivial acceptance（简单地接受）, 113, 271
Trivial rejection（简单地拒绝）, 113, 271
Turbulence（湍流）, 1046
Turning vertex（转折顶点）, 932
Tweaking（提拉）, 544, 558, 561
Two-bit graphics（二值图形学）, 853
2-manifolds（二维流形）, 542~543, 547~548
Two-pass transformation（二重变换）, 见Multipass transformation
Typeahead（键入在前）, 438
Typeface（字型）, 见Character, typeface
Typography, digital（数字印刷）, 977

U

Ultimate display（终级显示）, 917
Undercolor removal（消除颜色不足）, 599
Undo command（撤销命令）, 404, 409~410, 460
Uniform nonrational B-spline curves（均匀非有理B样条曲线）, 491~495, 515~516
Uniformly shaped β-spline curves（均匀形状β样条曲线）, 505~506
Update, screen（屏幕更新）, 见Display traversal
User characteristics（用户特性）, 429
User interface（用户界面）, 5, 561
User interface design（用户界面设计）, 348, 391~392, 405, 421, 429, 458
User Interface Design Environment(UIDE)（用户界面设计环境）, 466~467
User interface language（用户界面语言）, 见Language
User Interface Management System(UIMS)（用户界面管理系统）, 348, 456~468
User interface prototype（用户界面原型）, 430
User profile（用户外貌）, 413
User requirements（用户需求）, 429
User-computer dialogue（人机对话）, 380, 392~395
User-computer interaction（人机交互）, 见Interaction
User's model（用户模型）, 见Conceptual design
Utah Raster Toolkit（美国犹他大学的光栅工具箱）,

845, 849

V

$\bar{V}$ (direction to viewpoint)（视点方向），729
Valuator logical device（定值逻辑设备），42, 188, 194, 352, 436
Value discretization（值离散化），817
Vanishing points（灭点），231
Variational calculus（变分法），1052
Variation-diminishing property, curves（变差缩减性质），509
Varifocal mirror（变焦镜），916~917
Vector cross-hatching（矢量交叉阴影），945
Vector display（矢量显示器），见Randam-scan display
Vector generator（向量发生器），9
Vector graphics（向量图形学），9
Vector space（矢量空间），1083~1108
Vectorize（向量化），196~197
Vectors（向量），
  angle between（向量之间的夹角），1096
  normalization of（向量规格化），1096
  projection of（向量的投影），1097
Vertical retrace（垂直回扫），168, 856
VGTS（一个系统名），445
Video controller（视频控制器），11, 164~168, 179 ~184
Video fields（视频域），见Fields, video
Video games（视频游戏），1065
Video look-up table（视频查找表），169~170, 369, 565, 860
  management（管理），447
Video mixing（视频混合），184
Video multiplexer（视频多路传输），860
Video RAM (VRAM)（视频RAM），178
Video random-access memory（视频随机存取存储器），见VRAM
Video signal（视频信号），640
View（视，视图），19
  mapping matrix（映射矩阵），242, 261
  orientation matrix（方向矩阵），242, 261
  plane（平面），237~240
  plane normal (VPN)（平面法线），237
  reference point (VRP)（参考点），237
  up vector (VUP)（上方矢量），238
View volume（视见体），229, 239, 253, 260, 868
  canonical（规范），259, 275~276
Viewing operation（观察操作），86, 258
  SPHIGS, 299~302
Viewing process（观察过程），229~230
Viewport（视口），210, 241~242, 278, 443~445
  SPHIGS，300~302
Virtual buffers（虚拟缓存），902~903
Virtual buffers（虚拟处理器），见Parallel rasterization architectures, hybrid-parallel
  parallel（并行），904~906
Virtual processors, 903
Virtual world（虚拟世界），4, 617。另见Artificial reality, Head-mounted display
Visibility, in geometric model（几何建模中的可见性），332
Visibility determination（可见性判定），见Rasterization
Visible-line determination（可见线判定），611, 649
  Appel's algorithm（Apple的算法），666~667
  Roberts's algorithm（Roberts的算法），665~666
Visible-surface determination（可见面判定），13, 612, 229, 649~720。另见Area-subdivision algorithm, Depth-sort algorithm, List-priority algorithm, Ray tracing algorithm, Scan line algorithm, Subpixel area subdivision algorithm, Warnock's algorithm，Weiler-Atherton algorithm, z-buffer algorithm
  curved surfaces（曲面），698~701
  efficiency（效率），656~665
  formalization（形式化），717~718
  functions of two variables（双变量函数），651~656
  horizon-line algorithm（水平线算法），652
  image precision（图像精度），650
  object precision（对象精度），650
  octree（八叉树），695~698
  sorting（排序），715
  voxels（体素），697
Visual（视觉，视，可视），
  acuity（敏度），569
  balance（平衡），427
  continuity（连续性），368, 408
  elements（元素），425
  organization rules（组织规则），418~422
  proportion（比例），427
Voice recognition（语音识别），352~353, 362, 402
Voice synthesizers（语音合成器），353, 452
Volume rendering（体绘制），914, 1034~1039
Von Koch snowflake（Von Koch雪花），1020
Voxel（体素），549
  visible-surface determination（可见面判定），697
VRAM (video random-access memory)（视频随机存取存储器），859~861, 890, 894, 921

W

Wagon wheel illusion（车轮幻觉），1058
WaitEvent, 438~439
Warn lighting controls（警示灯控制），731~734
Warnock's algorithm（Warnock算法），686~689, 714, 716
Waves（波），1043

Wedges（楔形物），100
Weiler polygon algorithm（Weiler多边形算法），见Clipping, Weiler polygon algorithm
Weiler-Atherton algorithm（Weiler-Atherton算法），689~693, 747
What you see is what you get (WYSIWYG)（所见即所得），348, 396~397, 398, 403
Wheel of reincarnation（轮回），175
Widget（窗口小部件），452~456
Width scans（宽度扫描），972
Winding rule（环绕规则），965
Window（窗口）
  hardware-assisted（硬件辅助），907~908
  hierarchical（分层），445~446
  ID bits（标识位），908
  pick（拾取），见Pick window
  priority（优先级），908
  world coordinate（世界坐标系），210, 229, 237, 443~445
Window events（窗口事件），447~451
Window management systems（窗口管理系统），439~443, 451
Window manager（窗口管理器），1, 176, 439~451, 601
Window system（窗口系统），440
Window-to-viewport mapping（窗口到视口映射），210, 359
Winged-edge data structure（翼边数据结构），545~546
Wireframe（线框），560
  rendering（绘制），323
Workstation（工作站），145, 866, 890~893
World-coordinate system（世界坐标系），60
Write mode（写模式），58

X

X Window System, 16, 440~441, 452, 963, 996
Xerox, 348, 364, 397

Y

YIQ color model（YIQ颜色模型），584, 589~590, 597
Yon clipping plane（Yon剪贴板），见Clipping plane, back
Young Sherlock Holmes（一部电影的名字），1050

Z

z-Buffer algorithm（z缓存算法），13, 668~672, 685, 698, 716, 751~753, 799, 812
  z compression problem（z向压缩问题），718~719
  picking（拾取），720
Zoom（放缩），182, 362, 639