国外计算机科学教材系列

www.Linuxidc.

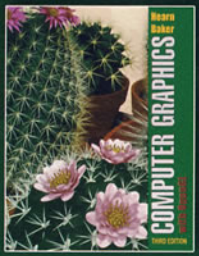
PEARSON Prentice Hall

计算机图形学

（第三版）

Computer Graphics with OpenGL

Third Edition



[美] Donald Hearn
M. Pauline Baker 著

蔡士杰 宋继强 蔡 敏 译

蔡士杰 审校

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

Linux公社（www.LinuxIDC.c

www.Linuxidc.com

译 者 序

交互式计算机图形学的飞速发展令人兴奋，其广泛的应用使科学、艺术、工程、商务、工业、医药、政府、娱乐、广告、教学、培训和家庭等各方面均获得巨大收益。

由 Donald Hearn 和 M. Pauline Baker 合著的《计算机图形学》初版于 1986 年，在 1994 年进行了部分修改，1997年又将其重写，继而推出了第二版。本书第二版的翻译版在国内深受广大读者的欢迎。2004年，两位作者针对计算机图形学领域的最新进展，再次对本书进行了大幅度的修改，从而推出了全新的第三版。在这一版本的相关内容中，除了将二维和三维概念集成在一起讲解之外，作者还对动画、对象表示、三维观察流水线、光照模型、表面绘制技术和纹理映射等许多主题进行了扩展，并且增加了讨论的深度。另外，书中还引入了现在广泛应用的 OpenGL 技术。通过采用 OpenGL 软件包中的图形子程序库，全书提供了 20 多个完整的 C++ 程序示例。

为了适应计算机图形学的发展并促进其应用，几乎所有的高等学校均已开设了计算机图形学课程。人们都希望有更新、更好的计算机图形学教材。在这种情况下，我们把本书介绍给国内读者，希望能对计算机图形学的教学、研究与应用起到积极的作用。

本书由蔡士杰、宋继强和蔡敏共同翻译。蔡士杰翻译了第1章～第6章，宋继强翻译了第7章～第12章，蔡敏翻译了第13章～第15章以及附录。最后，由蔡士杰对全书进行了审校。由于译者的水平有限，书中难免出现错误和不妥之处，敬请读者不吝赐教。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

前　言

计算机图形学是一种最令人兴奋的且发展最快的现代技术。自从本书第一版面市以来，计算机图形已成为一般的应用软件和计算机系统的标准特点。计算机图形方法普遍地应用于许多产品的设计、培训模拟、音乐视频和电视广告产品、动画、数据分析、科学研究以及其他应用之中。各种应用领域所需的许多技术和硬件设备已经投入使用或正在开发之中。特别值得一提的是，当前计算机图形学研究中有许多与改善图片生成的效率和速度有关。为了生成一张自然景象的真实感视图，图形程序必须模拟实际的光照效果和物理对象的折射。因此，当前计算机图形学的趋势是将改进的物理原理的近似结合到图形算法中，从而更好地模拟物体和照明环境之间的复杂交互。

第三版的特色

本书第三版的内容来自于我们多年来开设的各种课程的要点，这些课程包括计算机图形学引论、高级计算机图形学、科学计算可视化、专题和项目课程。在撰写本书第一版时，许多图形学课程和应用仅处理二维方法，因而我们将二维和三维图形技术分开讨论。本书第一版的前一半内容讲述了二维计算机图形处理的基础概念，而在后面讨论三维方法。不过现在三维图形的应用非常普遍，并且许多主要的计算机图形学课程或者主要处理三维方法，或者较早地介绍三维图形；因此，第三版的主要特点是将二维和三维主题集成在一起。

我们也扩充了许多相关主题，包括图形学当前的发展概况和新应用的讨论。本书第三版涵盖的内容有：图形系统当前的硬件和软件组成、分形几何、光线追踪、样条、光照模型、表面绘制、计算机动画、虚拟现实、图形算法的并行实现、反走样、超二次曲面、BSP树、粒子系统、基于物理的建模、科学计算可视化、辐射度、凹凸映射和变形；主要扩展的内容有：动画、对象表示、三维观察流水线、光照模型、表面绘制技术和纹理映射。

第三版中另一有意义的改变是引入现在广泛应用并且多数计算机系统都具备的OpenGL图形子程序。OpenGL软件包为使用通用语言（如C和C++）进行编程以及创建计算机图形显示，提供了巨大的、有效的且设备无关的函数集合。OpenGL的辅助库提供对需要设备交互的输入和输出操作以及其他的图形过程（如生成圆柱形状、球形对象和B样条的处理）。

程序设计示例

第三版中提供了20多个完整的C++程序示例，这些都是采用流行的OpenGL软件包中的图形子程序库编写的。这些程序展示了基本构图技术、二维和三维几何变换、二维和三维观察方法、透视投影、样条生成、分形方法、交互鼠标输入、拾取操作、菜单和子菜单显示以及动画技术的应用。另外，书中还给出了100多个C++/OpenGL程序段，用于展示裁剪、光照效果、表面绘制、纹理映射和许多其他计算机图形方法的算法实现。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

所需背景

我们不要求读者预先熟悉计算机图形学，但要求读者具有程序设计和基本的数据结构方面的一些知识。计算机图形学的相关算法中用到了许多数学方法，这些方法的深入讨论将在附录中给出。附录中的数学内容包含了解析几何、线性代数、向量和张量分析、复数、四元数、微积分初步和数值分析等方面的技术。

第三版可作为没有计算机图形学背景的学生的教材和图形学专业人员的参考书。本书的侧重点是对设计、使用和理解计算机图形系统所需的基本理论，以及用来展示每一主题的方法和应用的大量示例程序。

课程安排的建议

对于一学期的课程，可以选择二维方法或二维和三维结合的方法进行教学，并按课程的特定要求进行选择。对两学期的课程，则在第一学期选择基本图形概念和方法，而在第二学期选择覆盖高级三维方法和算法。对于自学的读者，前面一些章节用来理解图形概念，然后再根据读者的兴趣选择后面的某些章节。

对于本科生来说，第2章～第6章、第11章和第13章的内容可用来组织计算机图形学引论的课程。从这几章中选择相关内容用以讲述二维方法，或者再加上第7章和第10章中的一部分内容来介绍三维主题。其他主题，如分形表示、样条曲线、纹理映射、深度缓存方法或颜色模型，可在第一门计算机图形学课程中介绍。对于低年级研究生或高年级本科生的课程来说，授课内容偏重于三维观察、三维建模、光照模型和表面绘制方法。一般情况下，两学期的授课内容可以较好地覆盖二维和三维计算机图形方法的基础，其中包括样条表示、表面绘制和光线跟踪。对于学生已掌握计算机图形学预备知识的专题课，相应地可以讲解可视化技术、分形几何、样条方法、光线跟踪、辐射度和计算机动画等专题中的一到两种。

第1章通过展示采用图形软件生成的各种不同类型的图片，给出了计算机图形应用领域间的差异。第2章介绍了计算机图形学的基本术语，以及图形系统的硬件和软件的组成，然后详细介绍了OpenGL和一个完整的OpenGL示例程序。第3章和第4章给出了表示和显示简单对象的基本算法。这两章考察了生成多边形和圆等基本图形成分的方法与设定对象颜色、大小和其他属性的技术，以及这些技术在OpenGL中的实现。第5章讨论了旋转和缩放等几何变换。第6章和第7章详细解释了显示二维和三维场景视图的过程。复杂对象如二次曲面、样条、分形和粒子系统等的显示生成方法将在第8章进行讨论。识别三维场景中可见面的各种计算机图形技术将在第9章讨论。光照模型和对场景应用光照条件的方法将在第10章进行介绍。交互图形输入和设计图形用户界面（GUI）的方法在第11章给出。计算机图形学中非常有用的各种颜色模型和方法以及颜色设计的考虑在第12章讨论。动画技术则在第13章进行介绍。复杂系统的层次式建模方法将在第14章给出。在第15章，我们综述了主要的图形文件格式。

致谢

多年以来，许多人以各种方式对本书做出了贡献。对提供图片和其他材料的组织与个人，我们再次表示感谢。我们也感谢一些学生在计算机图形学及可视化课程和讲座中提出的许多有帮助的意

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

见。我们感谢所有审阅本书或对本书提出改进意见的人，同时我们对未能致谢的人表示歉意。谢谢Ed Angel、Norman Badler、Phillip Barry、Brian Barsky、Hedley Bond、Bart Braden、Lara Burton、Robert Burton、Greg Chwelos、John Cross、Steve Cunningham、John DeCatrel、Victor Duvaneko、Gary Eerkes、Parris Egbert、Tony Faustini、Thomas Foley、Thomas Frank、Don Gillies、Andrew Glassner、Jack Goldfeather、Georges Grinstein、Eric Haines、Robert Herbst、Larry Hodges、Carol Hubbard、Eng-Kiat Koh、Mike Krogh、Michael Laszlo、Suzanne Lea、Michael May、Nelson Max、David McAllister、Jeffrey McConnell、Gary McDonald、C. L. Morgan、Greg Nielson、James Oliver、Lee-Hian Quek、Laurence Rainville、Paul Ross、David Salomon、Günther Schrack、Steven Shafer、Cliff Shaffer、Pete Shirley、Carol Smith、Stephanie Smullen、Jeff Spears 、William Taffe、Wai Wan Tsang、Spencer Thomas、Sam Uselton、David Wen、Bill Wicker、Andrew Woo、Angelo Yfantis、Marek Zaremba、Michael Zyda和许多不知名的审稿者。我们也感谢编辑Alan Apt、Toni Holm以及Colorado工作人员的帮助、建议与鼓励。还有Lynda Castillo、Camille Trentacoste、Heather Scott、Xiaohong Zhu、Vince O'Brien、Patricia Burns、Kathy Ewing和David Abel，我们向他们的贡献和认真细致的工作再一次表示感谢。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

目　　录

第 1 章　计算机图形学综述 1
1.1　图和表 1
1.2　计算机辅助设计 3
1.3　虚拟现实环境 8
1.4　数据可视化 10
1.5　教学与培训 15
1.6　计算机艺术 17
1.7　娱乐 21
1.8　图像处理 23
1.9　图形用户界面 24
1.10　小结 25
参考文献 25

第 2 章　图形系统概述 26
2.1　视频显示设备 26
2.2　光栅扫描系统 39
2.3　图形工作站和观察系统 42
2.4　输入设备 44
2.5　硬拷贝设备 51
2.6　图形网络 53
2.7　因特网上的图形 53
2.8　图形软件 53
2.9　OpenGL 简介 56
2.10　小结 62
参考文献 63
练习题 63

第 3 章　输出图元 65
3.1　坐标系统 65
3.2　OpenGL 中指定二维世界坐标系统 66
3.3　OpenGL 画点函数 67
3.4　OpenGL 画线函数 69
3.5　画线算法 70
3.6　并行画线算法 77

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3.7 帧缓存值的装载 79
3.8 OpenGL 曲线函数 80
3.9 圆生成算法 80
3.10 椭圆生成算法 86
3.11 其他曲线 94
3.12 并行曲线算法 97
3.13 像素编址和对象的几何要素 97
3.14 填充区图元 100
3.15 多边形填充区 100
3.16 OpenGL 多边形填充区函数 109
3.17 OpenGL 顶点数组 114
3.18 像素阵列图元 117
3.19 OpenGL 像素阵列函数 117
3.20 字符图元 120
3.21 OpenGL 字符函数 122
3.22 图形分割 123
3.23 OpenGL 显示列表 123
3.24 OpenGL 显示窗口重定形函数 125
3.25 小结 128
示例程序 130
参考文献 138
练习题 138

第 4 章 图元的属性 141
4.1 OpenGL 状态变量 141
4.2 颜色和灰度 141
4.3 OpenGL 颜色函数 143
4.4 点的属性 147
4.5 线的属性 148
4.6 曲线属性 151
4.7 OpenGL 点属性函数 153
4.8 OpenGL 线属性函数 154
4.9 填充区属性 156
4.10 通用扫描线填充算法 159
4.11 凸多边形的扫描线填充 163
4.12 曲线边界区域的扫描线填充 163
4.13 不规则边界区域的填充方法 163
4.14 OpenGL 填充区属性函数 167
4.15 字符属性 171
4.16 OpenGL 字符属性函数 175

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

4.17 反走样 175
4.18 OpenGL 反走样函数 182
4.19 OpenGL 查询函数 182
4.20 OpenGL 属性组 183
4.21 小结 183
参考文献 185
练习题 185

第 5 章 几何变换 188
5.1 基本的二维几何变换 188
5.2 矩阵表示和齐次坐标 194
5.3 逆变换 196
5.4 二维复合变换 196
5.5 其他二维变换 207
5.6 几何变换的光栅方法 212
5.7 OpenGL 光栅变换 213
5.8 二维坐标系间的变换 214
5.9 三维空间的几何变换 216
5.10 三维平移 216
5.11 三维旋转 218
5.12 三维缩放 229
5.13 三维复合变换 231
5.14 其他三维变换 234
5.15 三维坐标系间的变换 235
5.16 仿射变换 236
5.17 OpenGL 几何变换函数 236
5.18 小结 242
参考文献 243
练习题 243

第 6 章 二维观察 246
6.1 二维观察流水线 246
6.2 裁剪窗口 247
6.3 规范化和视口变换 249
6.4 OpenGL 二维观察函数 253
6.5 裁剪算法 259
6.6 二维点裁剪 260
6.7 二维线裁剪 260
6.8 多边形填充区裁剪 271
6.9 曲线的裁剪 279

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

6.10 文字的裁剪 280
6.11 小结 281
参考文献 283
练习题 283

第7章 三维观察 285
7.1 三维观察概念综述 285
7.2 三维观察流水线 288
7.3 三维观察坐标系参数 290
7.4 世界坐标系到观察坐标系的变换 294
7.5 投影变换 294
7.6 正投影 295
7.7 斜投影 299
7.8 透视投影 304
7.9 视口变换和三维屏幕坐标系 315
7.10 OpenGL 三维观察函数 316
7.11 三维裁剪算法 321
7.12 OpenGL 任选裁剪平面 327
7.13 小结 328
参考文献 328
练习题 329

第8章 三维对象的表示 330
8.1 多面体 330
8.2 OpenGL 多面体函数 330
8.3 曲面 333
8.4 二次曲面 334
8.5 超二次曲面 336
8.6 OpenGL 二次曲面和三次曲面函数 337
8.7 柔性对象 342
8.8 样条表示 343
8.9 三次样条插值方法 348
8.10 Bézier 样条曲线 355
8.11 Bézier 曲面 362
8.12 B 样条曲线 364
8.13 B 样条曲面 372
8.14 beta 样条 372
8.15 有理样条 373
8.16 样条表示之间的转换 375
8.17 样条曲线和曲面的显示 377
8.18 OpenGL 的逼近样条函数 380

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.19 扫描表示法 389
8.20 结构实体几何法 390
8.21 八叉树 392
8.22 BSP 树 394
8.23 分形几何方法 394
8.24 形状语法和其他过程方法 417
8.25 粒子系统 418
8.26 基于物理的方法 419
8.27 数据集可视化 421
8.28 小结 426
参考文献 428
练习题 429

第 9 章 可见面判别算法 431
9.1 可见面判别算法的分类 431
9.2 后向面判别 431
9.3 深度缓存算法 432
9.4 A 缓存算法 435
9.5 扫描线算法 436
9.6 深度排序算法 437
9.7 BSP 树算法 440
9.8 区域细分算法 440
9.9 八叉树算法 442
9.10 光线投射算法 443
9.11 可见性检测算法的比较 444
9.12 曲面 444
9.13 线框可见性算法 446
9.14 OpenGL 可见性检查函数 447
9.15 小结 449
参考文献 450
练习题 450

第 10 章 光照模型与面绘制算法 452
10.1 光源 452
10.2 表面光照效果 456
10.3 基本光照模型 457
10.4 透明表面 466
10.5 雾气效果 469
10.6 阴影 469
10.7 照相机参数 470
10.8 光强度显示 470

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.9 半色调模式和抖动技术 473
10.10 多边形绘制算法 478
10.11 光线跟踪方法 482
10.12 辐射度光照模型 496
10.13 环境映射 501
10.14 光子映射 502
10.15 添加表面细节 502
10.16 用多边形模拟表面细节 504
10.17 纹理映射 504
10.18 凹凸映射 508
10.19 帧映射 509
10.20 OpenGL 光照和表面绘制函数 510
10.21 OpenGL 纹理函数 517
10.22 小结 527
参考文献 529
练习题 530

第 11 章 图形用户界面和交互输入方法 532
11.1 图形数据的输入 532
11.2 输入设备的逻辑分类 532
11.3 图形数据的输入功能 536
11.4 交互式构图技术 537
11.5 虚拟现实环境 540
11.6 OpenGL 支持交互式输入设备的函数 540
11.7 OpenGL 的菜单功能 555
11.8 图形用户界面的设计 561
11.9 小结 563
参考文献 564
练习题 565

第 12 章 颜色模型和颜色应用 567
12.1 光的特性 567
12.2 颜色模型 569
12.3 标准基色和色度图 570
12.4 RGB 颜色模型 573
12.5 YIQ 颜色模型 574
12.6 CMY 和 CMYK 颜色模型 575
12.7 HSV 颜色模型 576
12.8 HLS 颜色模型 580
12.9 颜色选择及其应用 581
12.10 小结 581

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

参考文献 582
练习题 582

第 13 章　计算机动画 583
13.1　计算机动画的光栅方法 583
13.2　动画序列的设计 584
13.3　传统动画技术 585
13.4　通用计算机动画功能 586
13.5　计算机动画语言 586
13.6　关键帧系统 587
13.7　运动的描述 592
13.8　关节链形体动画 593
13.9　周期性运动 595
13.10 OpenGL 动画子程序 595
13.11 小结 598
参考文献 599
练习题 599

第 14 章　层次建模 601
14.1　基本建模概念 601
14.2　建模软件包 603
14.3　通用层次建模方法 604
14.4　使用 OpenGL 显示列表的层次建模 607
14.5　小结 607
参考文献 607
练习题 607

第 15 章　图形文件格式 609
15.1　图像文件结构 609
15.2　降色方法 610
15.3　文件压缩技术 611
15.4　主流文件格式的结构 616
15.5　小结 620
参考文献 621
练习题 621

附录 A　计算机图形学的数学基础 623

参考文献 656

索引 668

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第1章　计算机图形学综述

计算机已经成为快速、经济地生成图片的强大工具。实际上已经没有哪个领域不能从使用图形显示中获益，因此也就不奇怪为什么计算机图形学的应用是那么广泛。虽然在早期的工程和科学上的应用必须依赖于昂贵而笨重的设备，但是计算机技术的发展已经将交互式计算机图形学变成了一种实用工具。现在，我们可以看到计算机图形学已经频繁地应用于多种领域，如科学、艺术、工程、商务、工业、医药、政府、娱乐、广告、教学、培训和家庭等各方面的应用。我们还可以通过因特网将图像传播到世界各地。图1.1概括地给出了计算机图形学在仿真、培训和数据绘图等方面的应用。在开始深入了解计算机图形学如何工作以前，让我们先简要地了解一下图形学的应用。

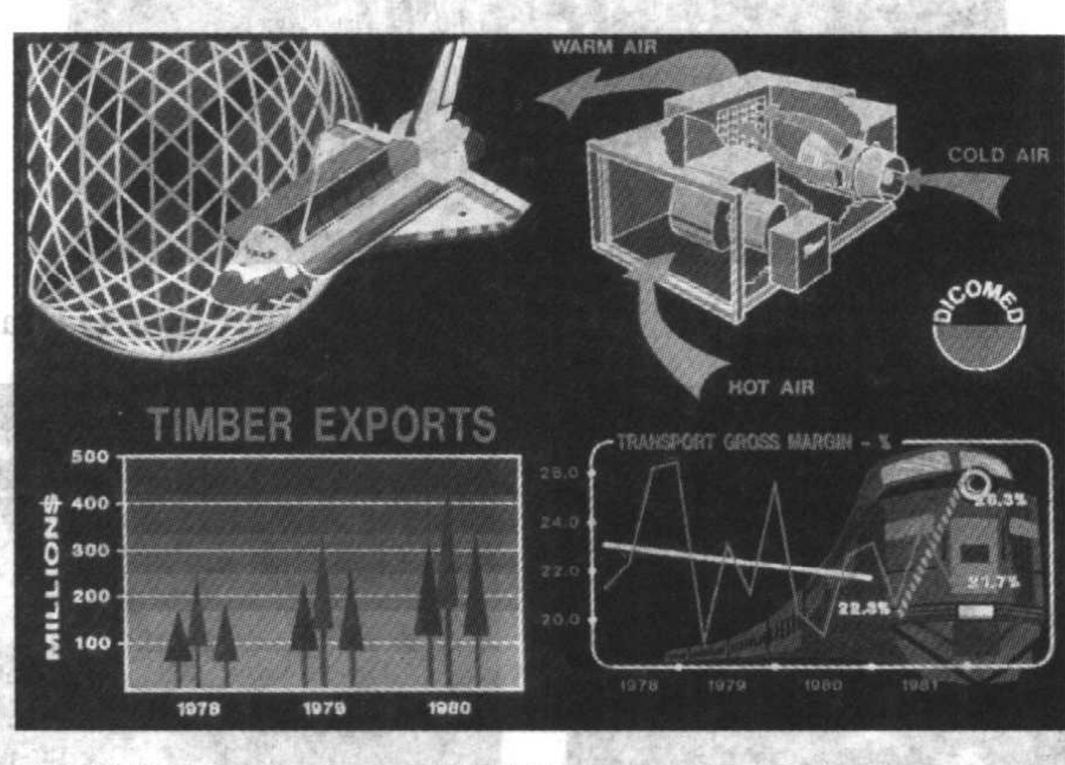


图 1.1　计算机图形学在多个领域的应用示例（DICOMED 公司提供）

1.1　图和表

计算机图形学的一个早期应用是显示简单的数据图，通常在字符打印机上进行绘制。数据绘图现在仍然是最普遍的图形应用之一，但是如今可以很容易地为打印报告或使用 35 mm 幻灯片、透明胶片和动感视频的演示而生成能展现高度复杂数据关系的图片。在研究报告、管理总结、消费信息公报和其他类型的出版物中，常常使用图和表来总结财政、统计、数学、科学、工程和经济数据。现在有各种商业图示软件包、工作站设备和服务部门，专门用来将屏幕显示转换成用于演示和存档的电影、幻灯片或投影用的透明胶片。典型的数据绘图有折线图、直方图、饼图、曲面图、等高线图，以及其他给出二维、三维或多维空间中多个参数之间关系的显示图。

图1.1和图1.2给出了二维数据绘图的例子。这两张图给出了基本的折线图、直方图和饼图。通过将饼图中的几部分向发散方向移出一点来生成“破碎”的饼图，可以引起人们对这些内容的注意。

三维的图和表用来增加显示的信息量，有时仅仅是为了改善效果，表达出引入注目的数据的相互关系。图 1.3 给出了一个与地理信息相结合的直方图例子。图 1.4 给出了颇具戏剧效果的三维数据图例子，图 1.5 中是另一个三维图例子，它以曲面方式绘制。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

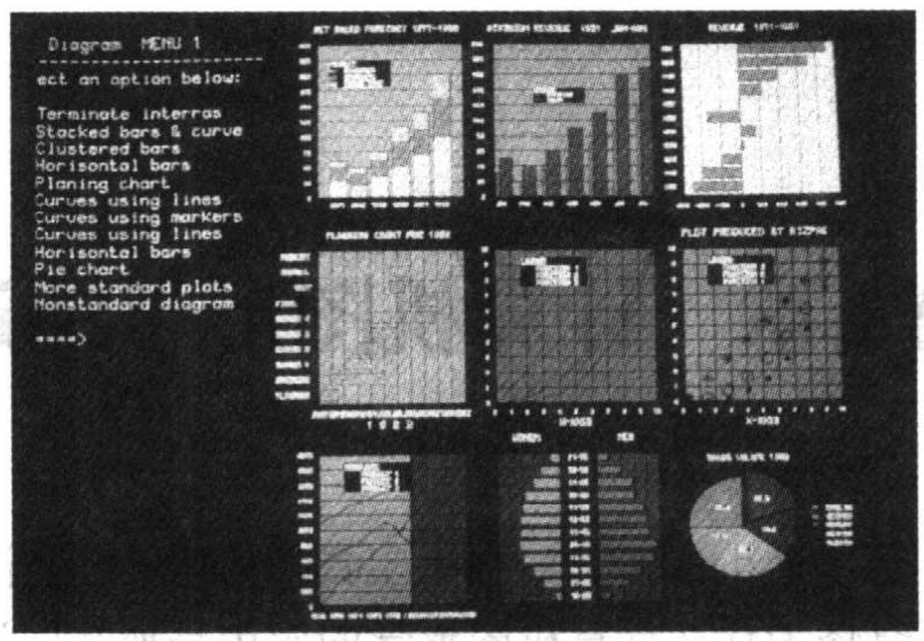


图 1.2　二维折线图、直方图和饼图（UNIRAS 公司提供）

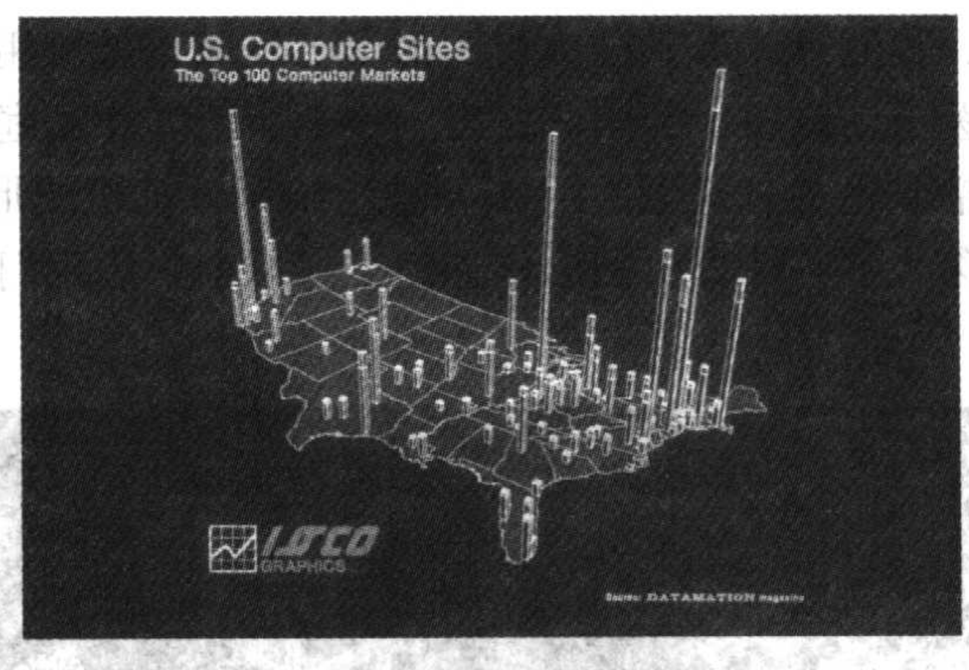


图 1.3　在地理分区图上以三维直方图方式显示两色数据组（ISSCO Graphics 公司提供）

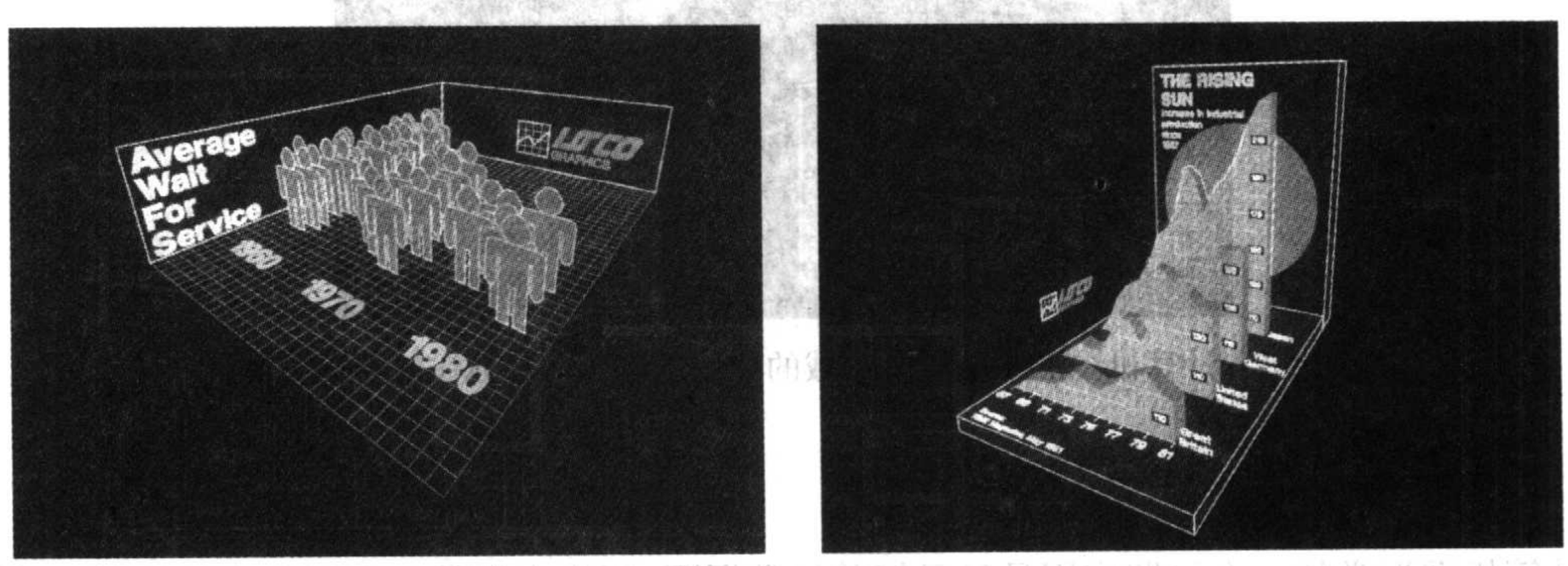


图 1.4　两个颇具戏剧效果的三维图（ISSCO Graphics 公司提供）

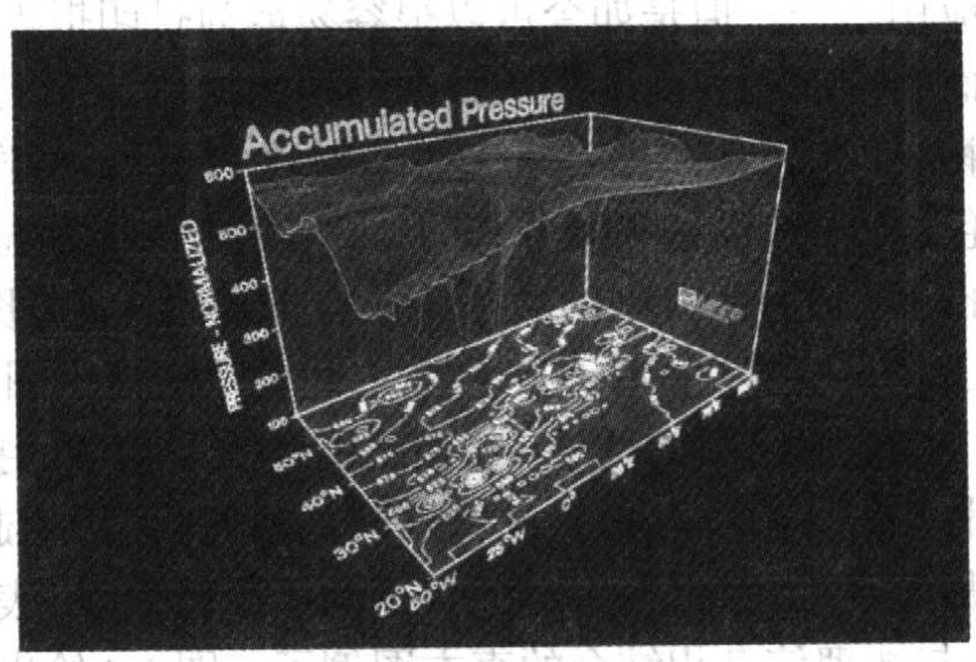


图 1.5　地平面上绘制二维等高线图，上面用曲面形式绘制高度场（ISSCO Graphics 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图1.6给出了用于任务安排的时间图。在项目管理中，经常使用时间图和任务网络分布来制订日程表及管理项目进展。

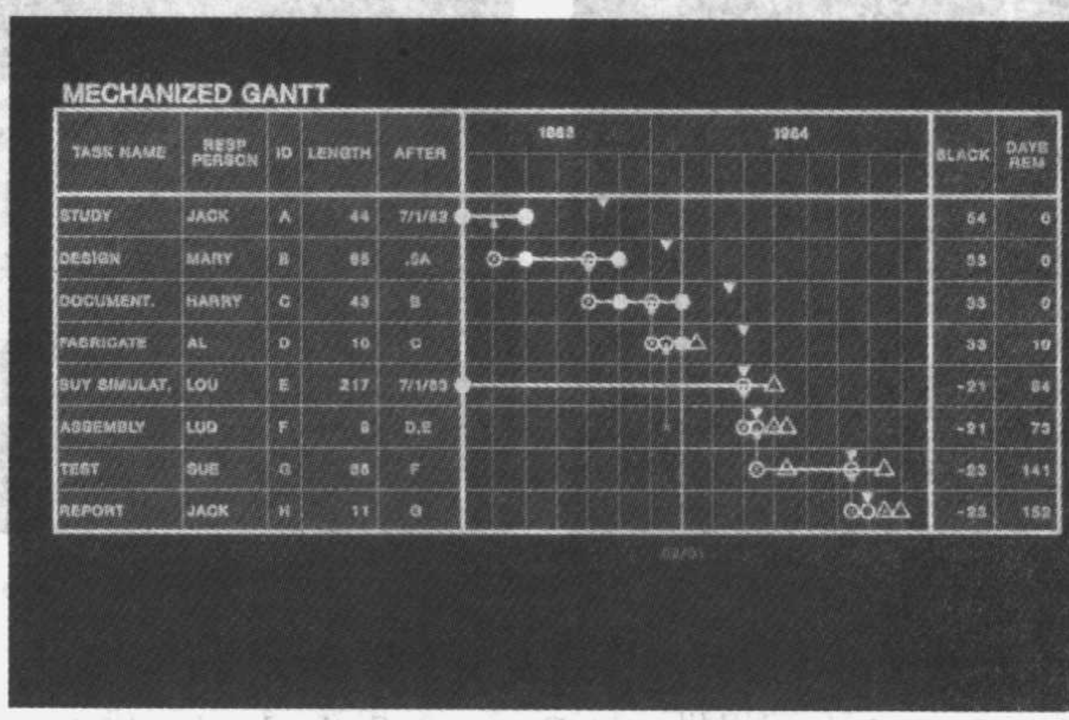


图 1.6　显示规划及其他与项目任务相关的信息的时间表（ISSCO Graphics 公司提供）

1.2　计算机辅助设计

尽管现在几乎所有的产品都已经使用计算机进行设计，但是计算机图形学的主要应用还是在设计过程中，尤其是在工程和建筑系统中。简称为 CAD 的计算机辅助设计（computer-aided design）或称为 CADD 的计算机辅助绘图和设计（computer-aided drafting and design）方法，现在已频繁地应用于大楼、汽车、飞机、轮船、宇宙飞船、计算机、纺织品、家庭用品和许多其他产品的设计中。

在某些设计中，对象首先是以线框轮廓的形式显示出来，从而展现其整个外形以及该对象的内部特征。显示线框图可让设计者很快地看到对设计的外形进行调整的结果，而不用等待对象表面全部生成。图 1.7 和图 1.8 给出了在设计应用方面的线框图例子。

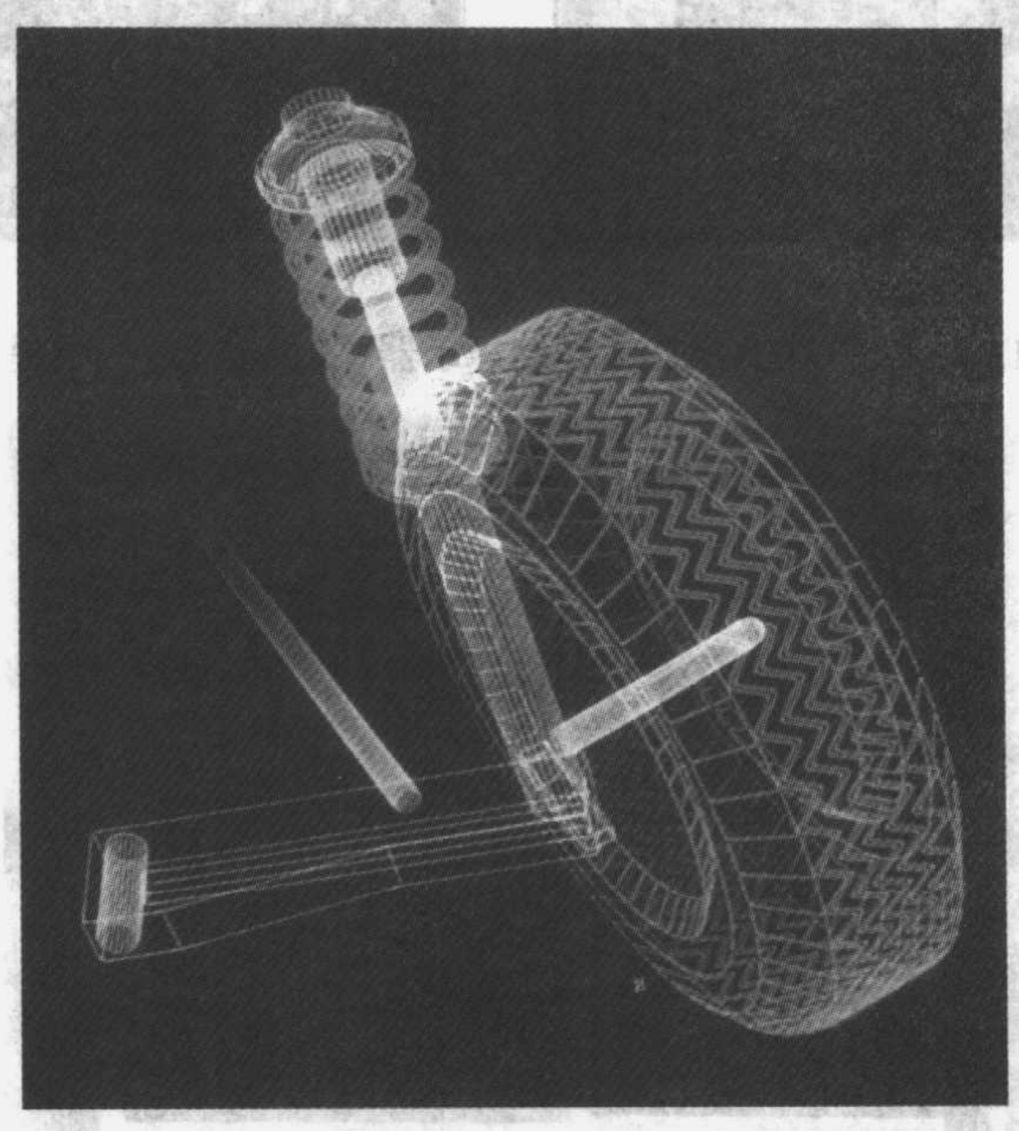
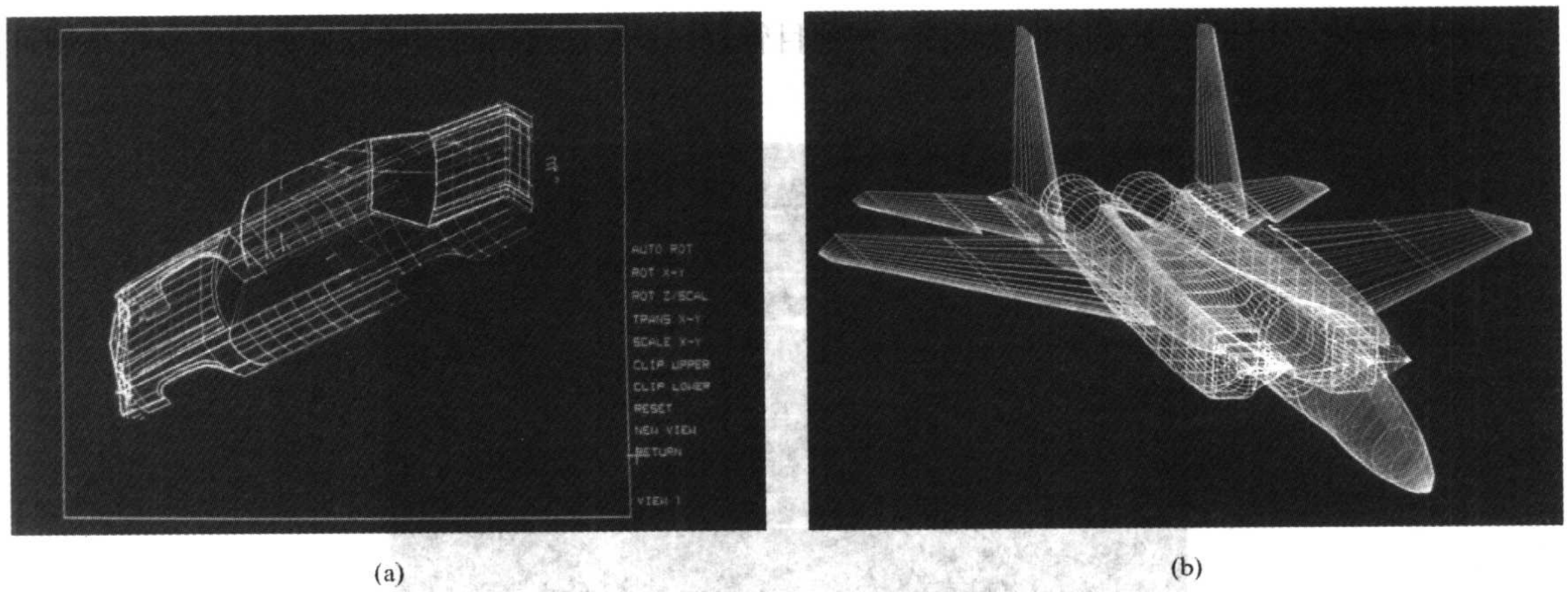


图 1.7　汽车轮胎安装的彩色线框图显示（Evans & Sutherland 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

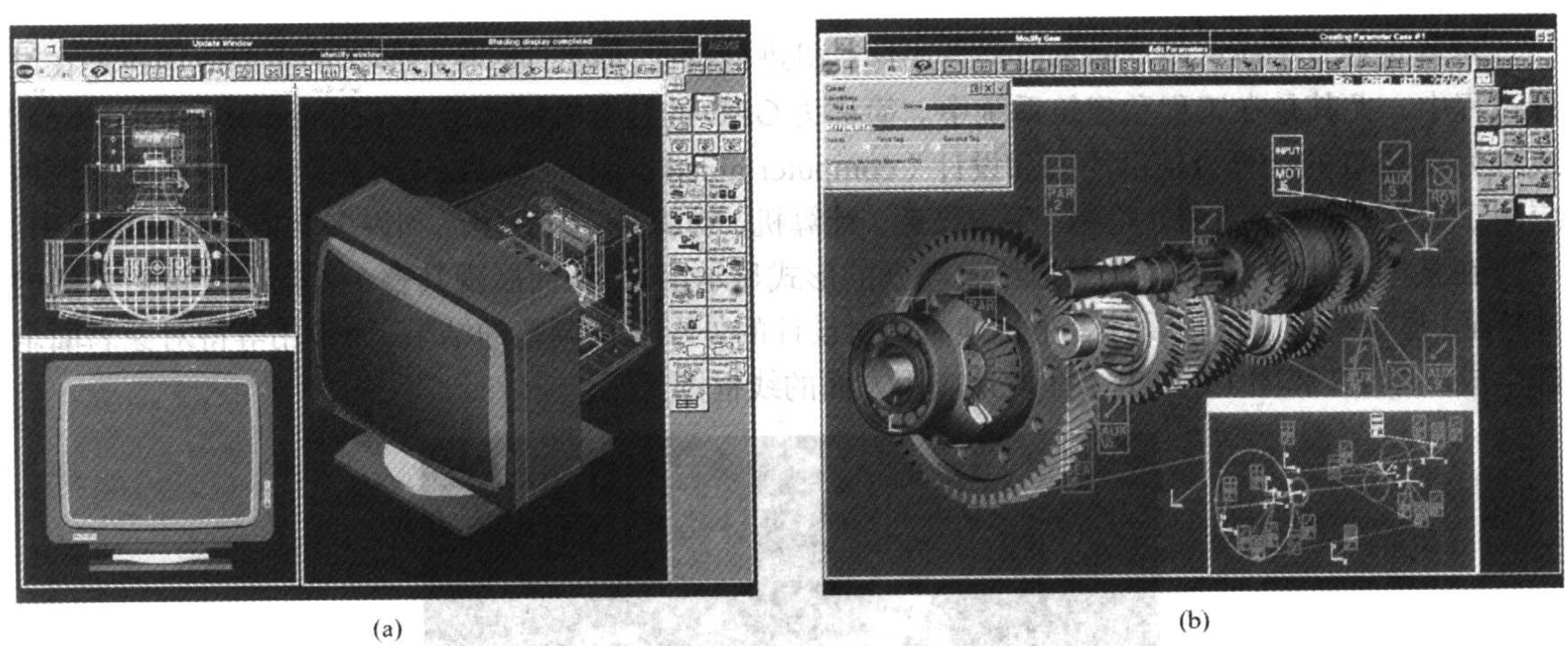
www.Linuxidc.com



(a) (b)

图 1.8 汽车和宇宙飞船外形设计使用的彩色线框图显示
（(a)Peritek 公司提供；(b) Evans & Sutherland 提供）

CAD 应用软件包通常为设计者提供如图 1.9 和图 1.10 所示的多窗口环境。不同的显示窗口展示放大的部分或对象的不同视图。



(a) (b)

图 1.9 多窗口的 CAD 工作站显示器（Intergraph 公司提供）

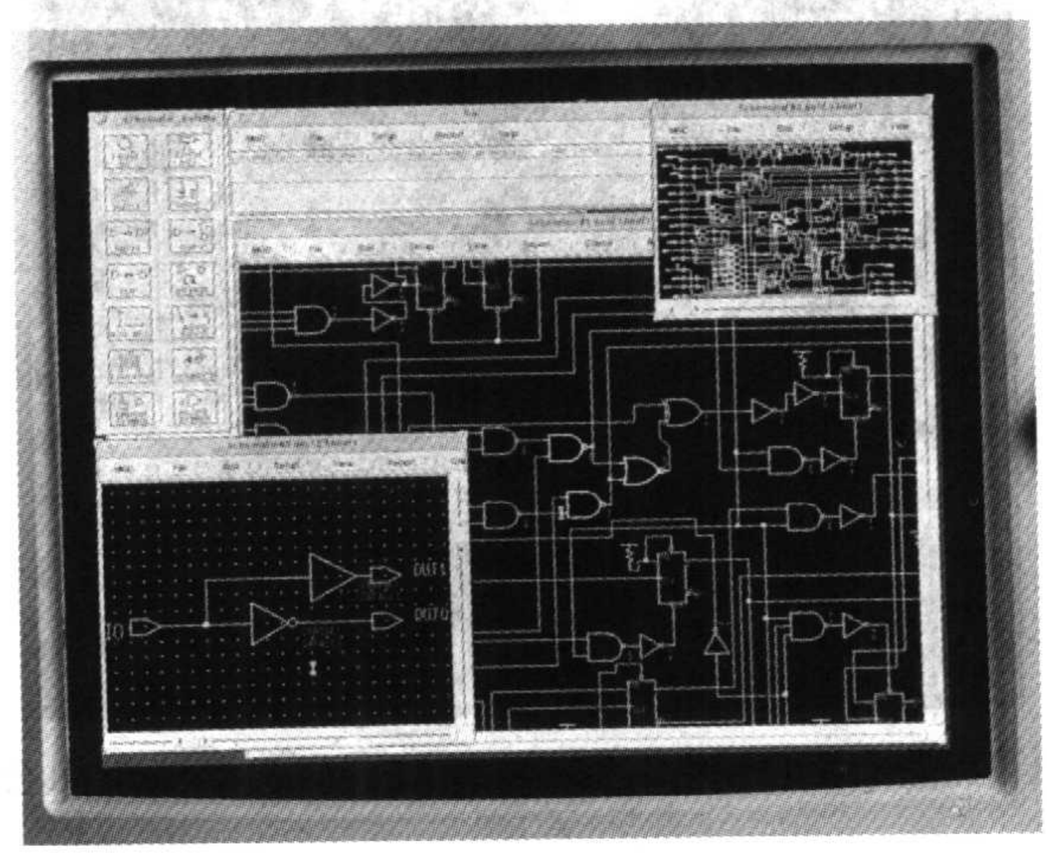


图 1.10 使用多窗口和彩色逻辑部件的电路设计应用（Sun Microsystems 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如图 1.10 所示的电路以及通信网络、供水系统等设施都通过少量图型的反复布局来构建。在设计中使用的图型代表不同的网络或电路部件。而用于机械、电气、电子及逻辑电路的标准图形，则通常由设计软件包提供。在其他一些应用中，设计者可以设计个性化的符号来构建网络或电路。通过依次在布局图中安放部件，并由图形软件包自动提供部件之间的联系来完成对该系统的设计。这使得设计者能够快速测试不同的电路设计方案，从而减少系统需要使用的部件数量或空间。

CAD 应用中还经常使用动画。如图 1.11 所示，在视频监视器上显示线框式的实时动画，这对于测试汽车及系统的性能是很有用的。因为线框图的显示并不需要绘制表面，其每帧的计算可快速完成，从而使屏幕上的运动平稳。线框图显示还可让设计者观察飞行器的内部结构以及在运动时内部构件的变化情况。

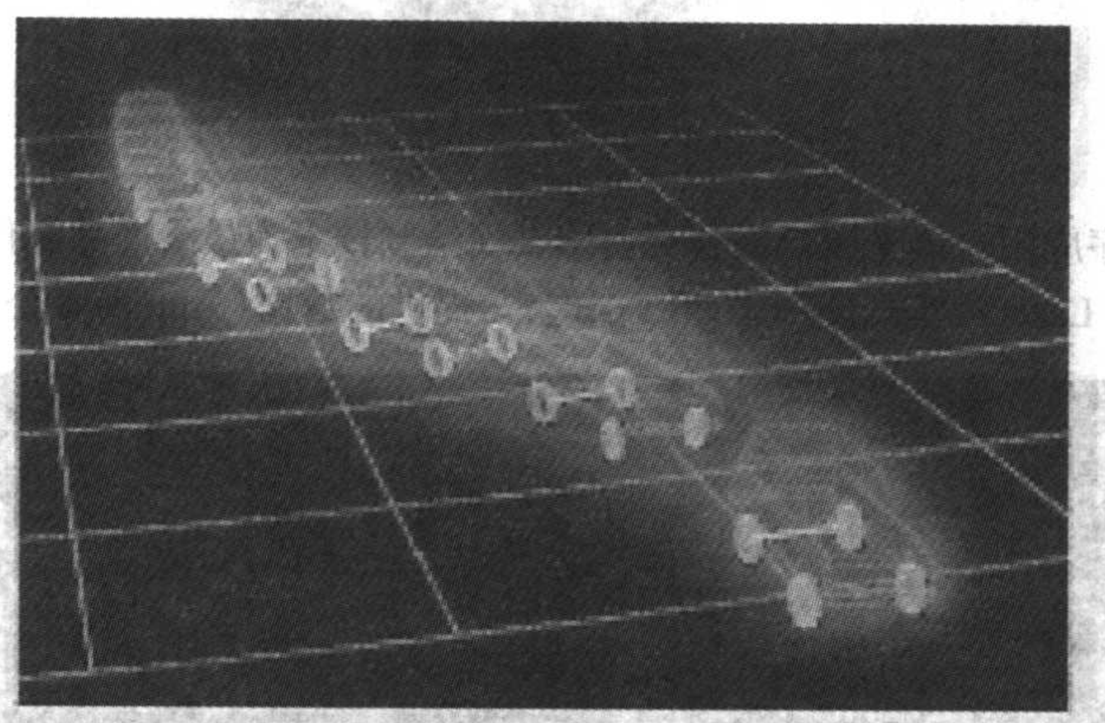
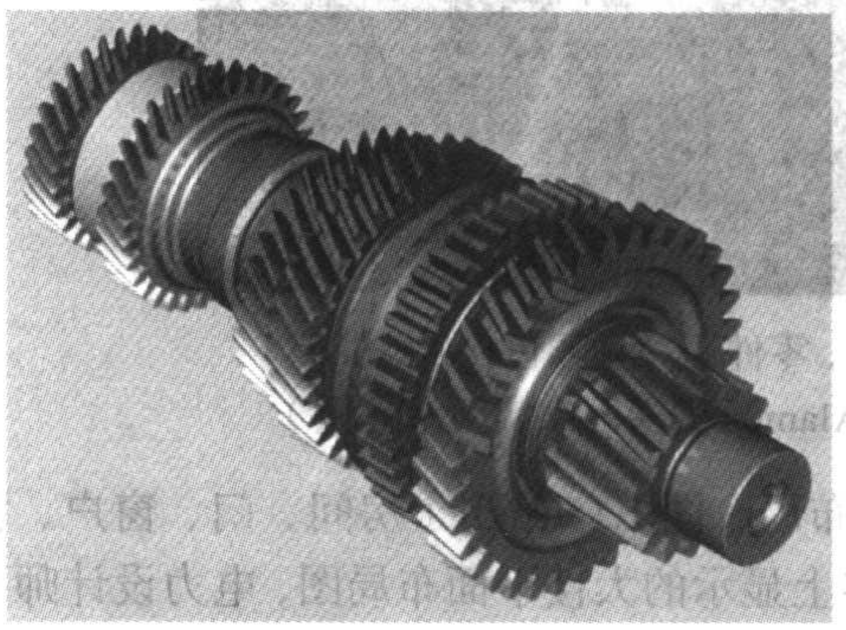
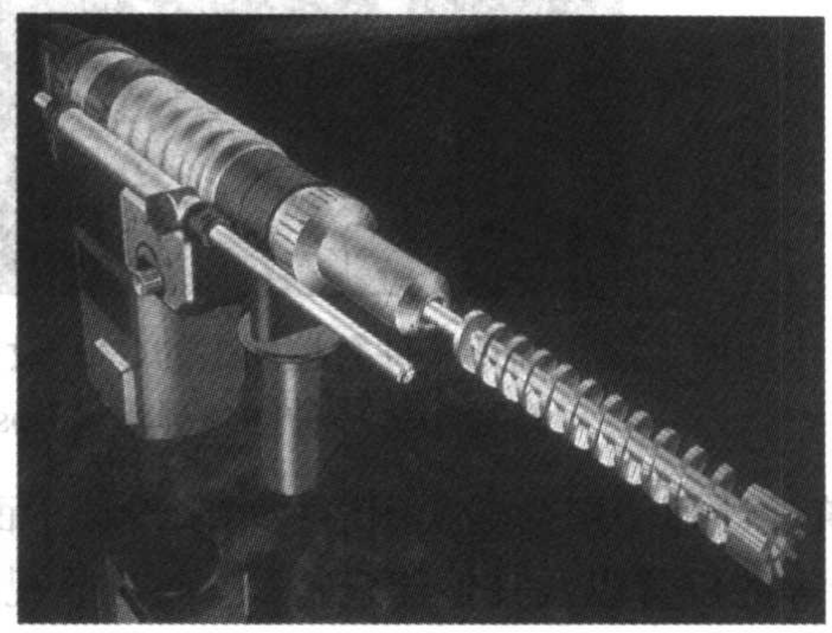


图 1.11　在路线改变时汽车性能的模拟（Evans & Sutherland 和 Mechanical Dynamics 公司提供）

在对象设计完成或将要完成的时候，应用逼真的光照模型和曲面绘制技术可以生成最终产品的外形图，图1.12给出了这样的例子。在汽车和其他交通工具的广告中也使用特殊的光照效果及背景场景来产生真实感效果的显示（参见图 1.13）。



(a)



(b)

图 1.12　工程设计的真实感绘制（(a)Intergraph 公司提供；(b)Evans & Sutherland 提供）

制造过程也和设计对象的计算机描述联系起来，因此使用**计算机辅助制造技术**（computer-aided manufacturing，CAM），可以实现产品安装的自动化。例如，电路板布局可转换成构造该电路图的各个过程的描述。某些机械零件按照指定如何使用机床加工表面的描述来制造。图1.14给出了机床加工一个零件时在表面上的路径，而数控机床则按该指定路径来加工零件。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图 1.13 计算机图形程序应用立体光照效果和真实感表面绘制技术为最终产品制作广告片段，由计算机生成的Chrysler Laser汽车图使用了Chrysler公司提供的数据（Autodesk公司的Eric Haines提供）

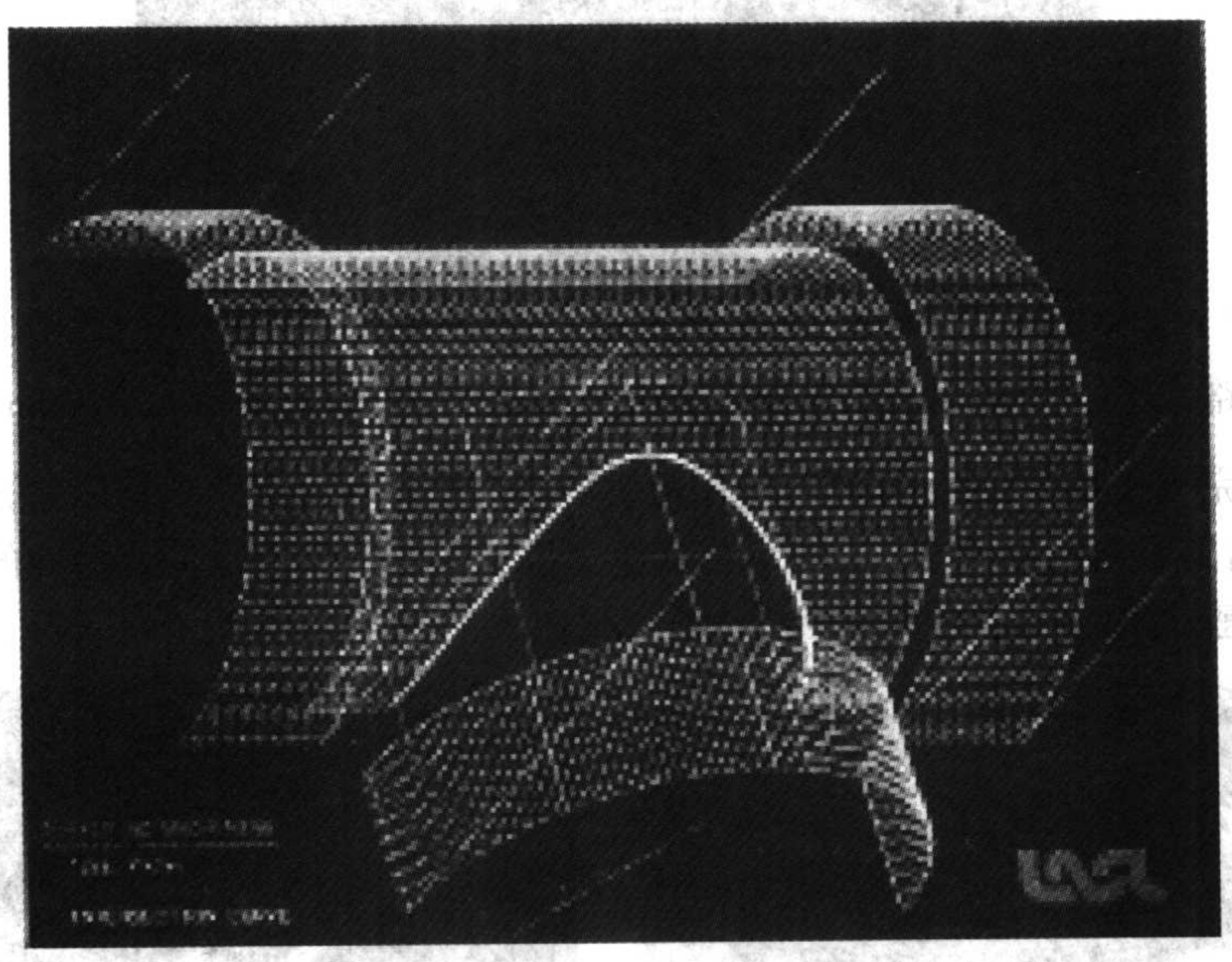


图 1.14 描述零件数控加工的CAD设计，零件用一种颜色显示，机床路径用另一种颜色显示（Los Alamos 美国国家实验室提供）

建筑设计师使用交互式图形技术来设计平面布局，图 1.15 给出了房间、门、窗户、楼梯、架子、柜台等的位置以及其他建筑特征。面对监视器上显示的大楼平面布局图，电力设计师可以进行布线设计及电源插座和防火警报系统的布局。同样也可以使用设备布局软件来优化办公室或车间的空间使用。

图 1.16 给出了建筑设计的逼真显示，这样可以让建筑设计师和他们的客户一起研究学校或工业区的一座或一群大楼的外貌。除了大楼外貌的真实感显示以外，建筑CAD软件包还可提供三维室内布局和光照的功能（参见图 1.17）。

还有许多其他类型的系统和产品也使用了通用CAD软件包或专门开发的CAD软件进行设计。例如，图 1.18 给出了利用CAD系统设计的地毯图案。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

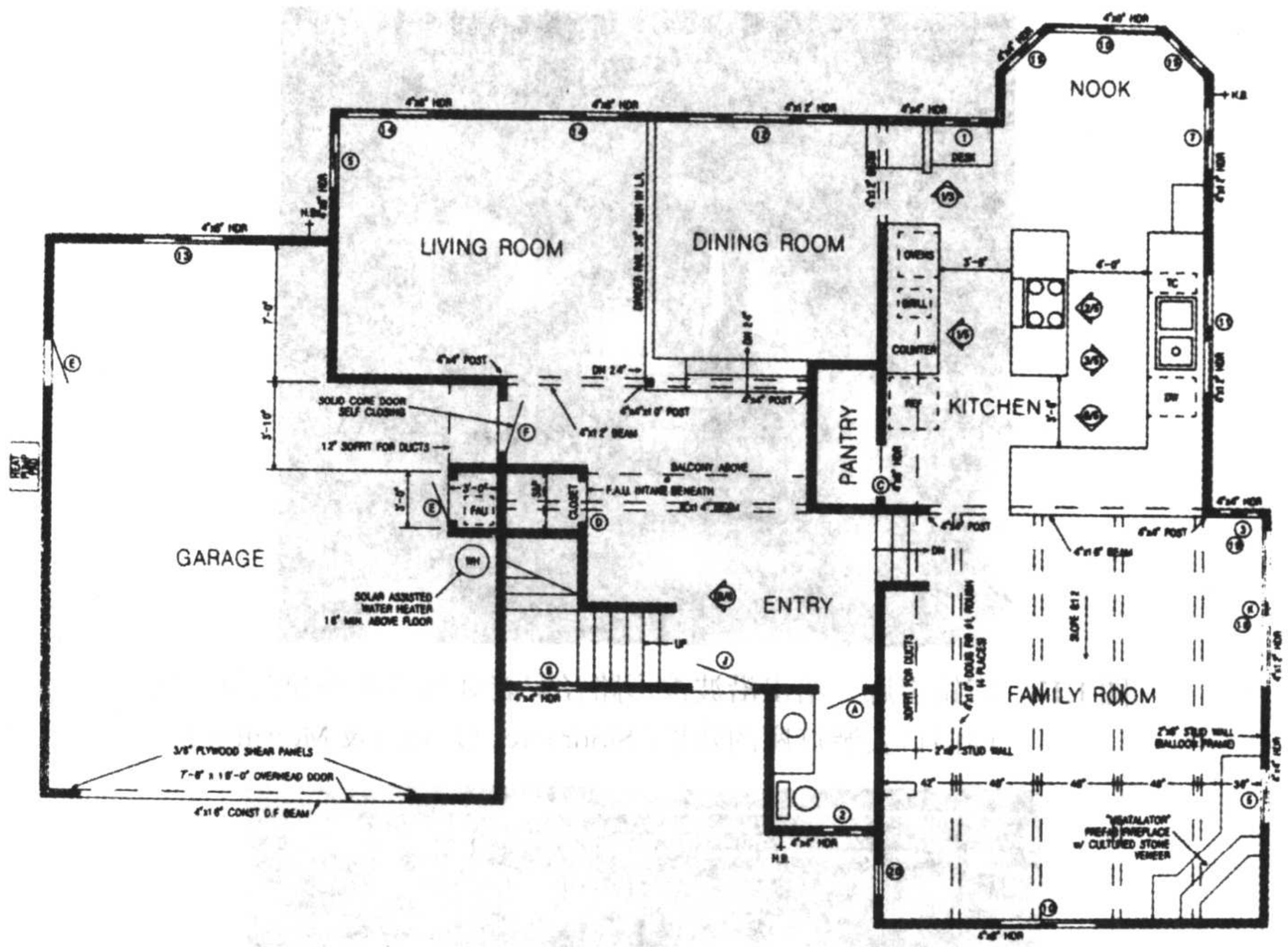
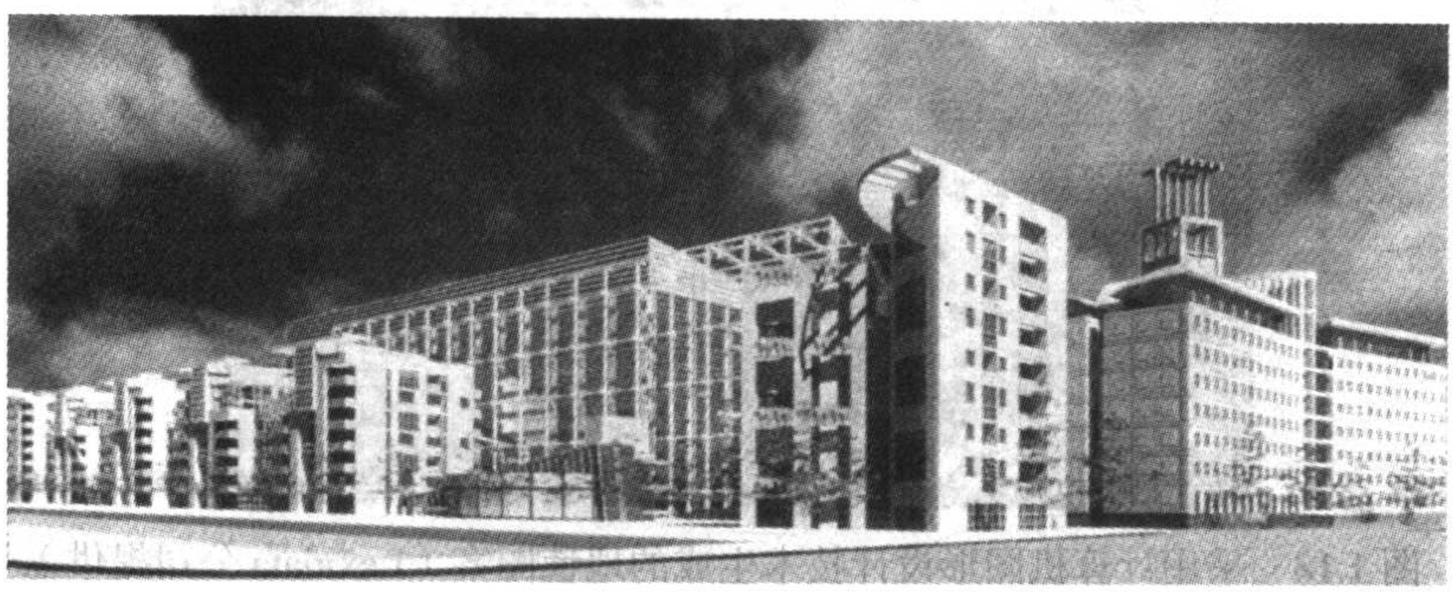
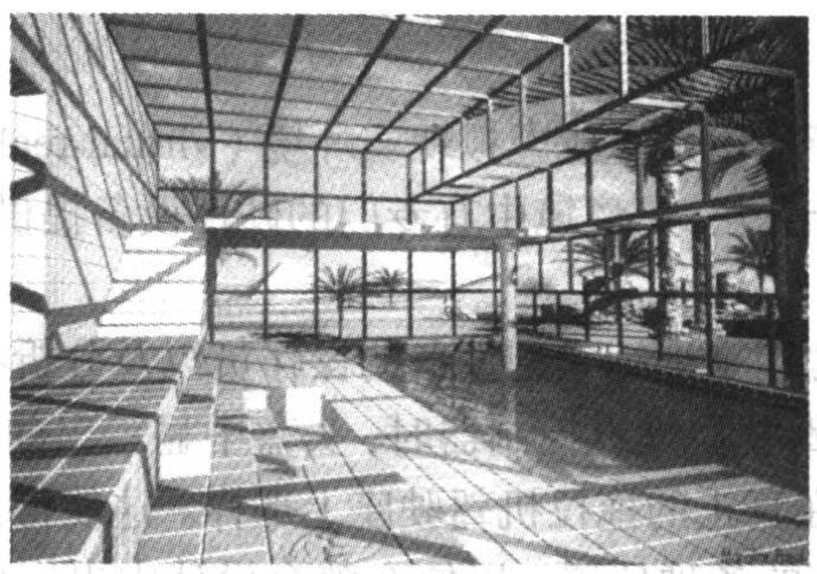


图 1.15　大楼的建筑 CAD 平面布局（Precision Visuals 公司提供）



(a)



(b)

图 1.16　大楼设计的三维真实感绘制：(a)世贸中心街头透视图（Skidmore, Owings & Merrill 提供）；(b)门廊建筑可视化，是为法国里昂 Marialine Prieur 的计算机动画而制作的（Thomson Digital Image 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

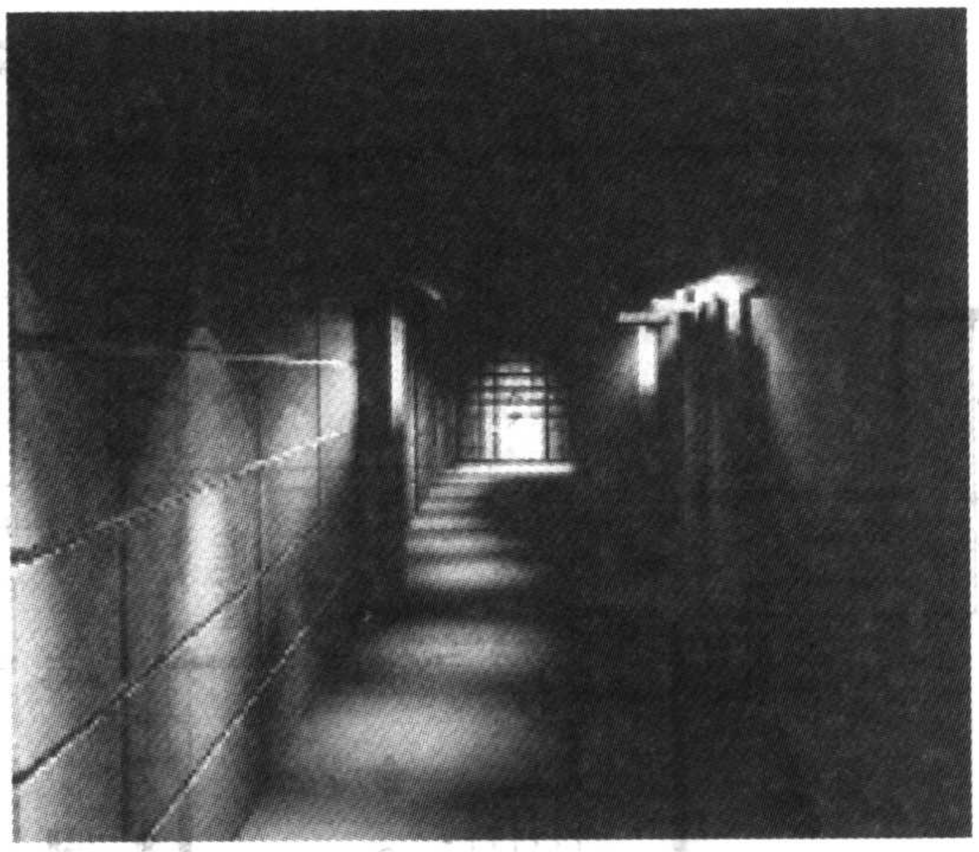


图 1.17 宾馆走廊，给出沿波浪形路径安装灯具的效果和在每一房间门口安装灯座的效果（Skidmore, Owings & Merrill提供）

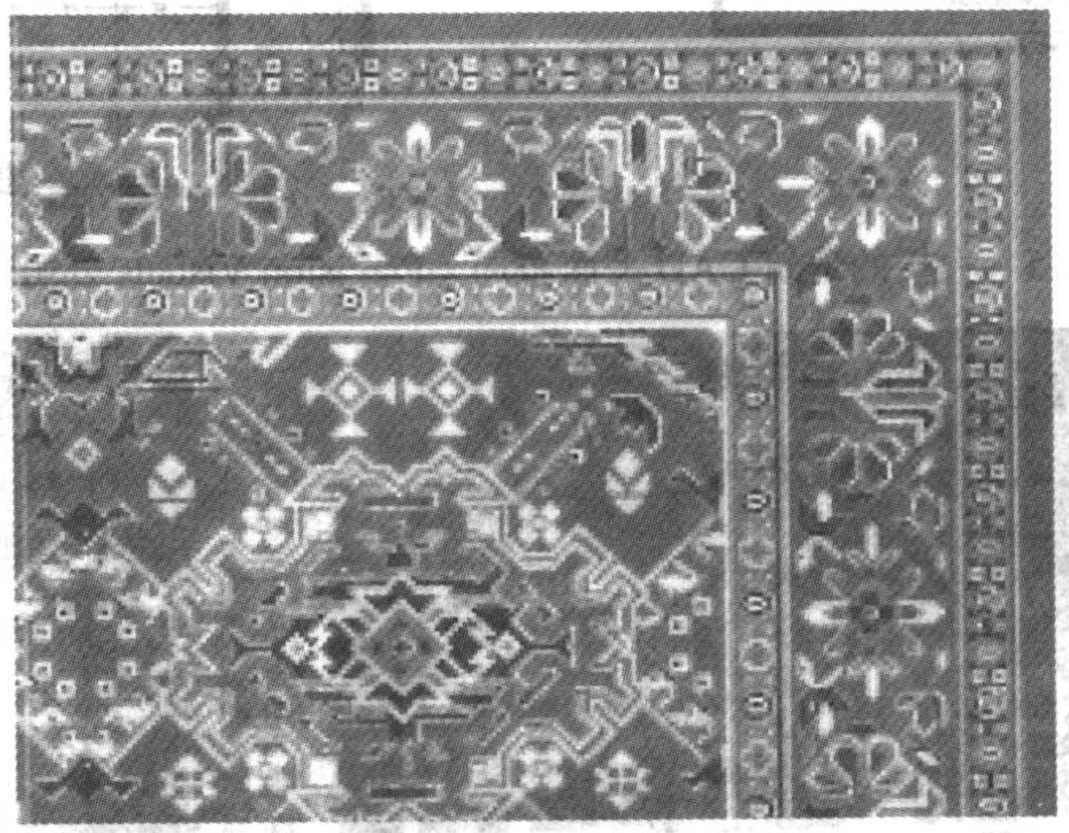


图 1.18 采用计算机图形设计技术生成的地毯图案（Lexidata 公司提供）

1.3 虚拟现实环境

计算机图形学的一个最新应用是生成虚拟现实环境（virtual-reality environment），在此环境中用户可与三维场景的对象进行交互。该环境中有专门的硬件设备提供三维观察效果，并允许用户在场景中拾取对象。

虚拟现实环境中的动画常用来训练大型设备的操作员以及分析各种机舱配置和控制安排的有效性。在图1.19中，当拖拉机驾驶员操纵控制器时，头盔中显示出前铲斗或反向铲的立体视图（参见图 1.20），就好像驾驶员坐在真正的拖拉机驾驶座上那样。这使得设计者可以了解在驾驶员位置可能看不到的前铲斗和反向铲的各种位置，以便在整个设计中更好地考虑它们。图 1.21 给出了拖拉机驾驶员所能看到的组合广角视图，它不是显示在虚拟三维布景上，而是显示在标准的视频监视器上。图 1.22 给出了一个可以在单独的窗口或其他监视器上显示的视图。

有了虚拟现实系统，设计者和其他人可以使用各种方式移动对象并与之进行交互。人们可以通过仿真方式“走入”房间或围绕大楼转圈欣赏特定设计的整体效果，从而测试建筑设计。我们甚至可以借助于一种专门的手套从场景中“抓取”对象、将其放回场景或从一处移到另一处。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

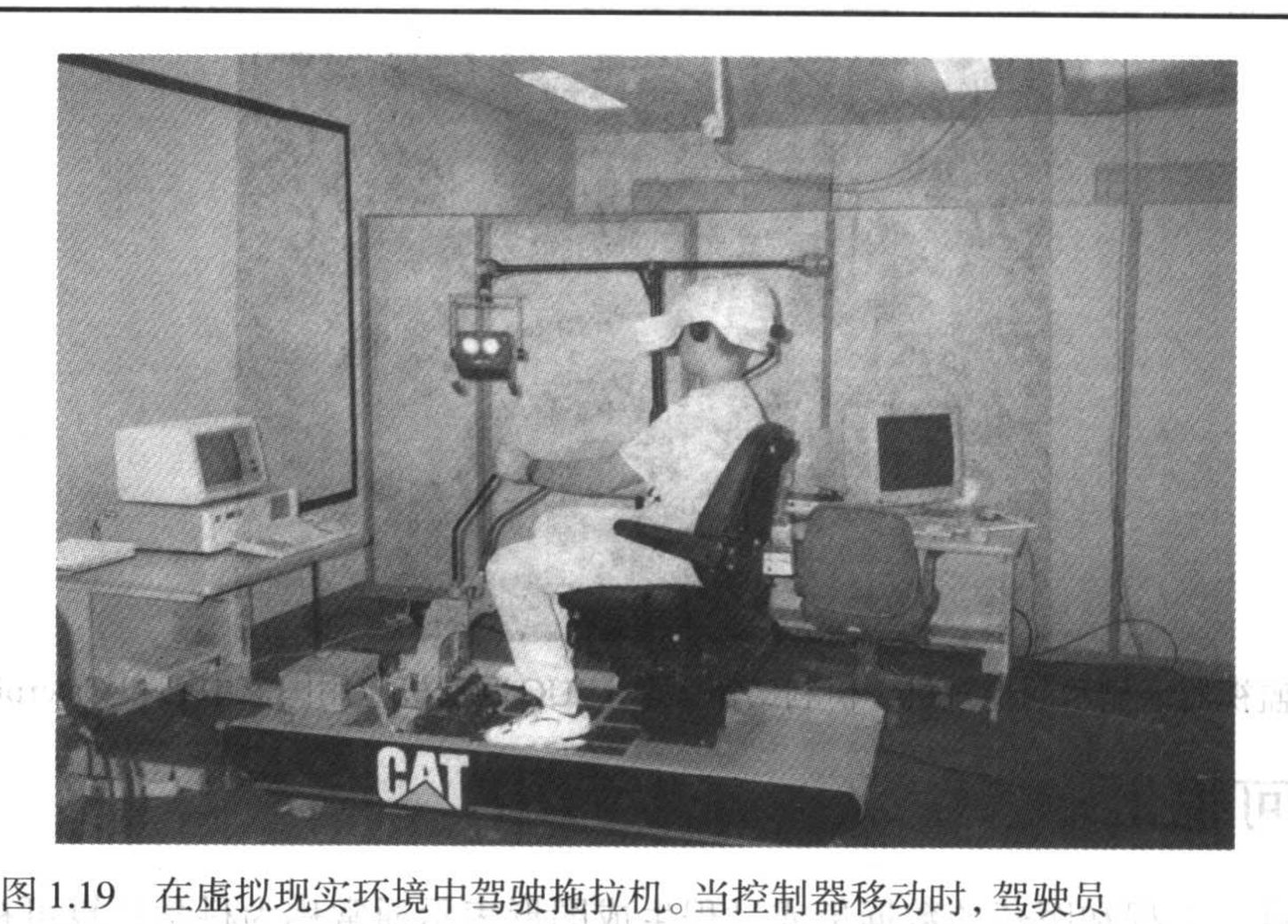


图 1.19　在虚拟现实环境中驾驶拖拉机。当控制器移动时，驾驶员通过头盔观察前铲斗、反向铲以及周围环境（伊利诺伊大学 UIUC 巨型机应用国家中心和 Caterpillar 公司提供）

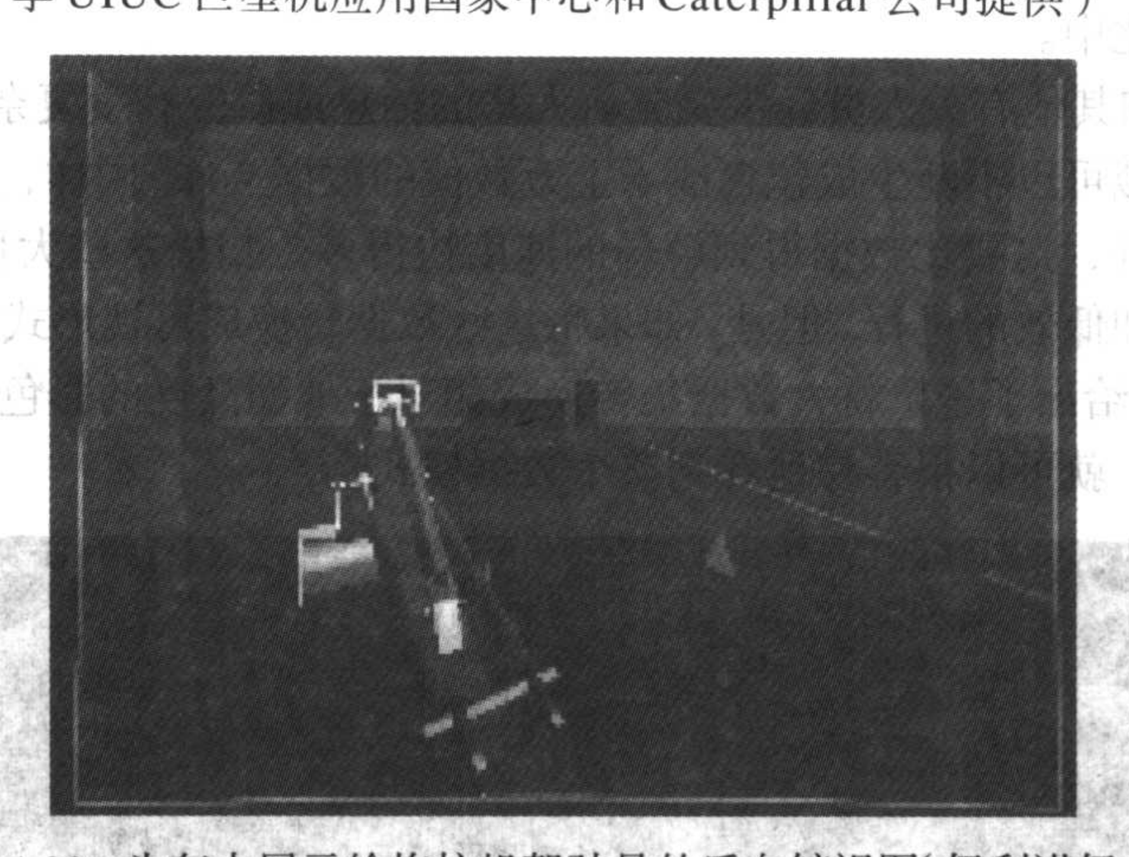


图 1.20　头盔中展示给拖拉机驾驶员的反向铲视图（伊利诺伊大学 UIUC 巨型机应用国家中心和 Caterpillar 公司提供）

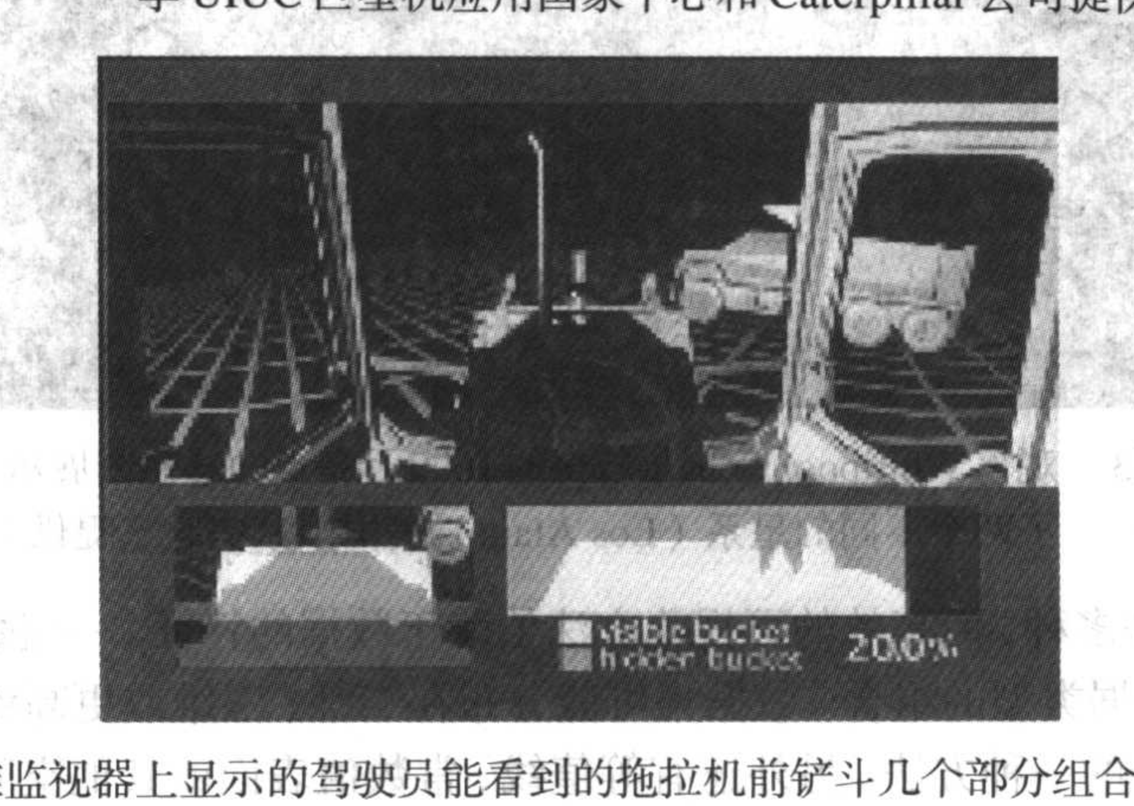


图 1.21　标准监视器上显示的驾驶员能看到的拖拉机前铲斗几个部分组合而成的广角视图（伊利诺伊大学 UIUC 巨型机应用国家中心和 Caterpillar 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

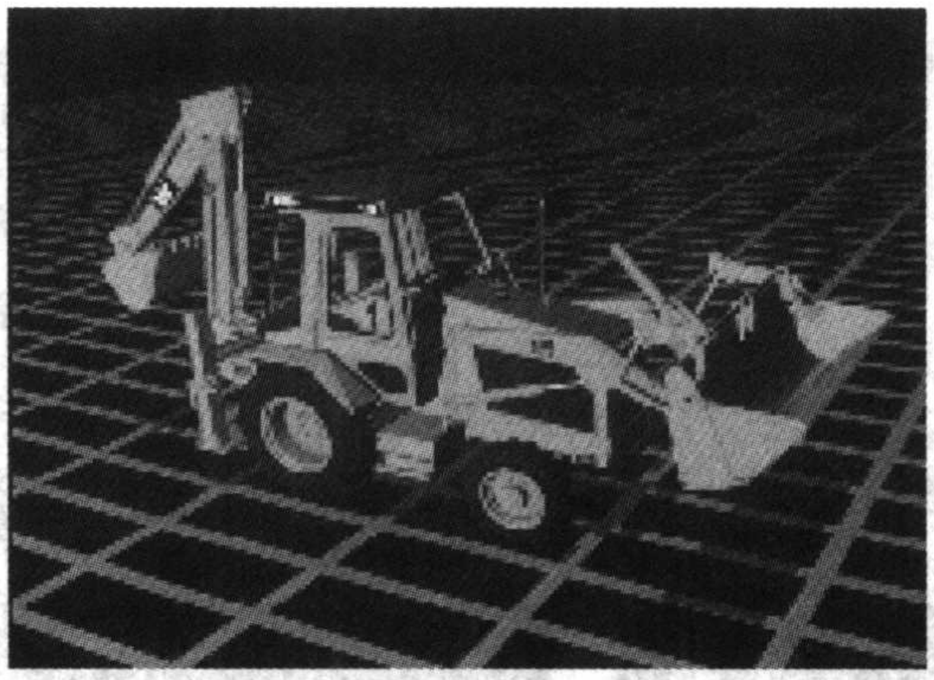


图 1.22 标准监视器上显示的拖拉机（伊利诺伊大学 UIUC 巨型机应用国家中心和 Caterpillar 公司提供）

1.4 数据可视化

为科学计算、工程和医药的数据集或过程生成图形表示通常称为科学计算可视化（scientific visualization）。而术语商务可视化（business visualization）则用在与贸易、工业和其他非科学计算领域相关的数据可视化中。

研究员、分析员和其他有关人员经常要分析大量的信息或研究高度复杂过程的行为。例如，计算机上进行的数值模拟可以不断生成包含成千上万数值的数据文件。同样，卫星摄像机等也在快速地积聚大量的数据文件，这要比数据得到解释的速度快得多。扫描这些大量的数据以确定趋势及相互关系是一个乏味和低效的过程。但是，如果将这些数据转换成可视形式，则趋势和模式就可以立刻呈现出来。图1.23给出了一个将大数据集转换成相对于地面高度的彩色图形例子。一旦我们按这种方法绘出密度值，就可以很容易地看到整个数据模式。

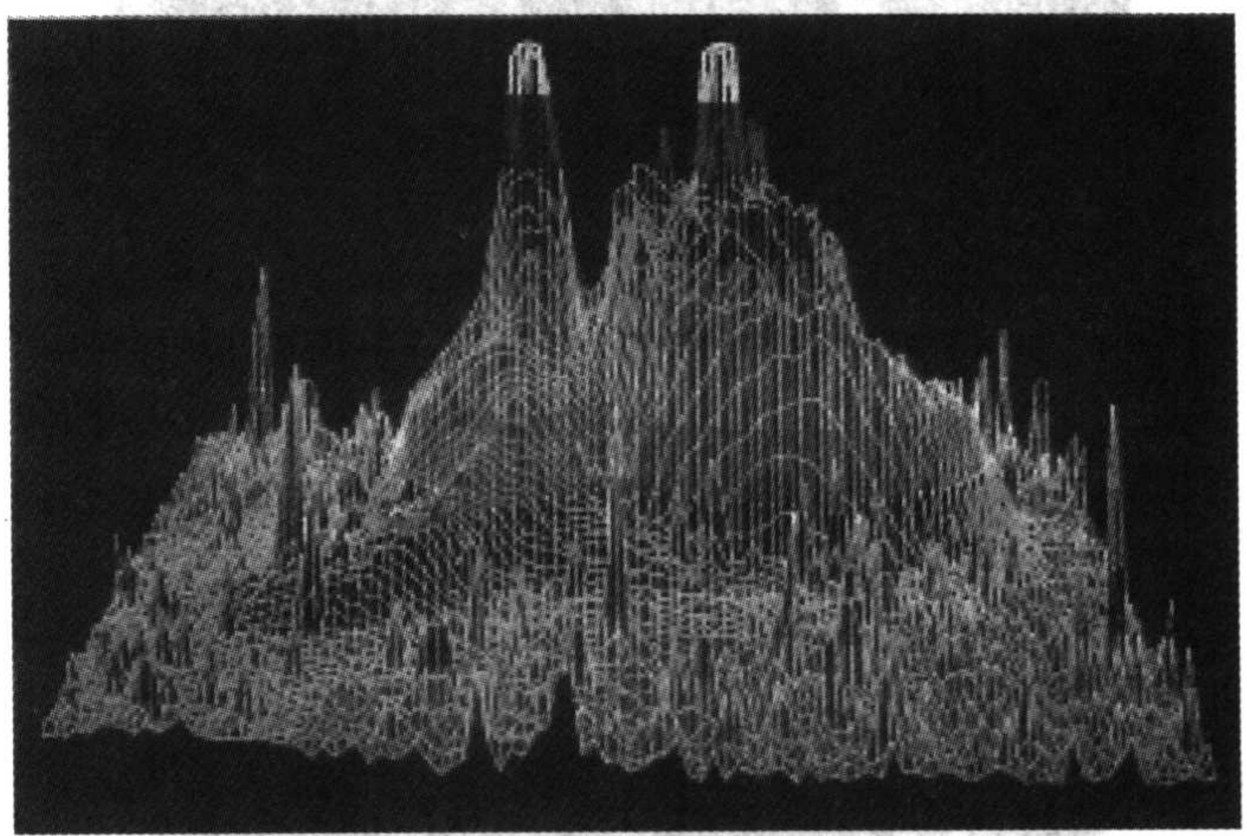


图 1.23 观察 Whirlpool Nebula 所得的 1600 万亮点的彩色图展示了两个不同的星系（Los Alamos 美国国家实验室提供）

数据集的类型有许多种，而高效的可视化方法依赖于数据的特征。一组数据可以包含标量、向量、高次张量或这些数据类型的组合。数据集可能分布在二维、三维或更高维的空间区域。颜色编码仅仅是数据集可视化的一种方法。另外还有等值线、常数值表面或其他空间区域的绘制，以及专门设计用来表达不同数据类型的形体等绘制技术。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

可视技术还用于帮助理解与分析复杂的过程和数学函数。图 1.24 给出了一个数学曲线函数的彩色图，而图 1.25 则给出了一张曲面图。图 1.26 给出了一个用四分法的分数维过程生成的物体，而图 1.27 显示了一个拓扑结构。科学家们还正在开发对一般数据类进行可视化的方法。图 1.28 给出了对一个球面上分布的数据进行建模和图示的一般技术。

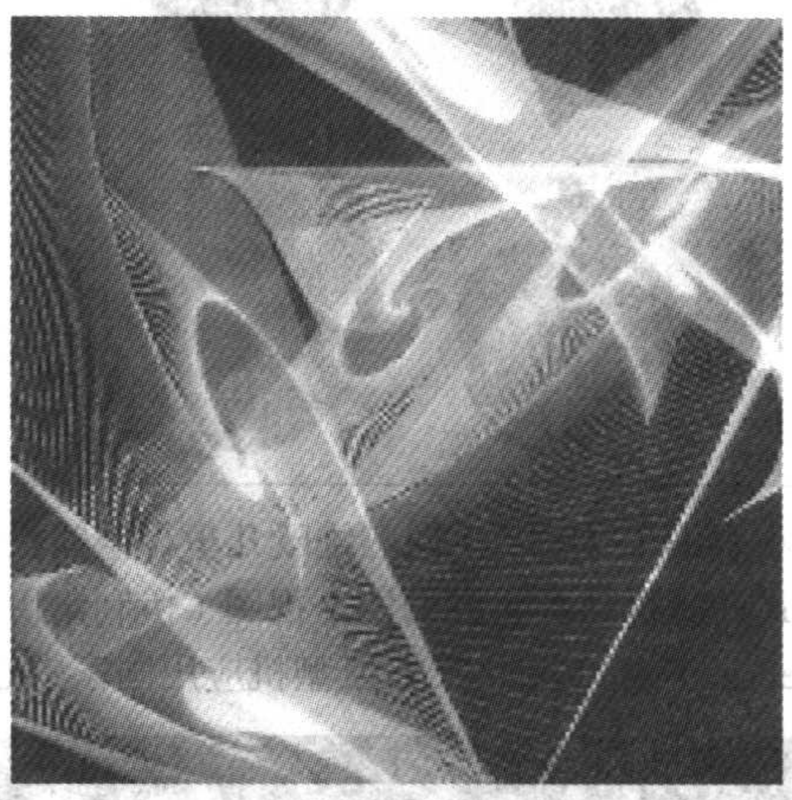


图 1.24　以多种颜色混合绘制的数学曲线函数（Los Alamos 美国国家实验室的 Melvin L. Prueitt 提供）

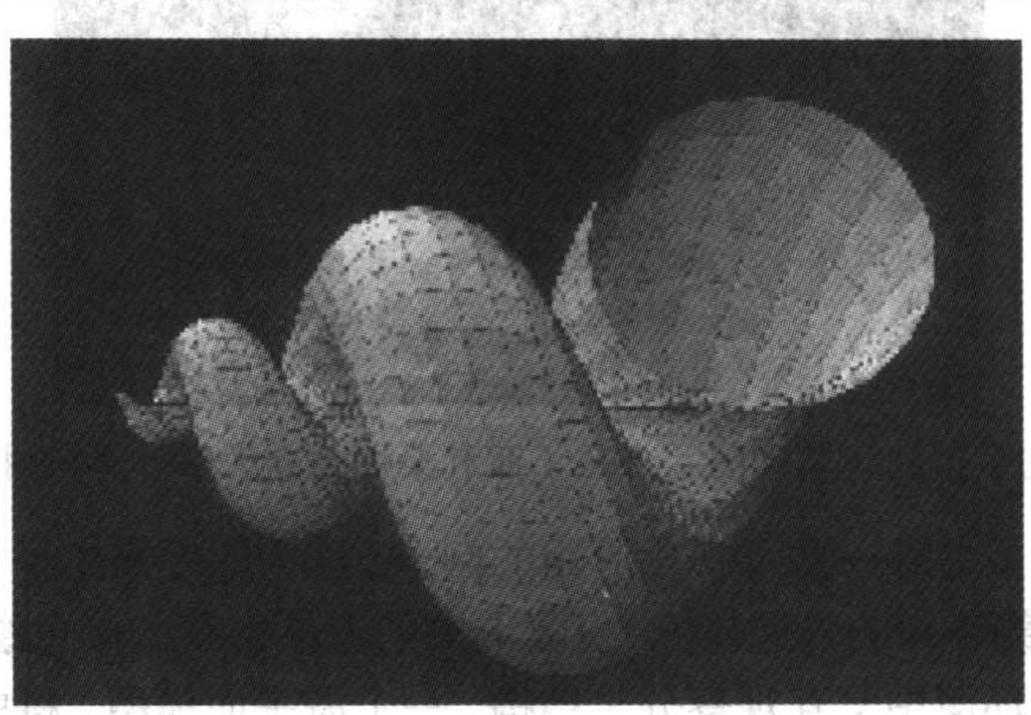


图 1.25　使用光照作用和表面绘制技术生成的表示三维函数的曲面（Mathematica 软件的开发者 Wolfram Research 公司提供）

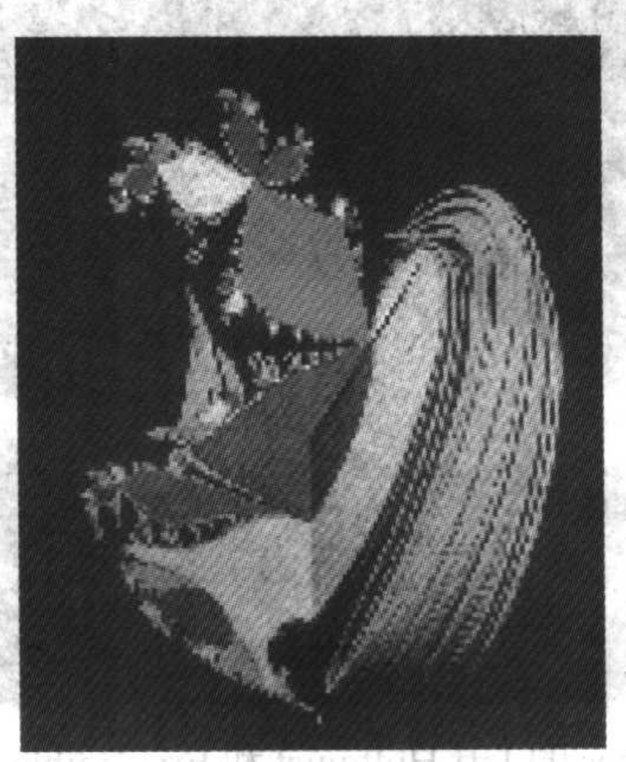


图 1.26　四维物体投影到三维空间，然后再投影到视频监视器上并给出颜色。该物体使用四分法和分数维二次幂过程生成，并切除了八分圆以展示复杂的 Julia 集（伊利诺伊大学计算机科学学院的 John C. Hart 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

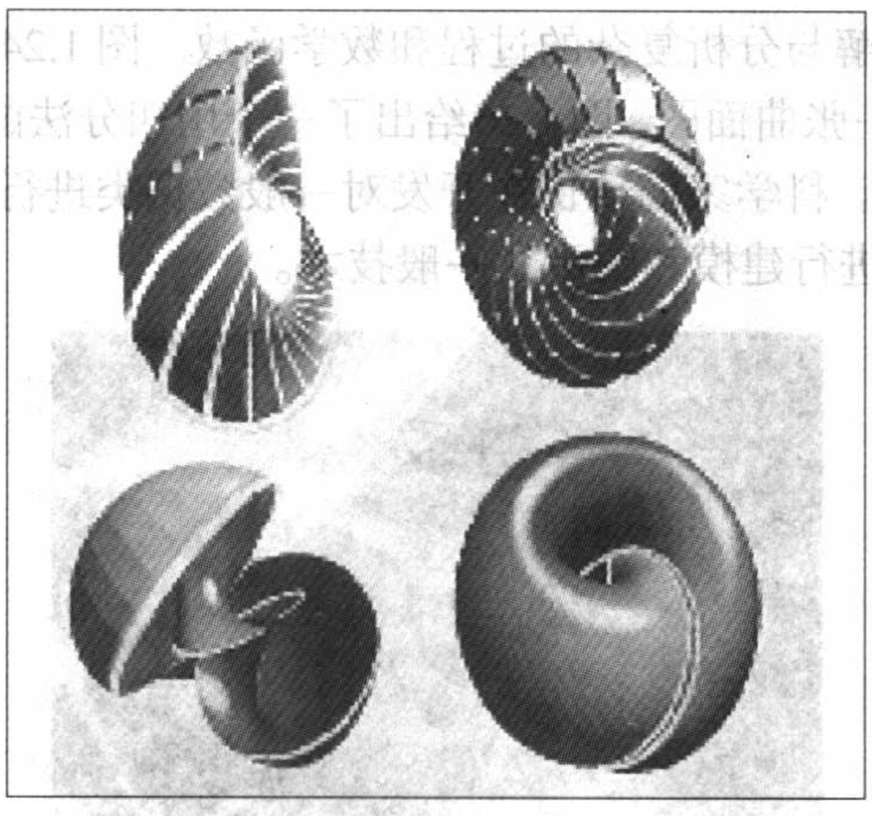


图 1.27　三球面最小表面（“蜗牛”）向三维欧几里得空间投影的实时、交互式计算机动画研究的四个视图（伊利诺伊大学数学系的 George Francis 和 UIUC 巨型机应用国家中心提供）

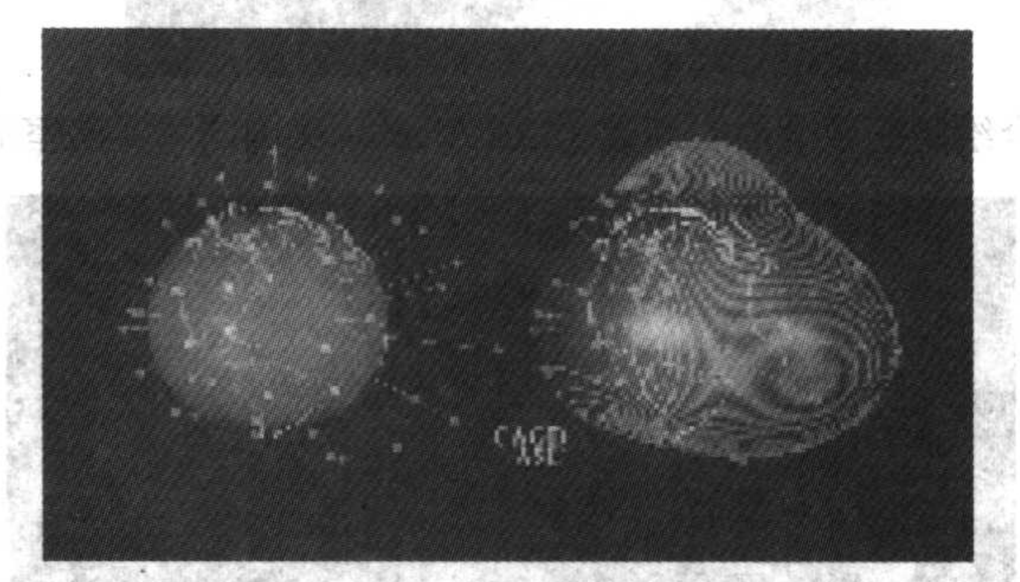


图 1.28　对分布在球面上的数据的建模和图示方法（亚利桑那州立大学计算机科学系的 Greg Nielson 提供）

图1.29～图1.42给出了另外一些可视化应用的例子。这些图形描述了宇宙飞船表面的气流、暴风雨的数值模型、金属内部断裂的传递研究、机翼上方的流体密度的彩色绘图、数据集的交叉切片、蛋白质建模、在虚拟现实环境中交互观察分子结构、一个海平面模型、科威特石油大火的模拟、空气污染研究、玉米生长研究、亚利桑那州的 Chaco 大峡谷废墟的重建以及汽车事故的统计图等。

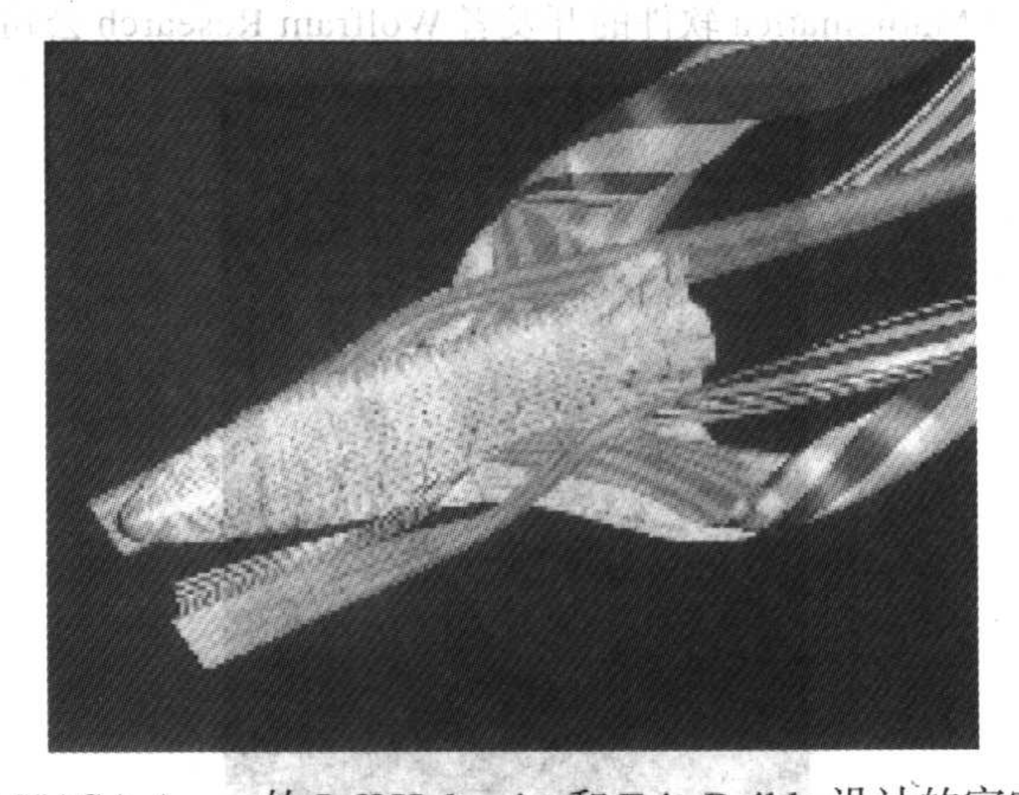


图 1.29　由 NASA Ames 的 Jeff Hultquist 和 Eric Raible 设计的宇宙飞船表面气流的可视化图形（NASA Ames研究中心的Sam Uselton提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

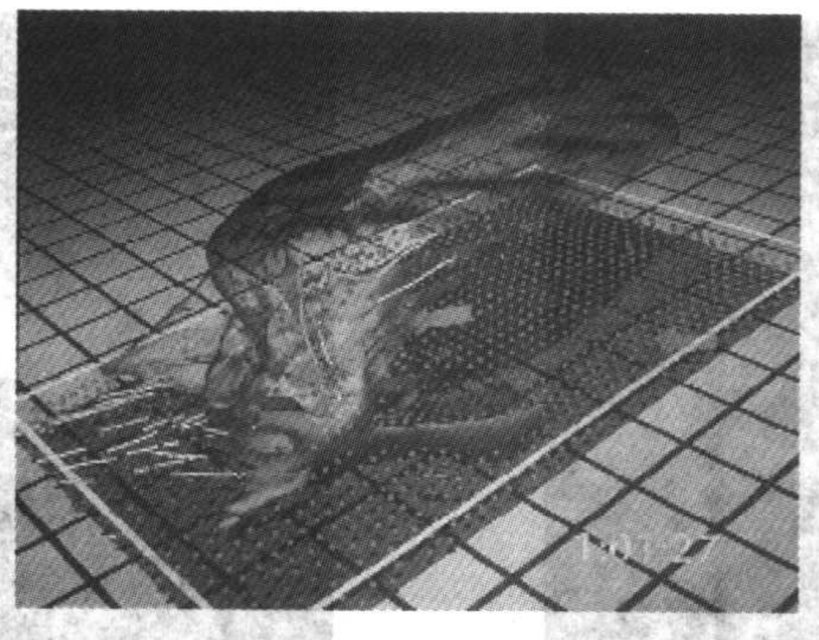


图 1.30　暴风雨内部气流的数值模型（伊利诺伊大学UIUC巨型机应用国家中心和大气科学系的Bob Wilhelmson提供）

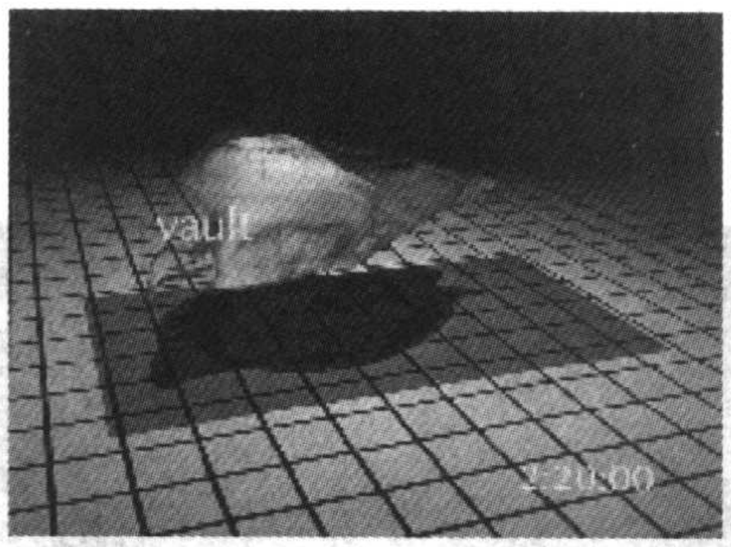


图 1.31　暴风雨表面的数值模型（伊利诺伊大学 UIUC 巨型机应用国家中心和大气科学系的 Bob Wilhelmson 提供）

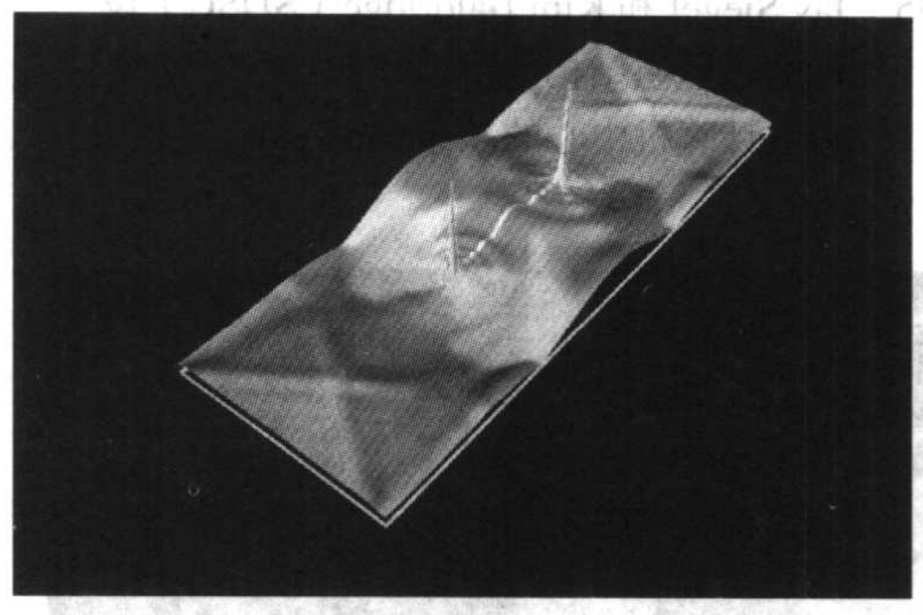


图 1.32　在金属内部断裂的传递研究中，由 Bob Haber 提供的应力能（stress energy）密度的可视化彩色图形（伊利诺伊大学 UIUC 巨型机应用国家中心提供）

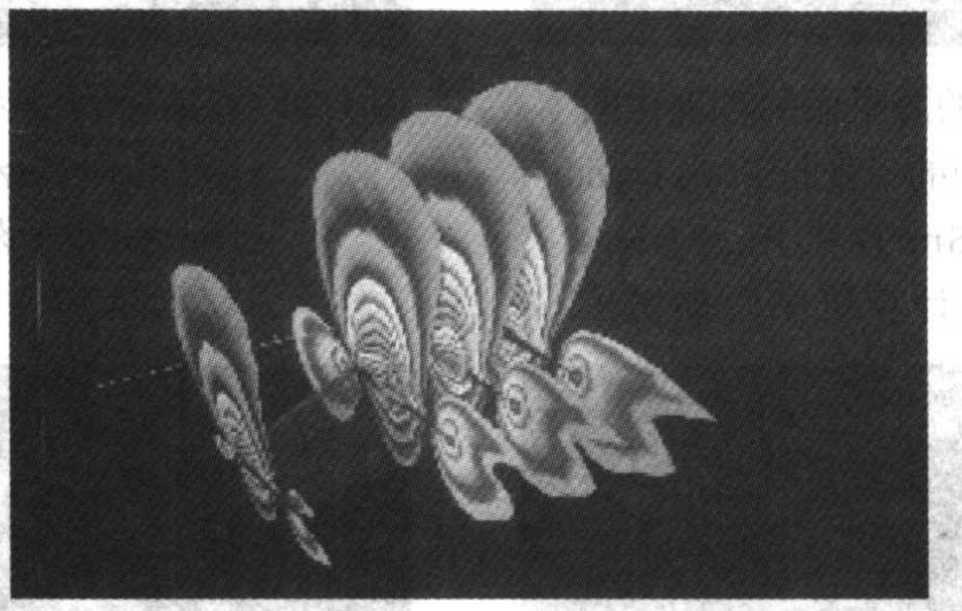


图 1.33　流体的动态模拟，演示了机翼上方的流体密度的彩色图形，由 Lee-Hian Quek、John Eickemeyer 和 Jeffery Tan 开发（新加坡信息技术协会提供）

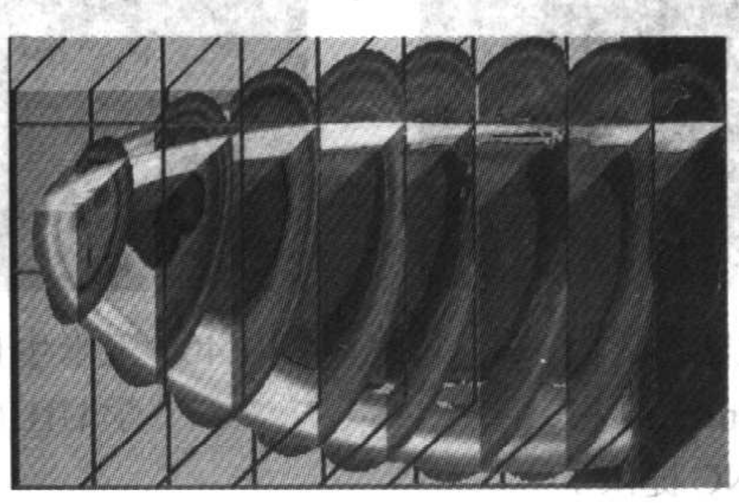


图 1.34　商用切片软件，给出了一个数据集的剖面切片的彩色数据值（Spyglass 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

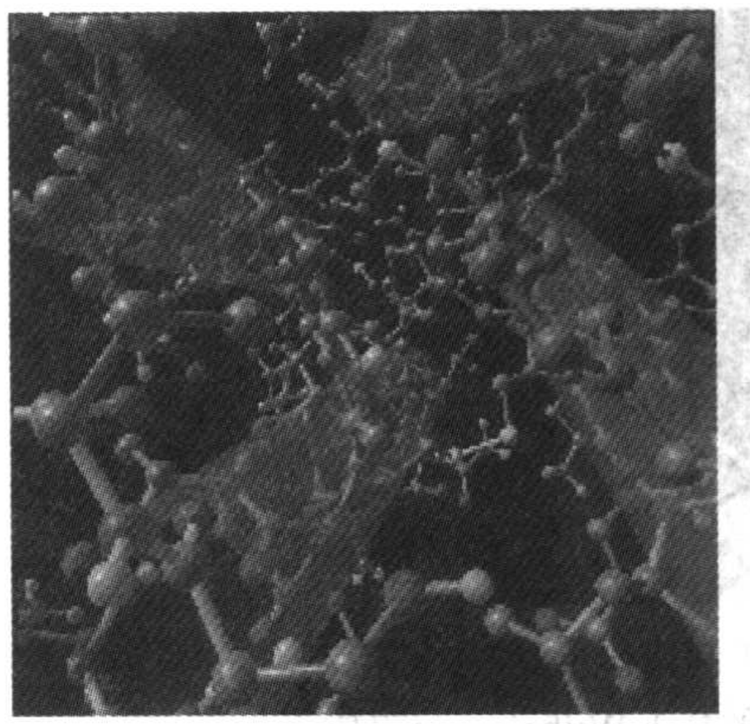


图 1.35 Jay Siegel 和 Kim Baldridge（SDSC）设计的蛋白质结构的可视化图形（圣迭戈超级计算机中心的 Stephanie Sides 提供）

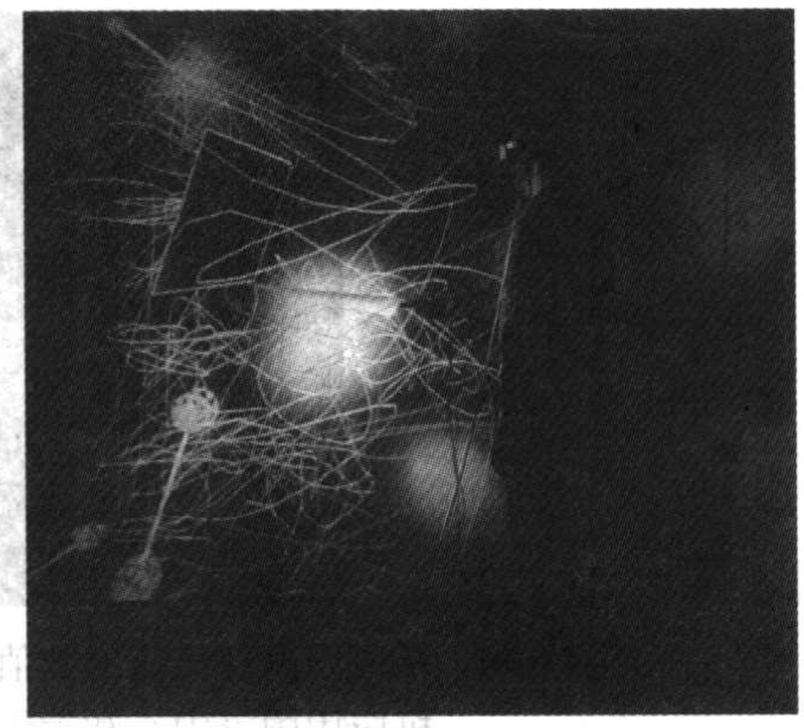


图 1.36 虚拟现实环境 CAVE 中科学家与分子结构的立体图进行交互（William Sherman 和伊利诺伊大学 UIUC 巨型机应用国家中心提供）

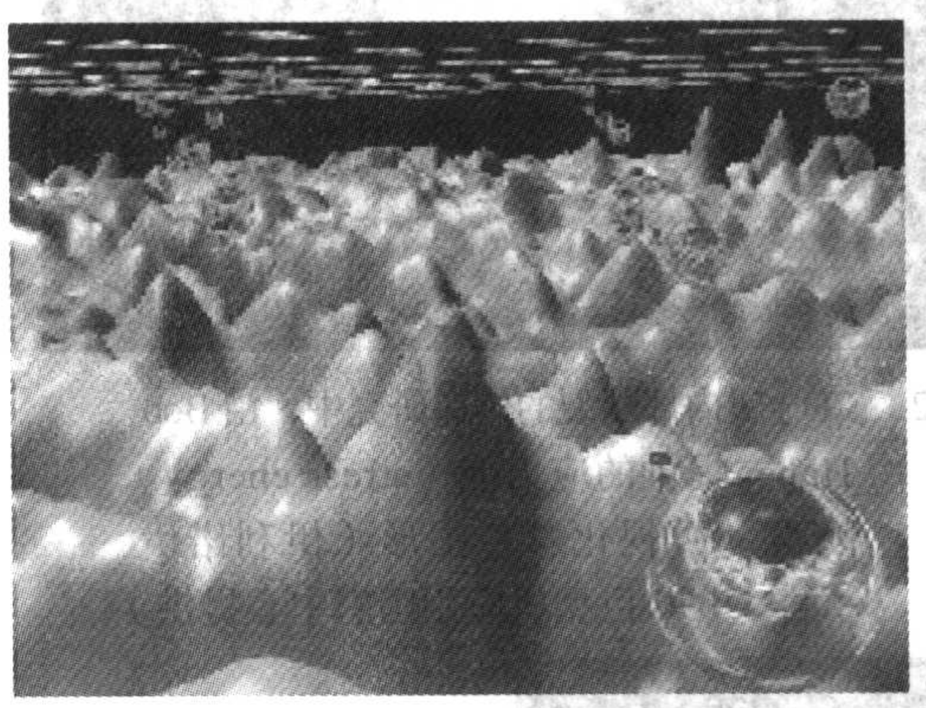


图 1.37 从卫星数据中得到的海平面的可视化图形（立体重叠照相中的一张图），由 David Sandwell、Chris Small 和 Jim Mcleod 设计（圣迭戈超级计算机中心提供）

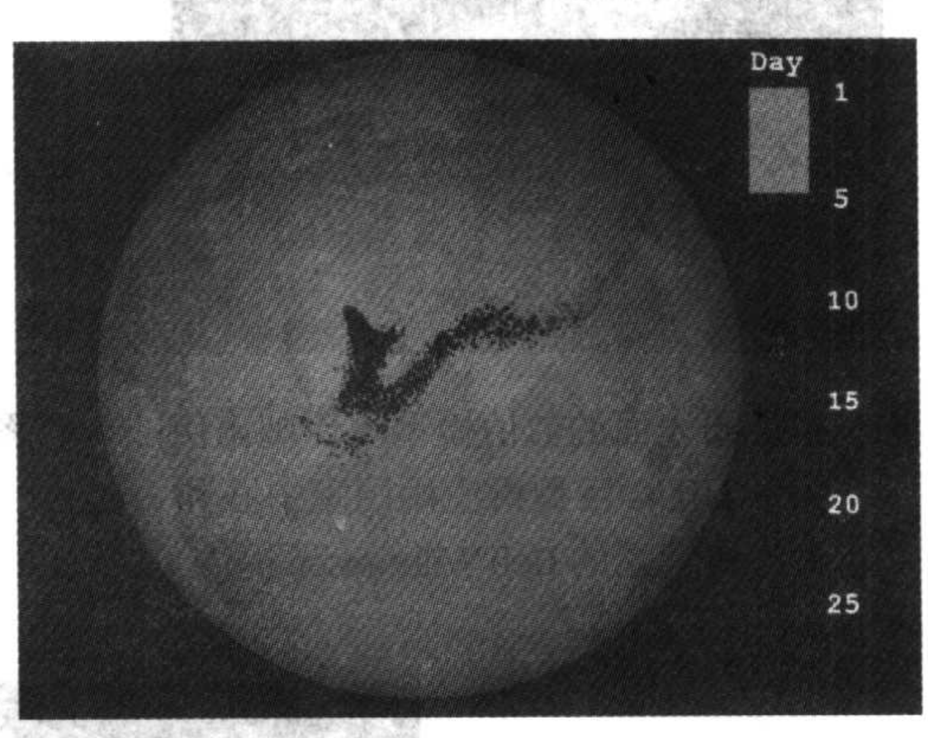


图 1.38 科威特石油大火的影响范围的模拟图，由 Gray Glatzmeier、Chuck Hanson 和 Paul Hinker 设计（Los Alamos 美国国家实验室的 Mike Krogh 提供）

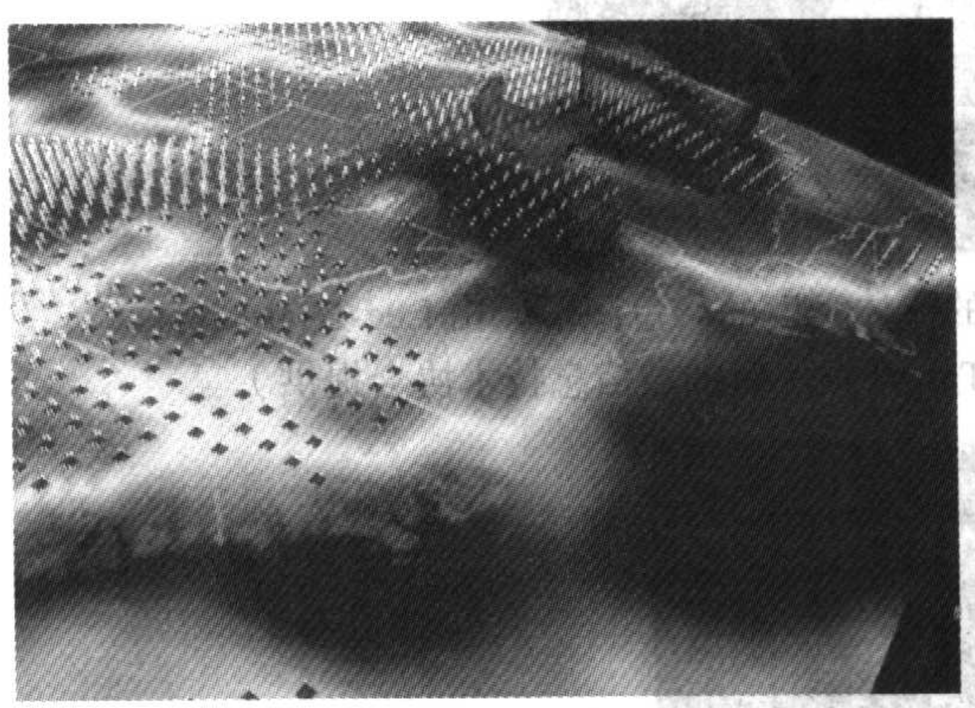


图 1.39 地球表面污染的可视化图形，由 Tom Palmer、Chris Landreth 和 Dave Bock 设计（MCNC 提供）

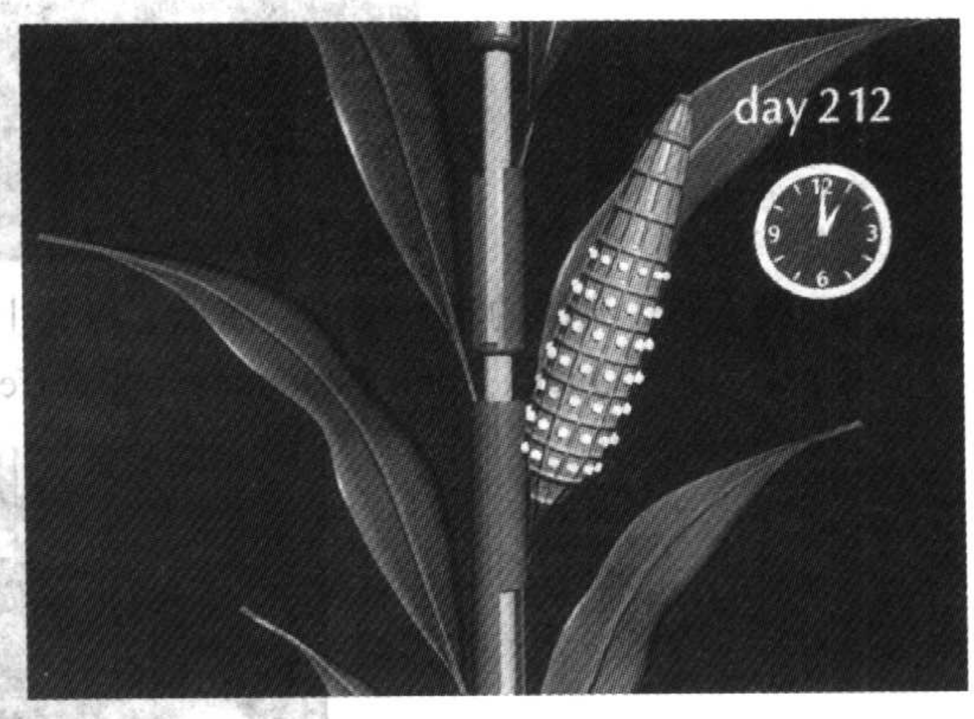


图 1.40 玉米生长动画中的一帧（伊利诺伊大学 UIUC 巨型机应用国家中心提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

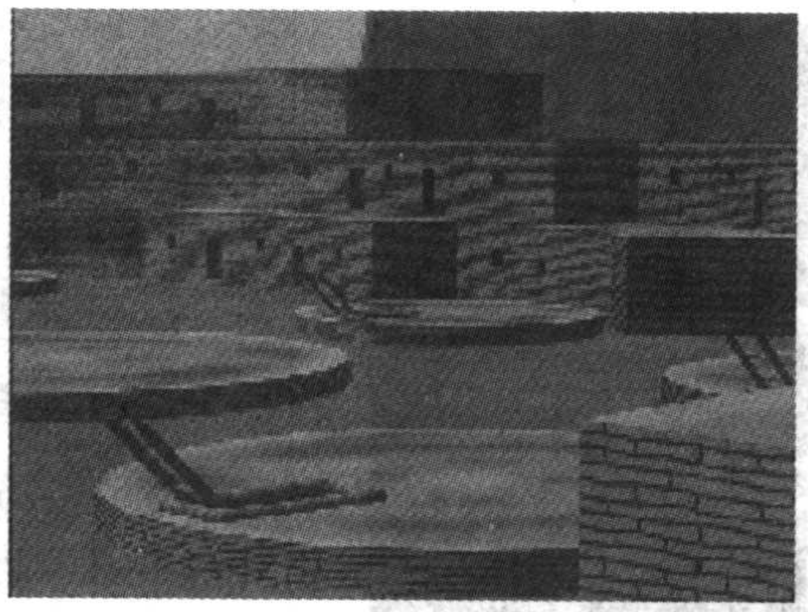


图 1.41　重建亚利桑那州的 Chaco 大峡谷废墟的可视化图形
（Los Alamos美国国家实验室的Melvin L. Prueitt提供）

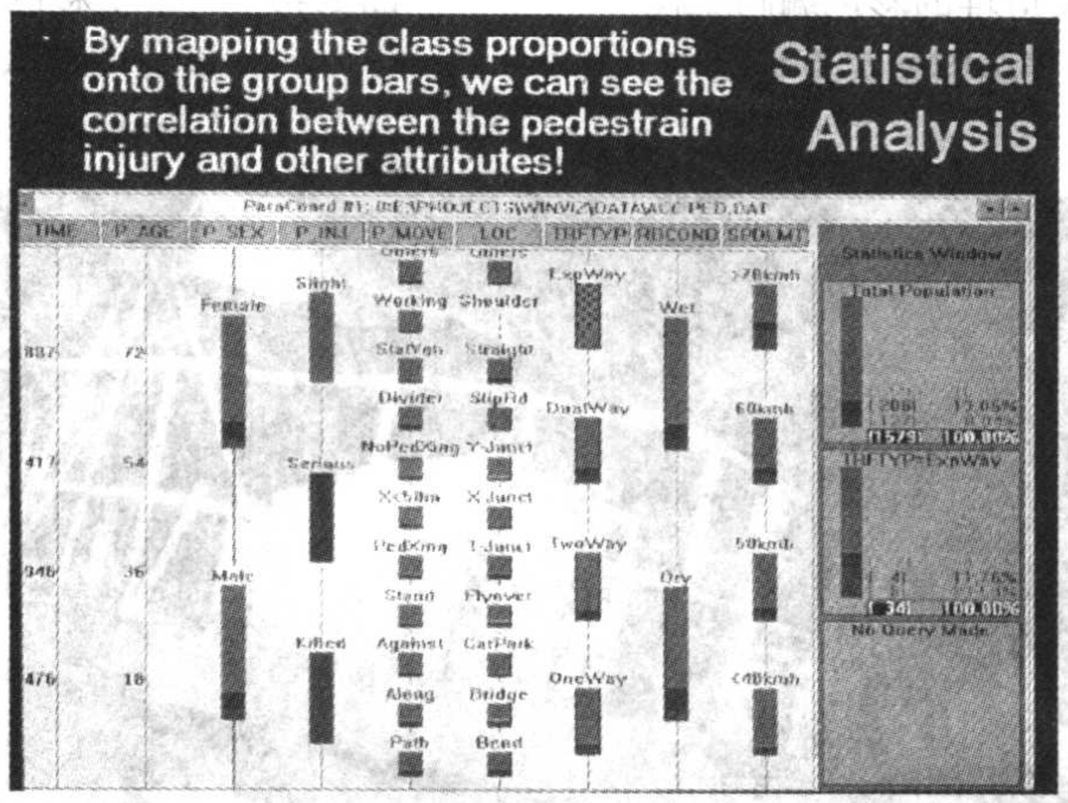


图 1.42　一种由新加坡信息技术协会可视化小组开发的、称为WinViz的原型技术，用于可视化表格形式的多维数据，这些数据与汽车事故中的统计数据相联系（Oracle 公司的 Lee-Hian Quek 提供）

1.5　教学与培训

计算机生成的物理模型、财政模型和经济系统模型常用做教学的辅助工具。物理过程、生理功能、人口趋势模型或设备的模型、如图 1.43 所示的彩色图表等都可以帮助学生理解系统操作。

有些方面的培训要设计专门的硬件系统。例如，船长、飞行员、大型设备操作员和空中交通管制人员的实习与培训模拟系统就是这样一种专用系统。有些模拟器没有显示屏幕，例如仅有飞行设备控制板的飞行模拟器，但是大多数模拟器配有屏幕，可以对外部环境进行可视化显示。图 1.44 和图1.45给出了带有内部观察系统的大型模拟器的实例。图1.46(b)和图1.46(c)则给出了另一类观察系统。其中，多组观察屏幕安装在模拟器的前端，而显示飞行外景的彩色投影器则放在屏幕控制板上。图1.47给出了飞行模拟器中飞机座舱后面的指导人员区域。指导人员可以在键盘上输入参数，控制飞机的性能和飞行环境。在培训和测试过程中，可以在监视器上观察飞机的飞行路径和其他数据。

图 1.48~图 1.50 给出了飞机、军舰和宇宙飞船等各种模拟器生成的场景。图 1.51 给出了一个汽车驾驶模拟器及辅助的街景，其中图1.51(a)图给出了模拟器以及通过挡风玻璃可以看到的观察屏幕的内部结构，图 1.51(b)则给出了一般的交通街景。尽管汽车模拟器可以用做培训系统，但是它们一般仅用于研究驾驶员在关键情况下的行为。在各种交通条件下驾驶员的反应将作为优化汽车设计的基础，从而尽可能地提高交通的安全性。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

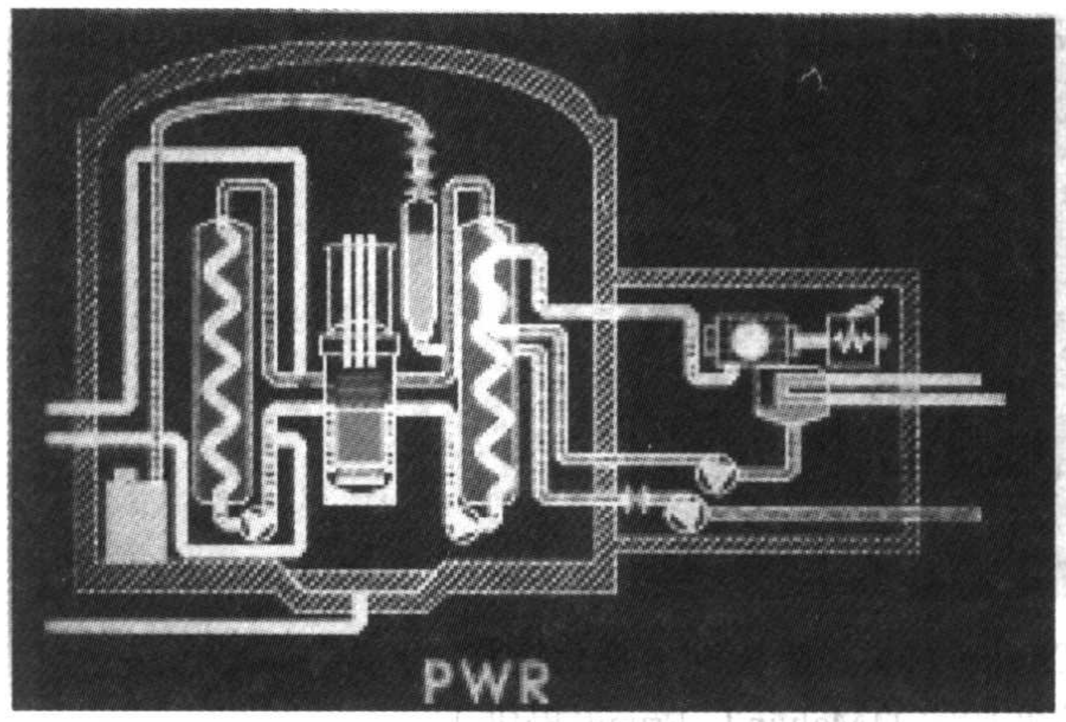


图 1.43 用来解释核反应堆的彩色编码流程（Los Alamos 美国国家实验室提供）

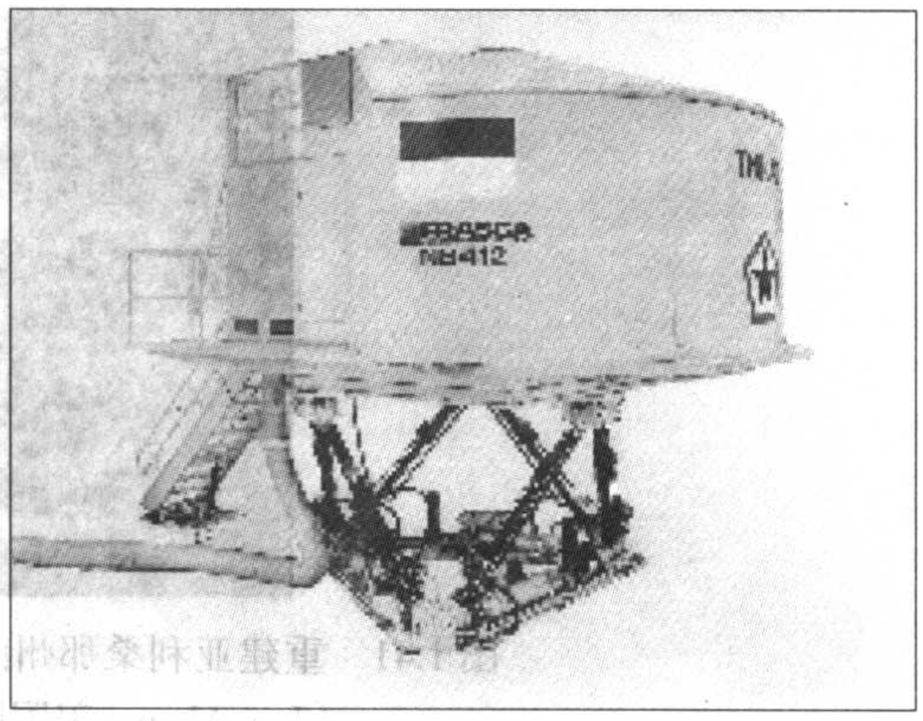


图 1.44 一个全彩色视觉系统和六自由度运动的大型嵌入式模拟器（Frasca International 提供）

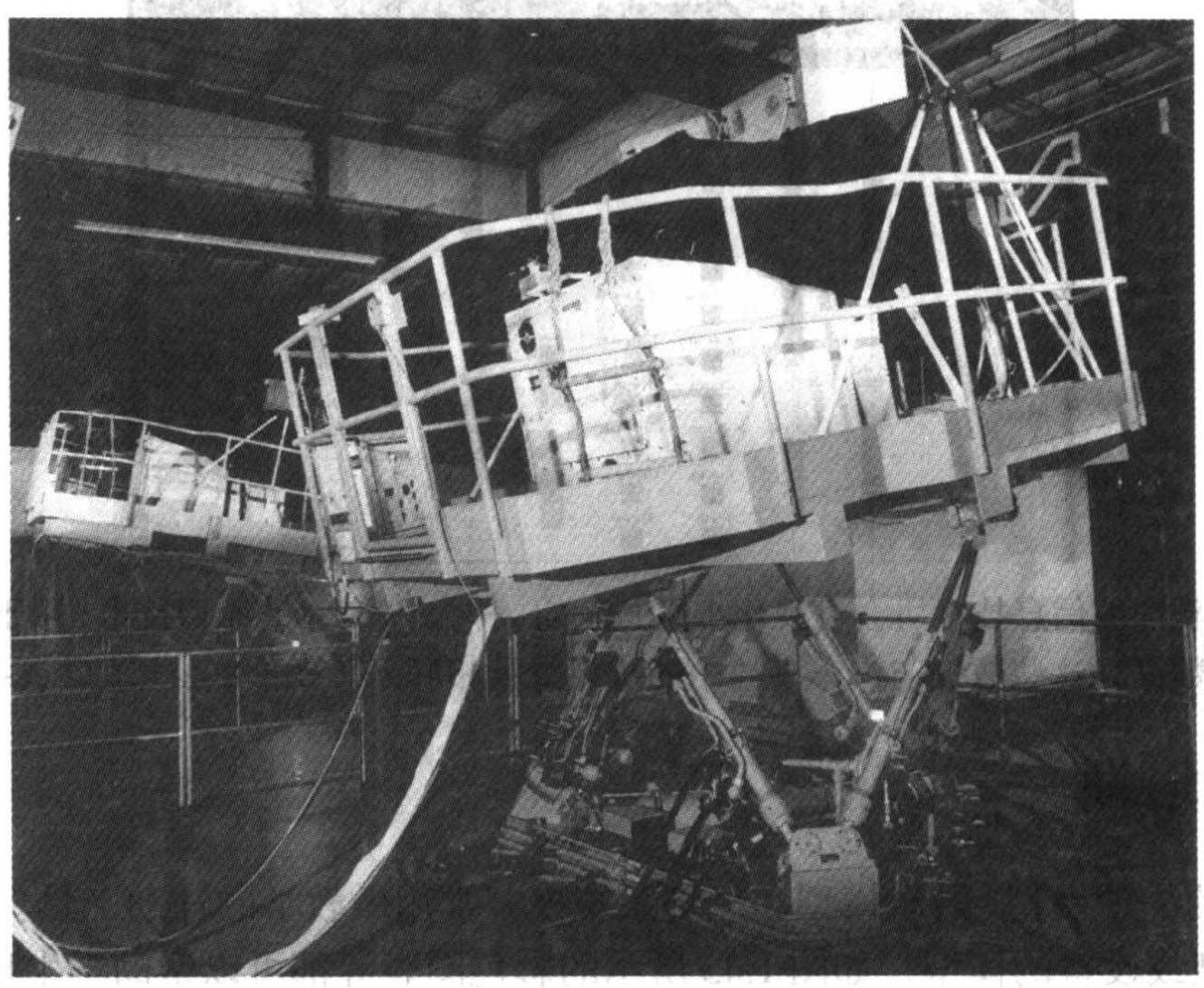


图 1.45 使用直观成像系统的军用坦克模拟器（Mediatech 和 GE Aerospace 提供）

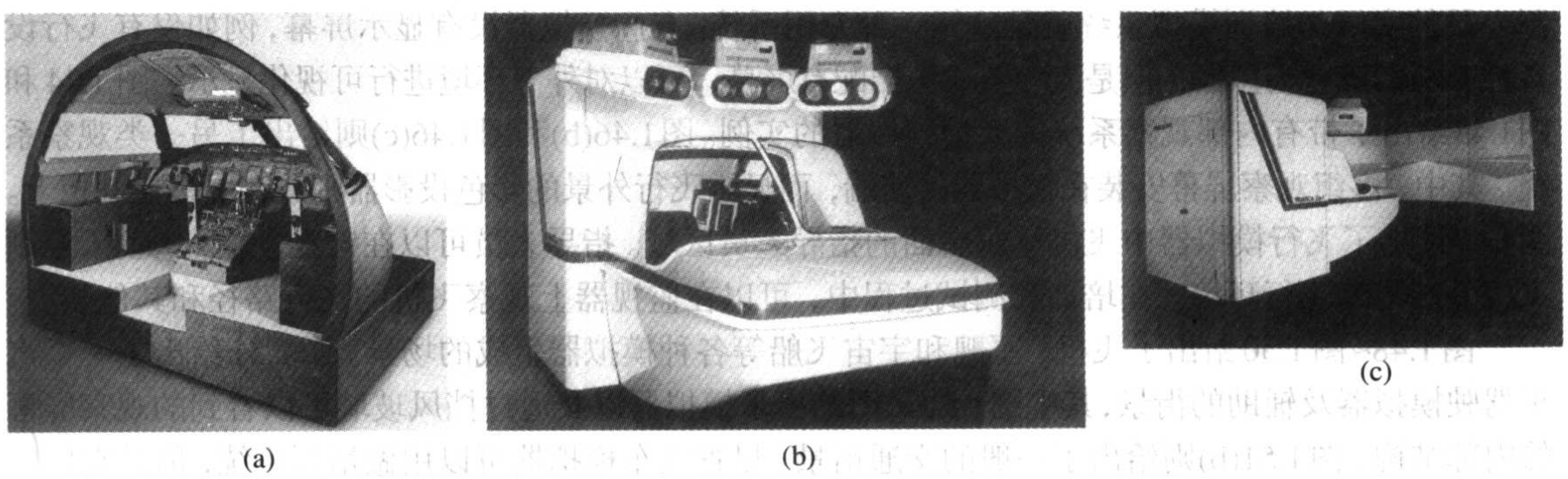


图 1.46 (a) 双控飞行模拟器的机舱内部；(b) 外部全彩色观察系统；(c) 小型飞行模拟器（Frasca International 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

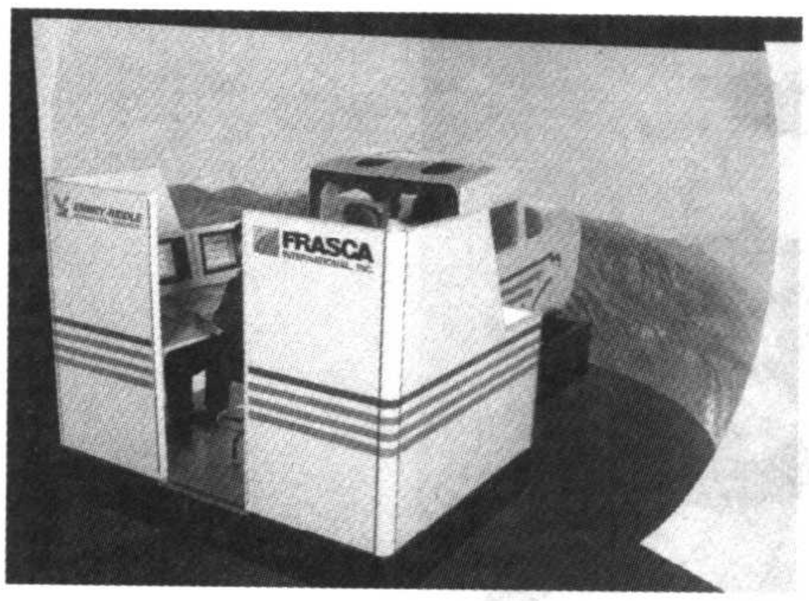


图 1.47　在飞行模拟器中指导人员的区域。该设备允许指导人员监控飞行状况并设置飞机和环境参数（Frasca International 提供）

图 1.48　飞行模拟器的景象（Evans & Sutherland 提供）

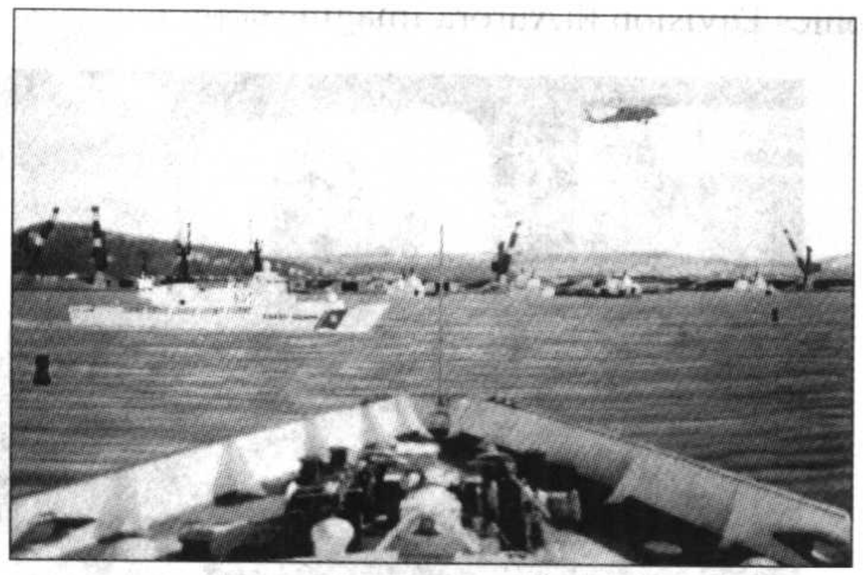


图 1.49　用于海军的模拟器的景象（Evans & Sutherland 提供）

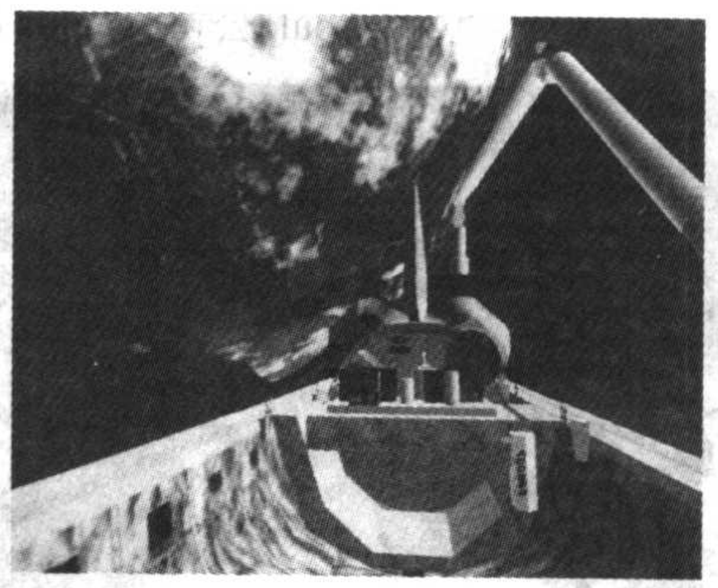
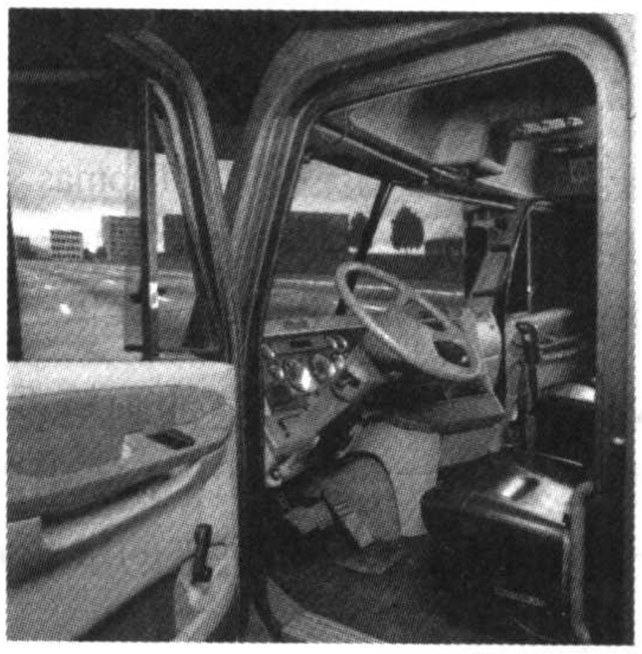


图 1.50　航天飞机模拟器的景象（Mediatech 和 GE Aerospace 提供）



(a)

(b)

图 1.51　(a) 汽车模拟器的内部；(b) 可展示给驾驶员的街景（Evans & Sutherland 提供）

1.6　计算机艺术

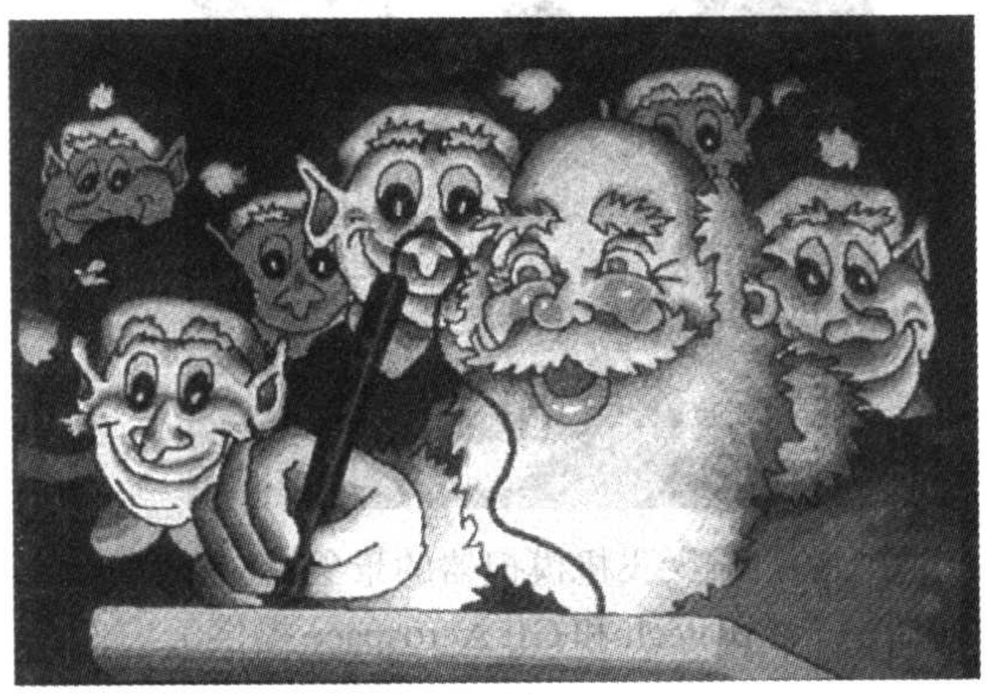
计算机图形学广泛使用于美术和商务艺术应用中。艺术家使用各种计算机方法，包括专用硬件、商业化的软件包（如 Lumena）、符号数学程序（如 Mathematica）、CAD 软件包、桌面出版软件和动画软件来设计物体的外形及描述物体的运动。

图 1.52 给出了艺术家在监视器上利用**画笔程序**（paintbrush program）绘画的象征性景象。实际上，绘画是以电子方式画在带有触笔的数据板上，该触笔能模拟不同的笔划、粗细线条以及颜色。图 1.53 给出了卡通画家创作的人物素描，看上去他们正在忙于创作。

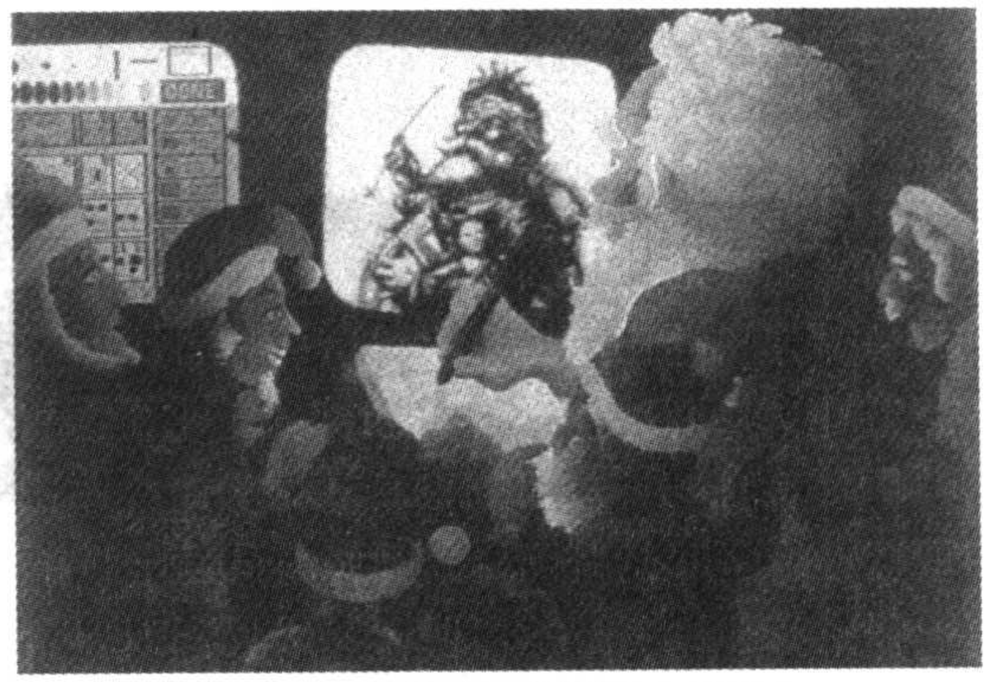
Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图 1.52　用画笔程序生成卡通图，形象地表明艺术家在监视器上的工作
（Gould公司Imaging & Graphics Division和Aurora Imaging提供）



(a)



(b)

图 1.53　展示“艺术家”用画笔系统进行创作的卡通图片：(a)在图形数据板上画图时几个小精灵在看着屏幕上图像的生成；(b)卡通艺术家和小精灵叠加到Thomas Nast的名画Saint Nicholas上，使用摄像机将其输入系统，然后进行缩放和定位（Gould 公司 Imaging & Graphics Division 和 Aurora Imaging 提供）

图1.54为模拟了梵高笔划的电子图画，这是使用配有Wacom无绳压感触笔的画笔系统生成的。触笔将变化的手的压力转换成各种粗细线条、笔划尺寸和颜色等级。图1.55给出了使用该触笔和有关软件创作的水彩画，该软件允许艺术家创作模拟不同的干燥时间、水分和轨迹的水彩画、粉笔画或油画。图 1.56 给出了将画笔方法和扫描图像混合起来的例子。

艺术家使用各种计算机技术来生成图像，为了生成如图 1.57 那样的图片，艺术家混合使用了三维建模软件包、纹理影射软件、绘图软件及 CAD 软件。图 1.58 给出了在艺术家不加干预的情况下使用一个专门软件自动在笔式绘图仪上绘制的图画。

图1.59给出了一个“数学”美术的实例。艺术家可混合使用数学函数、分数维过程、Mathematica软件、喷墨打印机以及其他系统来生成各种三维和二维形状及立体图像。图1.60给出了另一个通过数学关系图生成的电子图画的例子。这种作品的设计通常与频率变化以及音乐作品中其他用来制作具有视听效果的视频参数相联系。

商务艺术也将这些绘画方法用于标牌等商用美术设计、组合图文的页面布局设计以及电视广告等领域。图 1.61 给出了组合图文的页面布局设计的例子。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

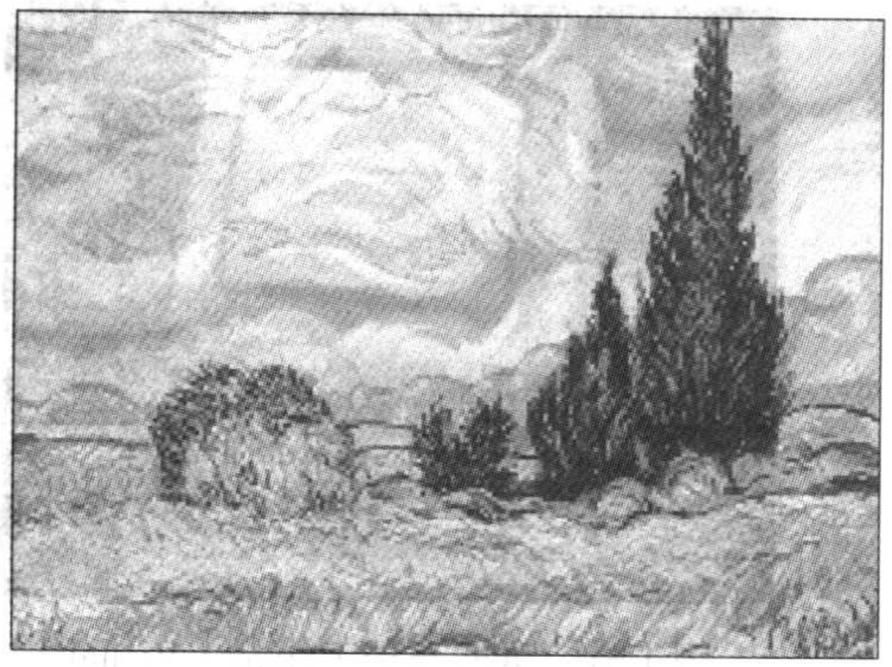


图 1.54　图形艺术家 Elizabeth O'Rourke 使用配有无线压感触笔创作的模仿梵高的图画（Wacom Technology 公司提供）

图 1.55　Time Arts 公司的 John Derry 使用配有无线压感触笔和 Lumena gouache-brush 软件创作的电子水彩画（Wacom Technology 公司提供）

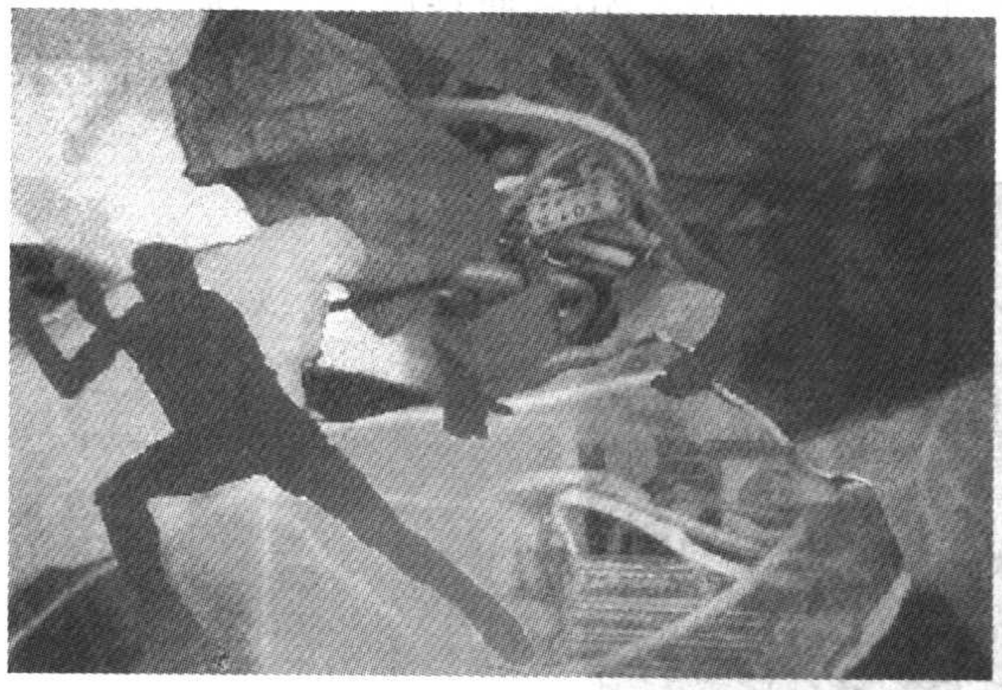


图 1.56　名为电子雪崩（Electronic Avalanche）的作者使用了配有图形数据板的计算机和 Lumena 软件将叶子、花瓣和电子部件与扫描图像混合，描述了现实中相互纠缠的一种状态（Williams Gallery 提供）

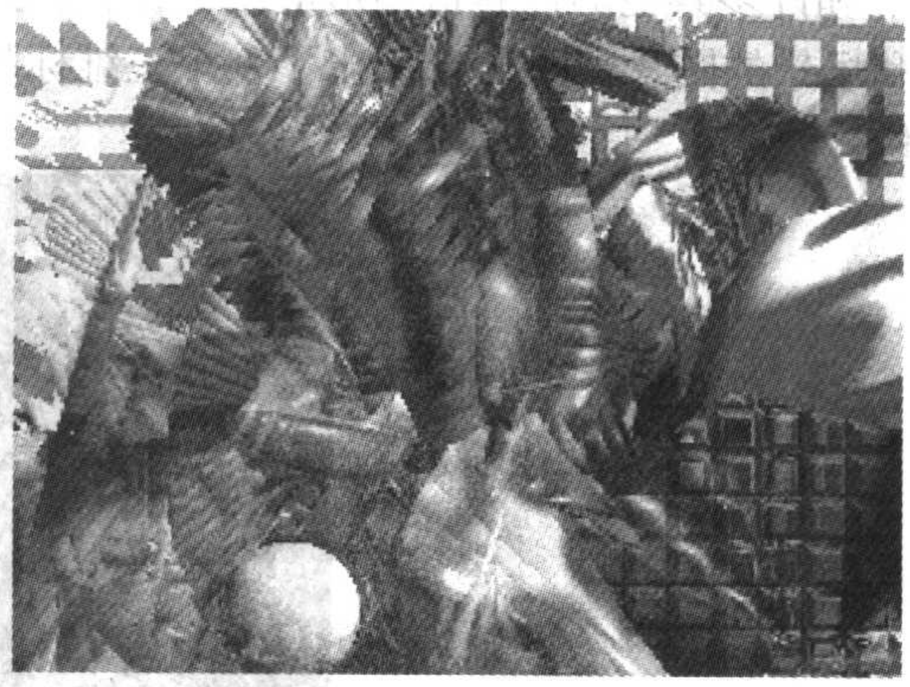


图 1.57　混合使用图形板、三维建模、纹理影射和一组变换等方法，生成了一张 Spheres of Influence 系列中题为 Whigmalaree 的电子画面（Williams Gallery 提供）

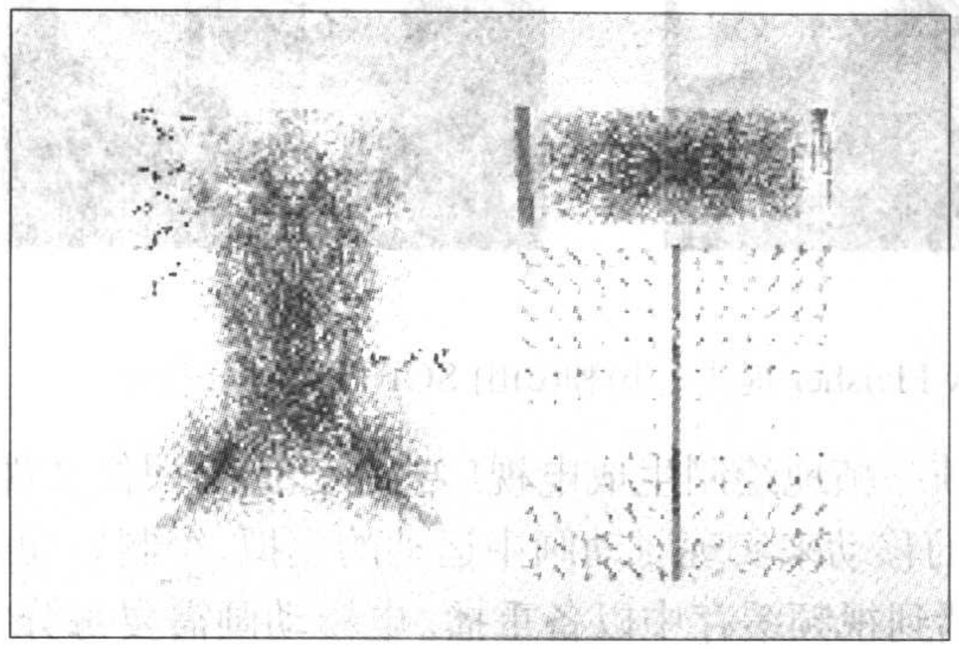


图 1.58　为模拟艺术家风格而设计的软件在笔式绘图仪上输出的电子图。笔式绘图仪带有多种画笔以及绘图设备，其中包括中国毛笔（Williams Gallery 提供）

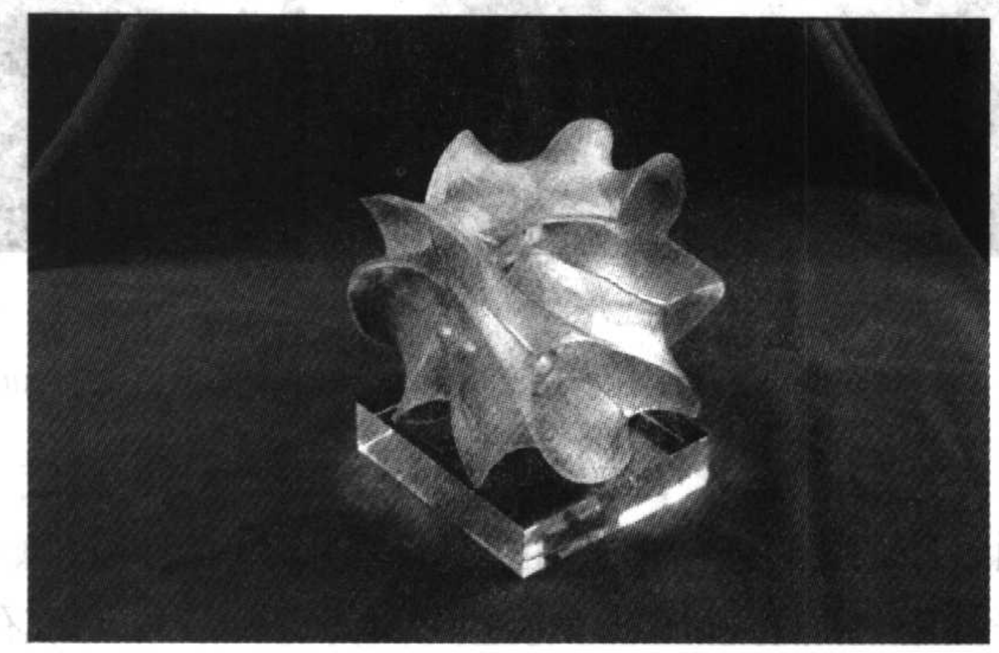


图 1.59　这一创作基于 Fermat 的最后定理 $x^n+y^n=z^n$（$n=5$）的可视化，由印第安纳大学计算机科学系的 Andrew Hanson 完成。该图像使用 Mathematica 和 Wavefront 软件绘制而成（Williams Gallery 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

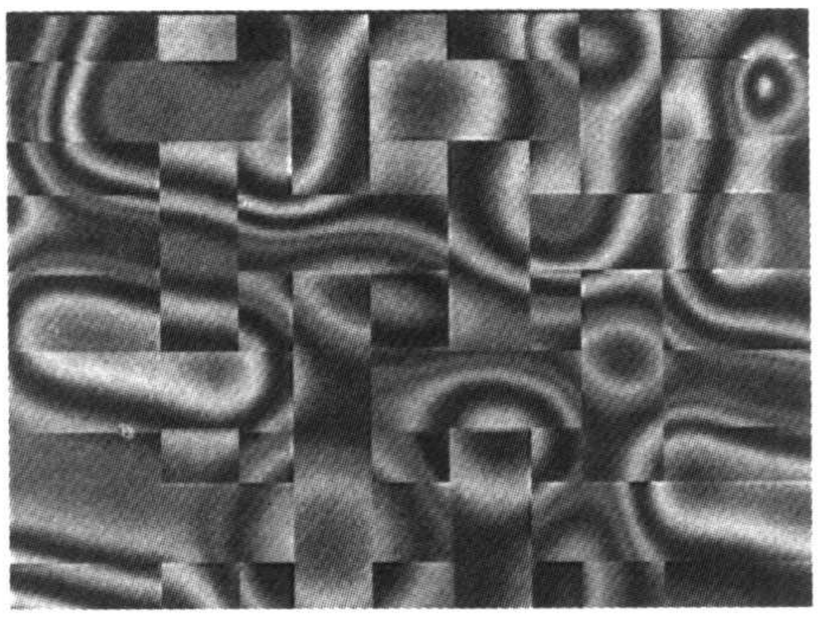


图 1.60 艺术合成器使用数学函数、分数维过程和超级计算机进行各种试验，将形态和颜色合成到音乐作曲中（Vanderbilt 大学的 Brian Evans 提供）

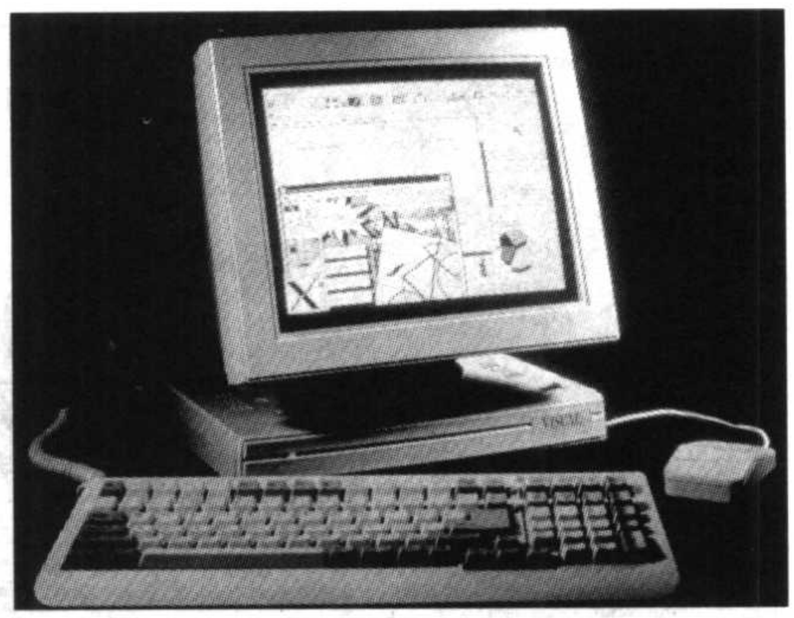


图 1.61 页面布局工作站（Visual Technology 公司提供）

和许多其他的计算机图形应用一样，商务艺术显示通常使用照相式逼真技术来绘制设计图、产品和场景的图片。图 1.62 给出了一个三维标牌设计的例子，图 1.63 给出了用于广告的三个计算机图像。

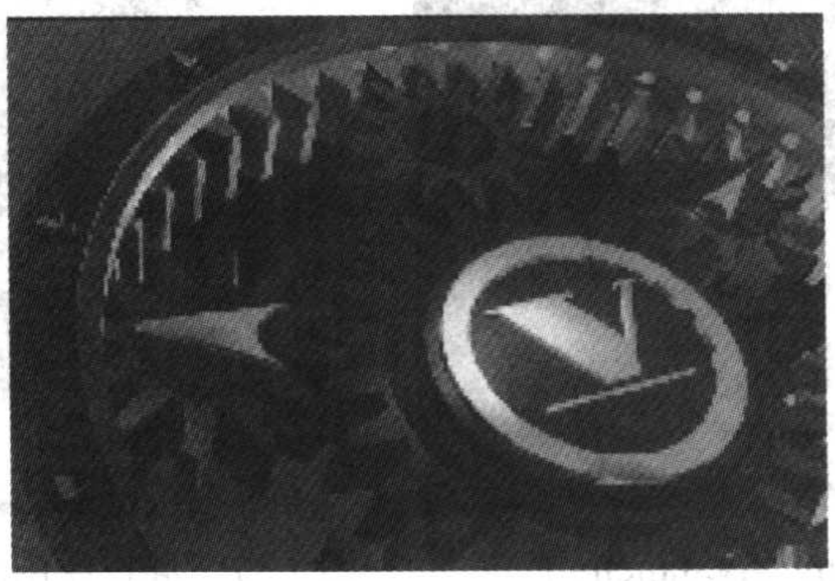
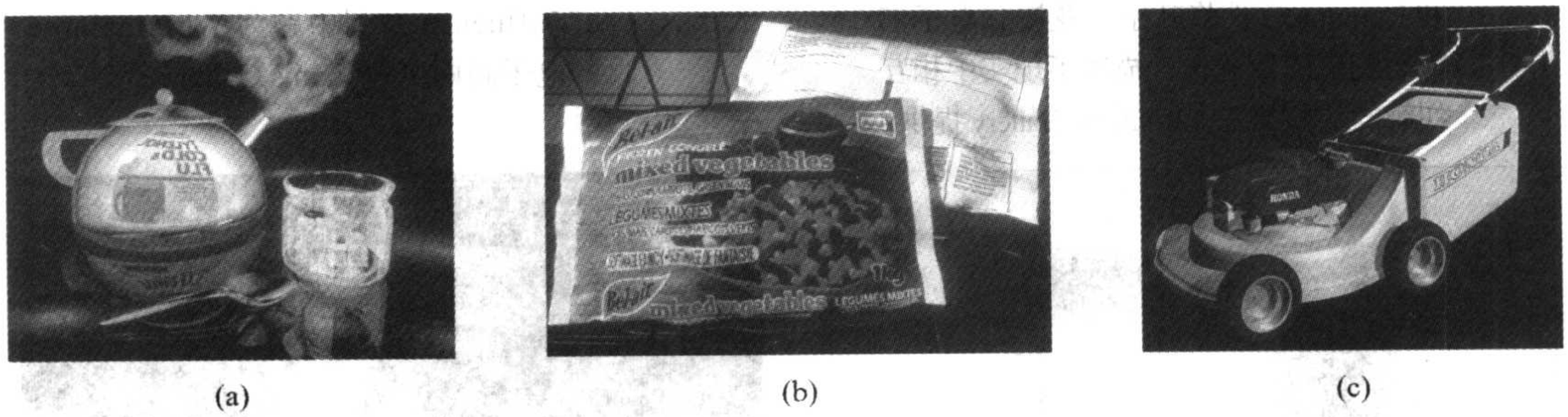


图 1.62 一个标牌的三维绘制（Vertigo Technology 公司提供）



(a) (b) (c)

图 1.63 三个用于广告的计算机图像（(a)由 Audrey Fleisher 提供；(b)和(c)由 SOFTIMAGE 提供）

电视商业片中也经常使用计算机动画，通过一帧一帧地绘制生成电视广告并存储为图像文件。通过将后继帧中的物体位置相对于前一帧进行微小的移动来实现对动画中运动的模拟。绘制好动画序列中的所有帧以后，将这些帧传送到胶片上或存储到视频缓存中以备重播。电影动画需要每分钟顺序播放 24 帧。如果在视频显示器上重播动画，则需每分钟 30 帧。

将一个物体转换为另一个的变形（morphing）技术是许多电视商业片中常用的一种图形方法。在电视商业片中使用该方法可以将一个油桶变为一个汽车引擎，将一辆汽车变成一只老虎，将一潭水变成一只轮胎，将一个人的脸变成另一个人的脸。变形的例子将在下一节给出（参见图 1.69）。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

1.7　娱乐

电视产品、动画片和音乐视频等频繁地使用计算机图形方法。有时将图形场景与演员及实际场景相混合，有的电影则完全由计算机绘制和动画技术生成。

许多电视剧经常使用计算机图形方法来产生特技效果，例如图 1.64 中给出的系列剧“Deep Space Nine”的场景。有些电视节目也使用动画技术将计算机生成的人、动物或卡通人物与真正的演员在场景中混合，或者将一个演员的脸变换成另外的形状。许多节目使用计算机图形学来生成大楼、地表特征或场景的背景等。图 1.65 是为日本广播电视机构制作的北京 13 世纪建筑的重建画面。

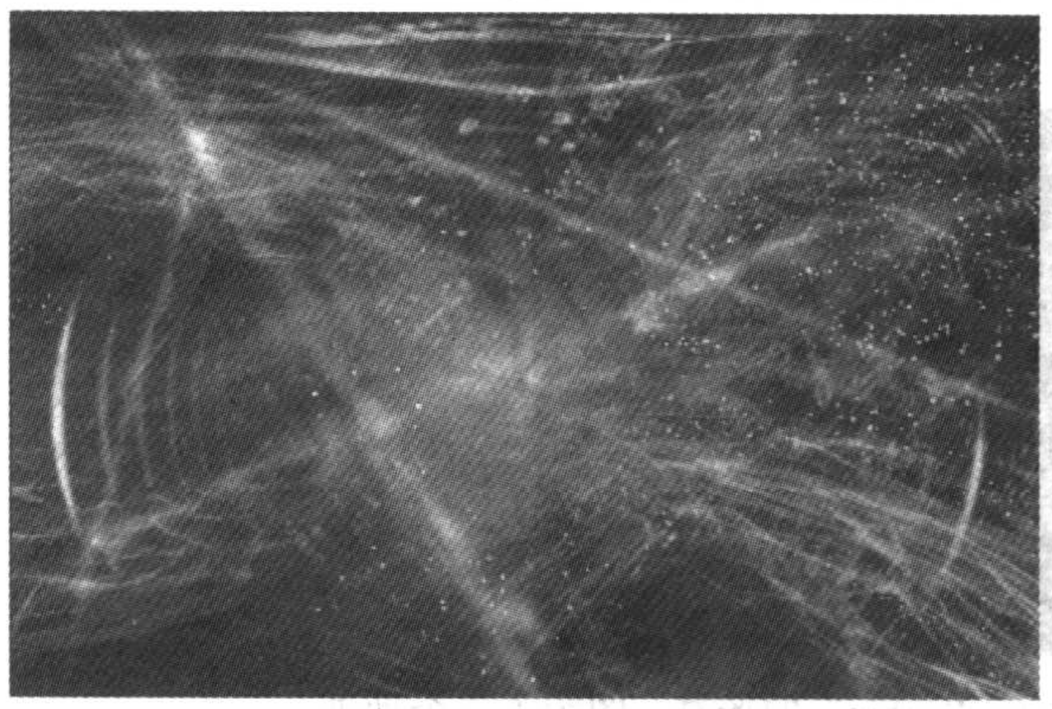


图 1.64　在电视系列剧“Deep Space Nine”中的图形场景（Rhythm & Hues Studios 提供）

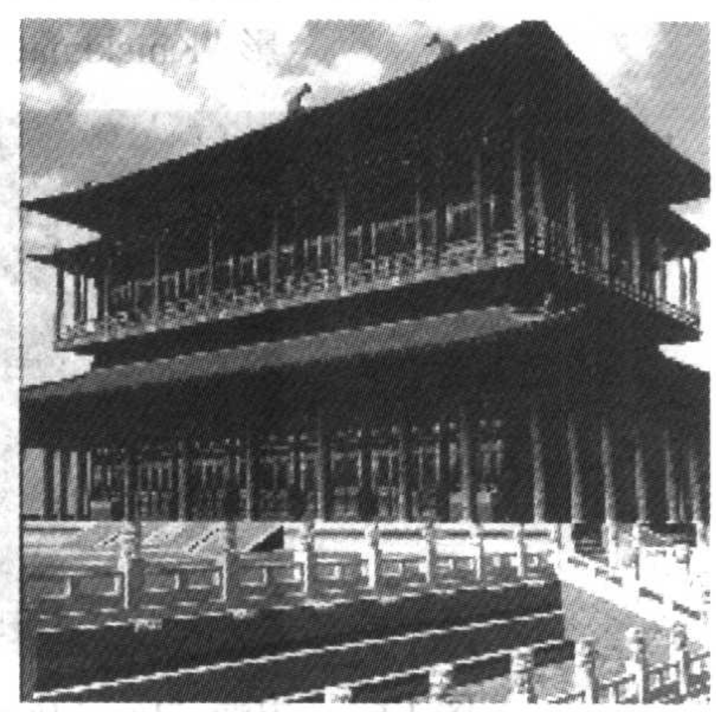


图 1.65　Taisei 公司制作、使用 TDI 软件绘制的 13 世纪的北京建筑（Thomson Digital Image 公司提供）

计算机生成的特技效果、动画、人物素描和场景被广泛地应用于当代电影中。图 1.66 给出了为电影“Star Trek —— The Wrath of Khan”生成的图形场景初稿。行星和宇宙飞船先用线框图绘出，然后再用绘制方法生成电影中这些对象的表面。图 1.67 给出了使用先进的建模和曲面绘制方法为两个获奖短片生成的场景。其他电影使用计算机建模、绘制和动画来生成完整的拟人化特征。照相级真实感技术为电影中计算机生成的演员提供肌肤色调、真实感的面部特征和皮肤缺陷（如痣、雀斑和粉刺）等。图 1.68 给出了电影“Final Fantasy: The Spirits Within”的场景，其中使用照相级真实感技术描绘了人造演员的外貌。

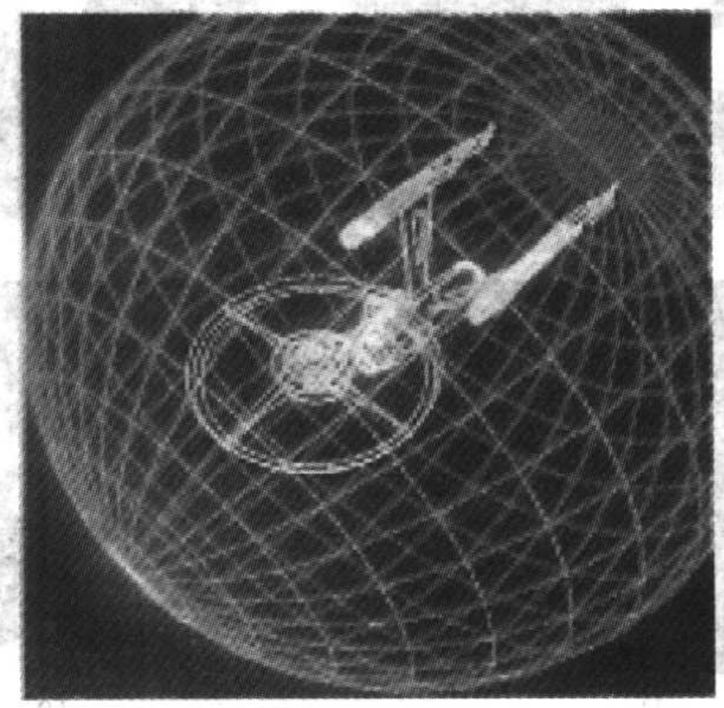
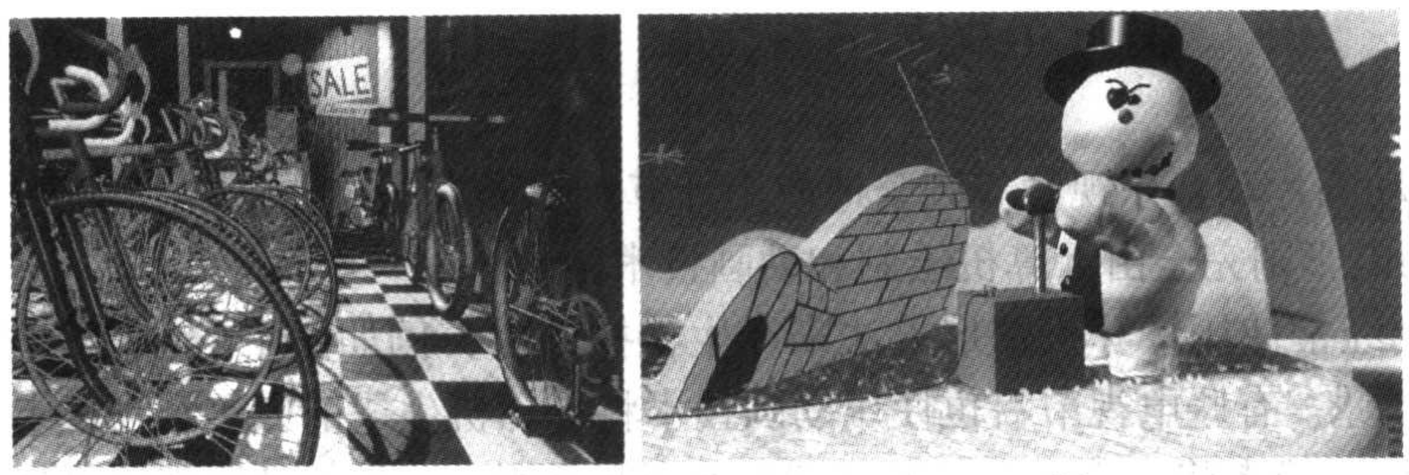


图 1.66　为 Paramount Pictures 的电影“Star Trek —— The Wrath of Khan”开发的图形初稿（Evans & Sutherland 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com



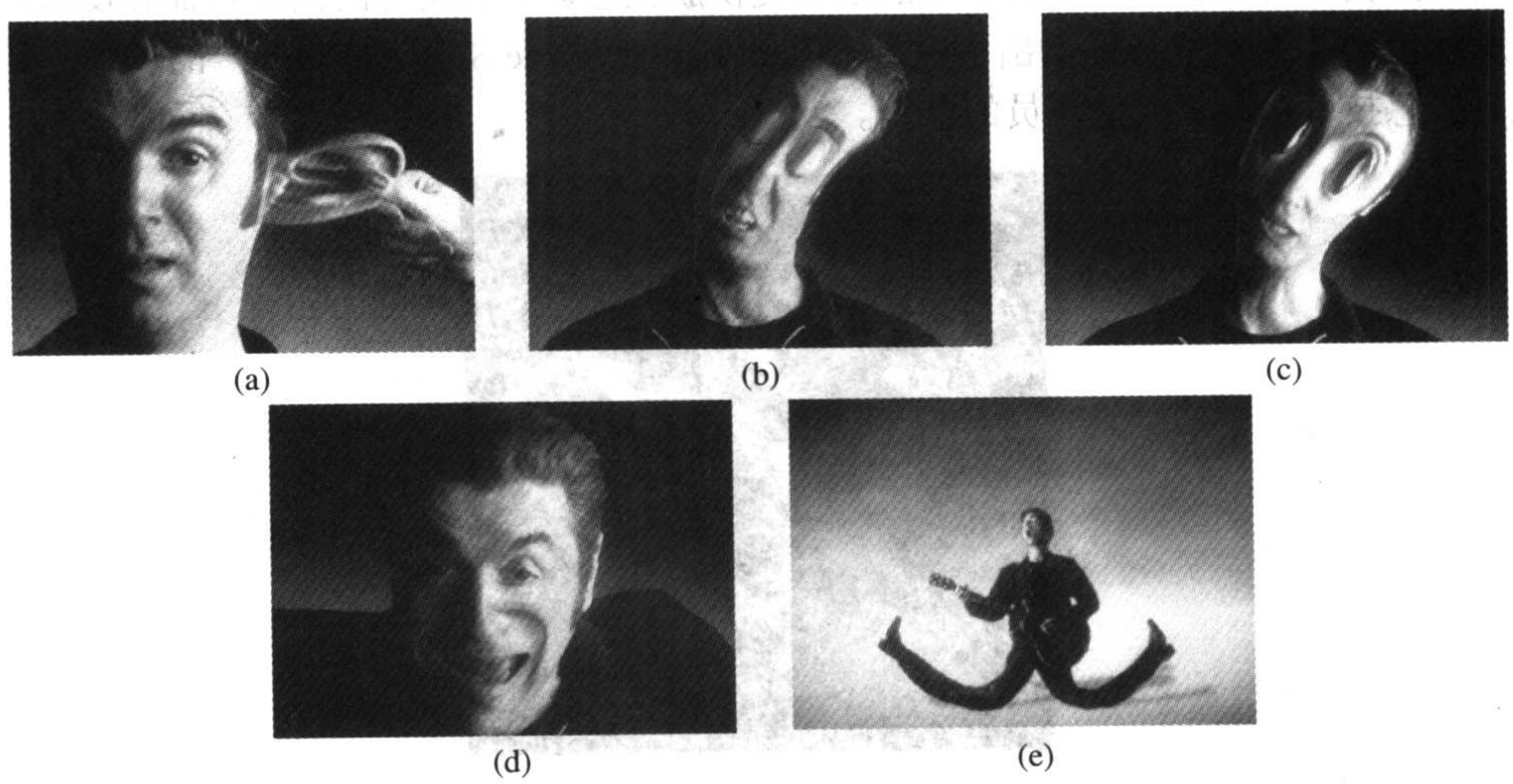
(a) (b)

图 1.67 (a)电影“Red's Dream”中一个计算机生成的场景；(b)电影“Knickknack”中一个计算机生成的场景（Pixar提供）

图 1.68 电影“Final Fantasy: The Spirits Within”的场景，图示了三个动画人物：Dr. Aki Ross、Gray Edwards 和 Dr. Sid（Square Pictures 提供）

计算机图形方法还可用来仿真真正的演员。使用记录演员脸部特征的数字文件，动画程序可生成包含这个人的计算机复制品的电影片段。当电影拍摄过程中遇到演员生病或出现事故时，这些仿真技术可用来在其后的电影场景中取代演员。

音乐录像片中按照多种不同的方式使用图形学。可以将图形对象混合进实景中，图形学和图像处理技术也可用来将一个人或对象变成另一个（变形）。图 1.69 给出了为 David Byrne 的录像片“She's Mad”所生成的一系列变形例子。



(a) (b) (c) (d) (e)

图 1.69 David Byrne的录像片“She's Mad”中的变形（David Byrne、Index Video 和 Pacific Data Images 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

1.8　图像处理

照片和电视扫描片等现有图片的修改或解释称为**图像处理**(image processing)。尽管在计算机图形学和图像处理中所使用的技术有所重叠，但两种领域着重于基本上不同的操作。在计算机图形学中，计算机用来生成图形；而图像处理技术用来改善图片质量、分析图像或为机器人应用识别可视图案。然而，图像处理技术经常应用于计算机图形学，计算机图形学方法也频繁应用于图像处理。

一般而言，照片或其他图片在使用图像处理方法之前先数字化成一个文件。然后使用数字方法重新安排图片的各部分、提高颜色分离度或改善着色质量。图1.70给出了应用图像处理技术增强图片质量的例子。这些技术被广泛地应用于商务艺术应用，包括对照片的某部分和其他美术作品进行调色或重新安排。类似的方法还用于分析地球的卫星照片或银河星系的望远镜记录。

医学上也广泛地将图像处理技术应用于图片增强、层析X射线造影术和外科手术模拟等方面。层析 X 射线造影术是一种 X 射线照片技术，它能将生理系统的剖面显示出来。计算机控制 X 射线断层造影术（ computed X-ray tomography，CT ）、定位发射造影术（ position emission tomography，PET ）和计算机轴向造影术（ computed axial tomography，CAT ）等均使用投影方法从数字数据中重建剖面。这些技术也用于在进行外科手术时监视身体的内部功能和显示剖面。其他医学图像技术包括超声波和核子医学扫描仪。超声波扫描仪使用高频声波代替X射线来产生数字数据。核子医学扫描仪从吸收的放射性核素的放射过程中收集数据并绘制彩色图像。

图像处理和计算机图形学在许多医学应用中常常结合在一起，用于对机体功能进行建模和研究、设计人造肢体以及计划和练习手术等。最后一种应用称为计算机辅助手术（ computer-aided surgery)。通过使用图像技术可以获得身体的二维剖面图，然后使用图形方法模拟实际的手术过程，从而观察和管理每一剖面，并试验不同的手术位置。图 1.71 和图 1.72 给出了医学应用的例子。

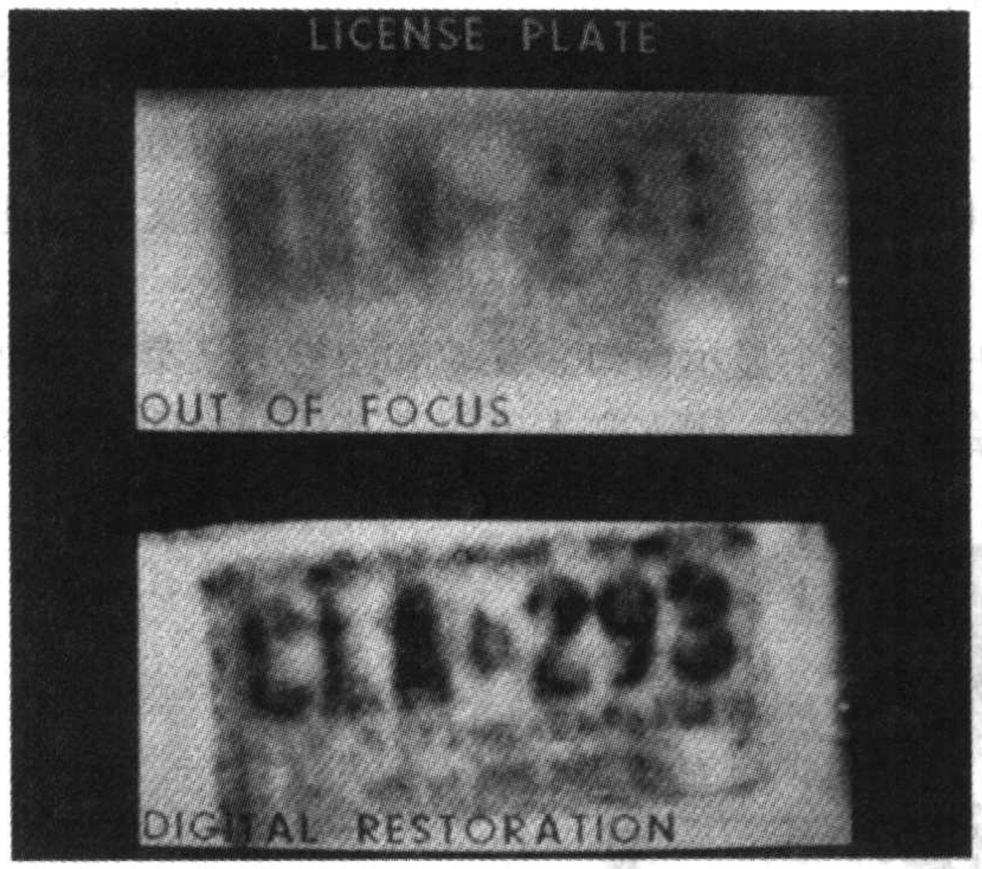


图 1.70　使用图像处理技术后使模糊不清的牌照照片变得清晰（ Los Alamos 美国国家实验室提供）

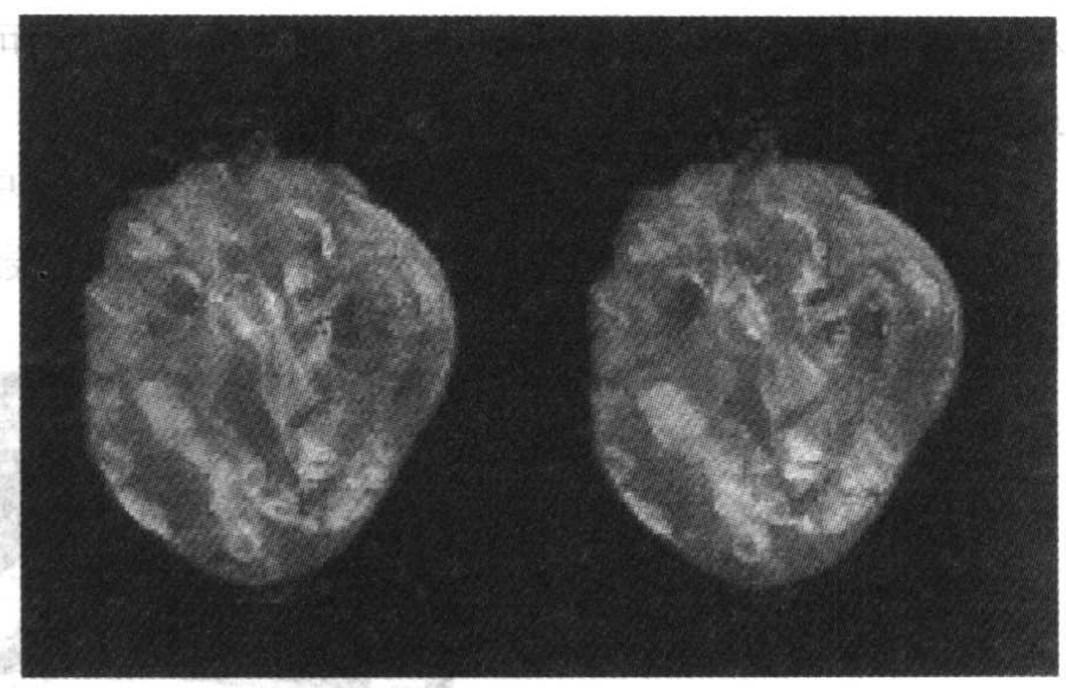


图 1.71　计算机动画中的一帧，在半透明体绘制区域中可视化狗心脏的活动层次。由 William Smith、Ed Simpson 和 G.Allan Johnson 提供医学数据，Cray Research 公司的 Tom Palmer 提供图像绘图软件（ Dave Bock 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

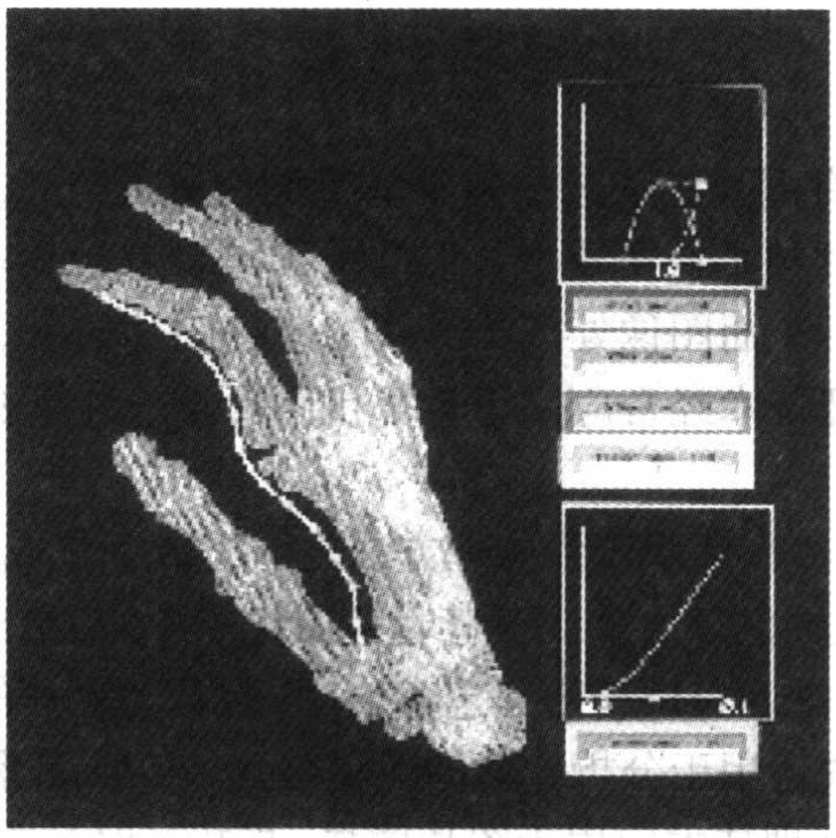


图 1.72 显示人手骨的一对立体图中的一幅图。由 Inmo Yoon、D. E. Thompson 和 W. N. Waggenspack 绘制，数据集是从 GWLNHDC 的 Rehabilitation 研究所的 CT 扫描中获得。这些图像为整形外科手术给出了一条可能的腱路径（路易斯安那州立大学提供）

1.9 图形用户界面

现在的应用软件提供**图形用户界面**（graphical user interface，GUI）是非常普遍的。图形用户界面的主要部分是一个允许用户显示多个矩形屏幕区域窗口的窗口管理程序。每一个屏幕显示区域可以进行不同的处理，展示图形或非图形信息，并且显示窗口可以用多种方法激活。可以通过使用鼠标之类的交互式点击设备将屏幕光标定位到某系统的显示窗口区域，并按下鼠标左键来激活该窗口。有的系统还可以通过点击标题条来激活显示窗口。

界面提供菜单和图标用于选择显示窗口、处理选项或参数值。图标是一个设计成能暗示所选对象的图形符号。图标的优点是它比相应的文本描述占用较少的屏幕空间，如果设计得好，可以很容易地理解。一个显示窗口与相应的图标表示可以相互转换，而菜单中可以包含一组文字描述或图标。

图 1.73 给出了一个典型的图形用户界面，其中包含多个显示窗口、菜单和图符。从菜单中可以选择不同的处理、颜色值和图形参数。图标代表绘画、缩放、键入文本串以及其他与构图有关的操作选项。

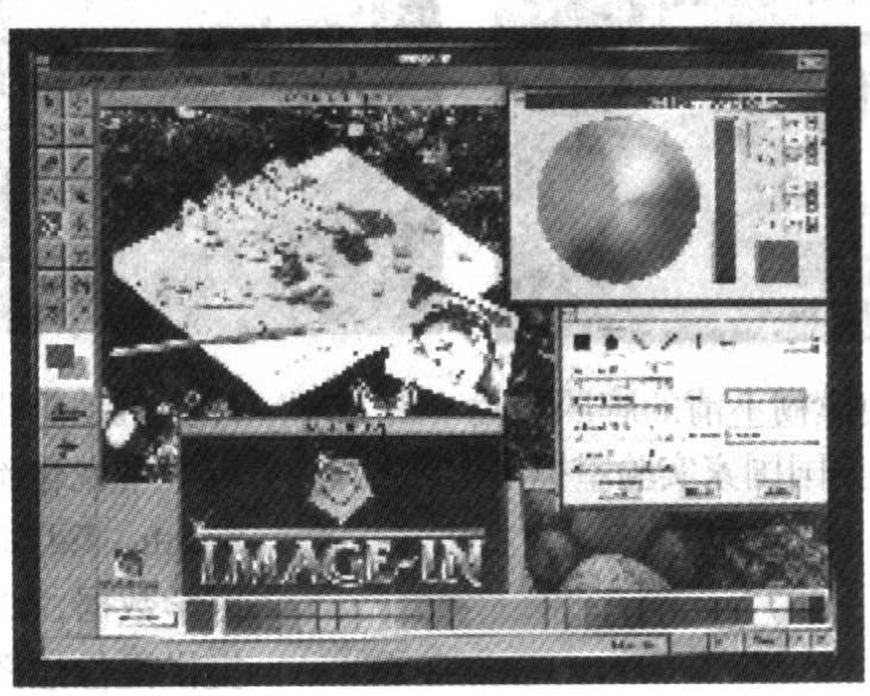


图 1.73 一个典型的包含多个显示窗口、菜单和图符的图形用户界面（Image-In 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

1.10　小结

本章综述了计算机图形学应用的许多领域，包括数值图、CAD、虚拟现实、科学计算可视化、教学、艺术、娱乐、图像处理和图形用户界面。但是还有许多其他领域没有提到，我们也无法将其他应用领域的例子都归入本书。后面几章将对本章介绍的应用及其他一些应用中使用的设备和方法进行探讨。

参考文献

图形方法在艺术、科学、数学和技术等方面的应用可以从Bouquet(1978)、Yessios(1979)、Gardner and Nelson(1983)、Grotch(1983)、Tufte(1983 and 1990)、Wolfram(1984)、Huitric and Nahas(1985)、Glassner(1989)和 Hearn and Baker(1979)等文献中找到。图形方法在音乐可视化中的应用请参见 Mitroo, Herman, and Baker(1979)。多个工业领域的计算机辅助设计和制造（CAD/CAM）在Pao (1984)中有详细讨论。飞行仿真中的图形技术请参见 Schachter(1983)。Fu and Rosenfeld(1984)讨论了视觉的模拟，Weinberg(1978)报道了宇宙飞船的仿真。图符和符号概念在Lodding(1983)和Loomis, et al.(1983)中给出。关于医学应用的更多信息请参考Hawrylyshyn, Tasker, and Organ(1977)、Preston, Fagan, Huang, and Pryor(1984)和 Rhodes, et al.(1983)。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第2章　图形系统概述

现在，计算机图形学的作用与应用已经得到了广泛承认，大量的图形硬件和软件系统已经应用到了几乎所有的领域。通用计算机甚至许多手持计算器也已经普遍具备二维及三维应用的图形功能。在个人计算机上也可以配用多种交互输入设备及图形软件包。对于高性能应用，可以选择许多高级的专用图形硬件系统和技术。本章将探讨图形硬件和图形软件包的基本特性。

2.1　视频显示设备

图形系统一般使用视频显示器作为其基本的输出设备（参见图2.1）。大部分视频监视器的操作是基于标准的**阴极射线管**（cathode-ray tube，CRT）设计的，但是也已经出现了其他一些技术，并且固态监视器终究要占据主导地位。

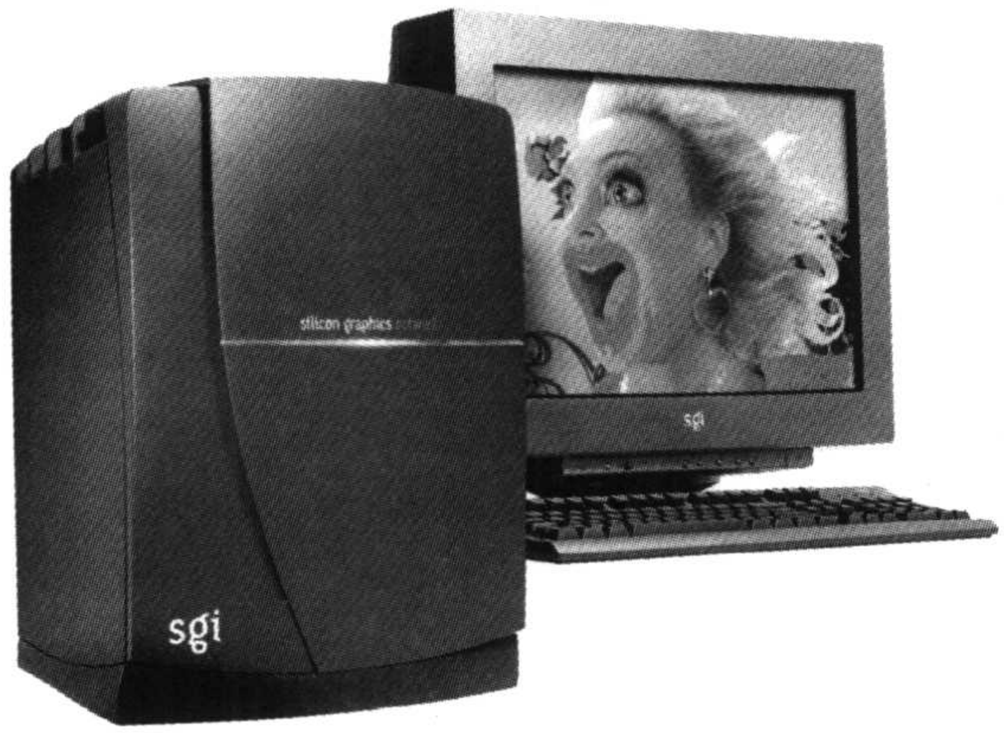


图 2.1　计算机图形工作站（SGI 公司提供）

2.1.1　刷新式 CRT

图2.2给出了CRT的基本工作原理。由电子枪发射出的电子束（阴极射线）通过聚焦系统和偏转系统，射向涂覆有荧光层的屏幕上的指定位置。在电子束轰击的每个位置，荧光层都会产生一个小亮点。由于荧光层发射的光会很快衰减，因此必须采用某种方法来保持屏幕图像。一种办法是将图形信息作为电荷分布存储在 CRT 上。这种电荷分布用来保持荧光粉处于激活状态。但现在使用较多的维持荧光粉亮度的办法是快速控制电子束反复重画图像。这类显示器称为**刷新式 CRT**（refresh CRT），在屏幕上重复画图的频率称为**刷新频率**（refresh rate）。

CRT电子枪的主要元件是受热激发的金属阴极和控制栅极（参见图2.3）。通过给称为灯丝的线圈通电来加热阴极，引起受热的电子“沸腾出”阴极表面。在 CRT 封装内的真空里，带负电荷的自由电子在较高的正电压的作用下加速冲向荧光屏。该加速电压可由 CRT 封装内靠近荧光屏处充以正电荷的金属涂层生成，或者采用加速阳极（参见图 2.3）。有时，电子枪结构中把加速阳极和聚焦系统放在同一部件中。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

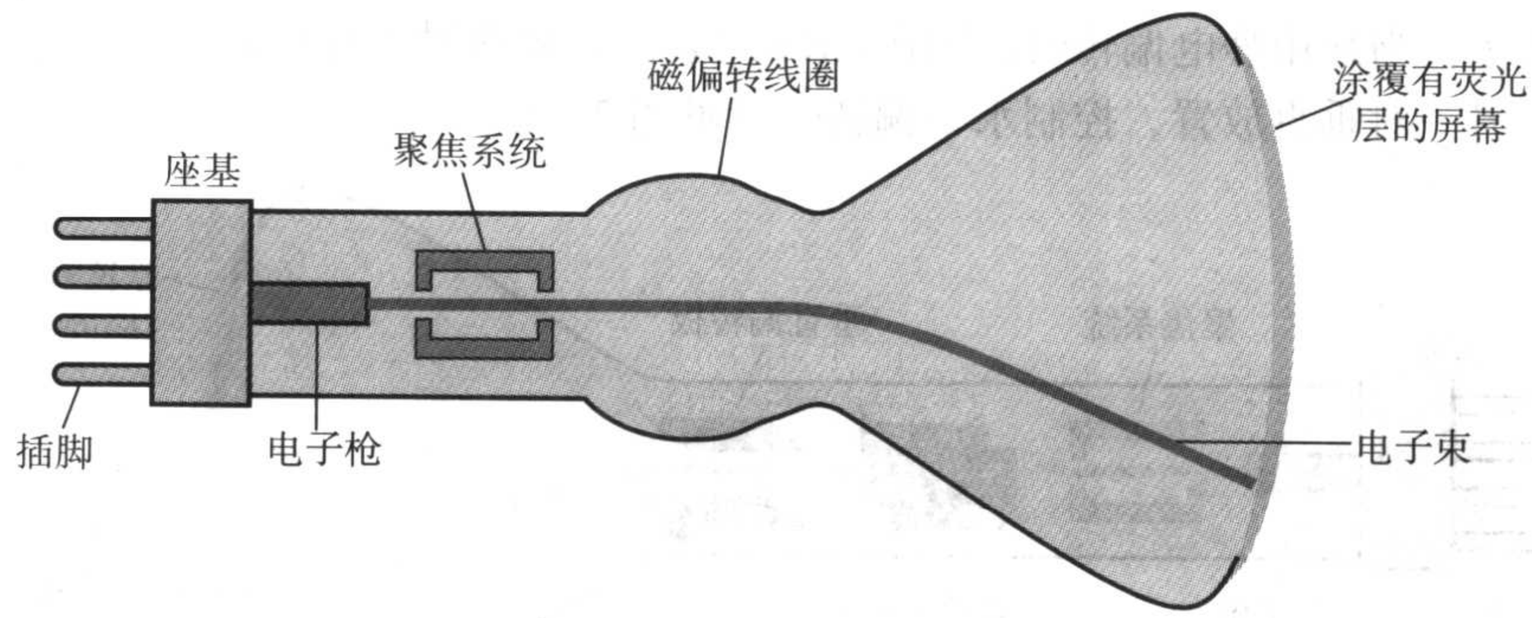


图 2.2　磁性偏转 CRT 的基本设计

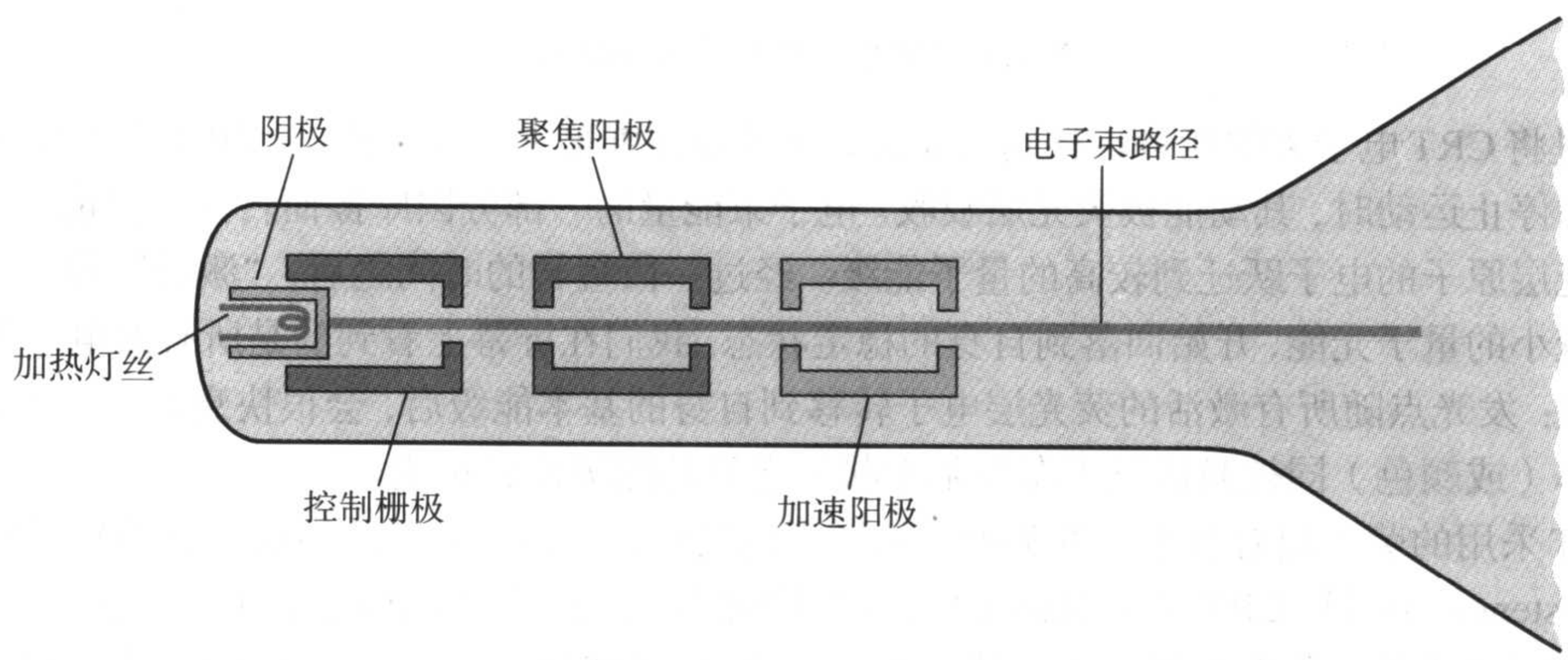


图 2.3　带加速阳极的电子枪工作原理

电子束的强度受设置在控制栅极上的电压电平控制。控制栅极是一个金属圆筒，紧挨着阴极安装。若在控制栅极上加上较高的负电压，则将阻止电子活动从而截断电子束，使之停止从控制栅极末端的小孔通过。而在控制栅极上施以较低的负电压，则仅仅减少了通过的电子数量。由于荧光层发射光的强度依赖于轰击屏幕的电子数量，因此可以通过改变控制栅极的电压来控制显示的光强。我们使用图形软件命令来设定各个屏幕位置的亮度级，这将在第 3 章进行讨论。

CRT 的聚焦系统用来控制电子束在轰击荧光层时会聚到一个小点。否则，由于电子互相排斥，电子束在靠近屏幕时会散开。聚焦既可以用电场实现，也可以用磁场实现。对于静电聚焦，电子束通过如图 2.3 所示的带正电荷的金属圆筒，该圆筒形成一个静电透镜。静电透镜的作用是使电子束聚焦在屏幕的中心，正如光学透镜将光束聚焦在指定的焦距一样。类似透镜的聚焦效果，可以由环绕CRT封装外部安装的线圈所形成的磁场来完成。磁性聚焦透镜能在屏幕上产生最小尺寸的亮点。

在高精度系统中，还使用附加的聚焦硬件，以保持电子束能聚焦到所有屏幕位置。因为多数CRT弯曲部分的直径大于从聚焦系统到屏幕中心的距离，所以电子束到屏幕不同点所经过的距离是不同的。因此，电子束只能在屏幕中心正确聚焦。当电子束移到屏幕边框时，所显示的图像会变得模糊。系统可按电子束的屏幕位置来调整聚焦，从而弥补这一缺陷。

电子束的偏转受电场或磁场控制。CRT现在通常配备一个装在CRT封装外部的磁性偏转线圈，如图 2.2 所示。使用两对线圈，将它们成对地安装在 CRT 封装的颈部，一对安装在颈部的顶部和底部，另一对设置在颈部两侧。每对线圈产生的磁场造成横向偏转力，该力正交于磁场方向，也垂直于电子束的行进方向。一对线圈实现水平偏转，另一对则实现垂直偏转。调节通过线圈的电流可得

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

到适当的偏转量。当采用静电偏转时，则在CRT封装内安装两对平行极板。一对为水平放置，控制垂直偏转；另一对垂直放置，控制水平偏转（参见图2.4）。

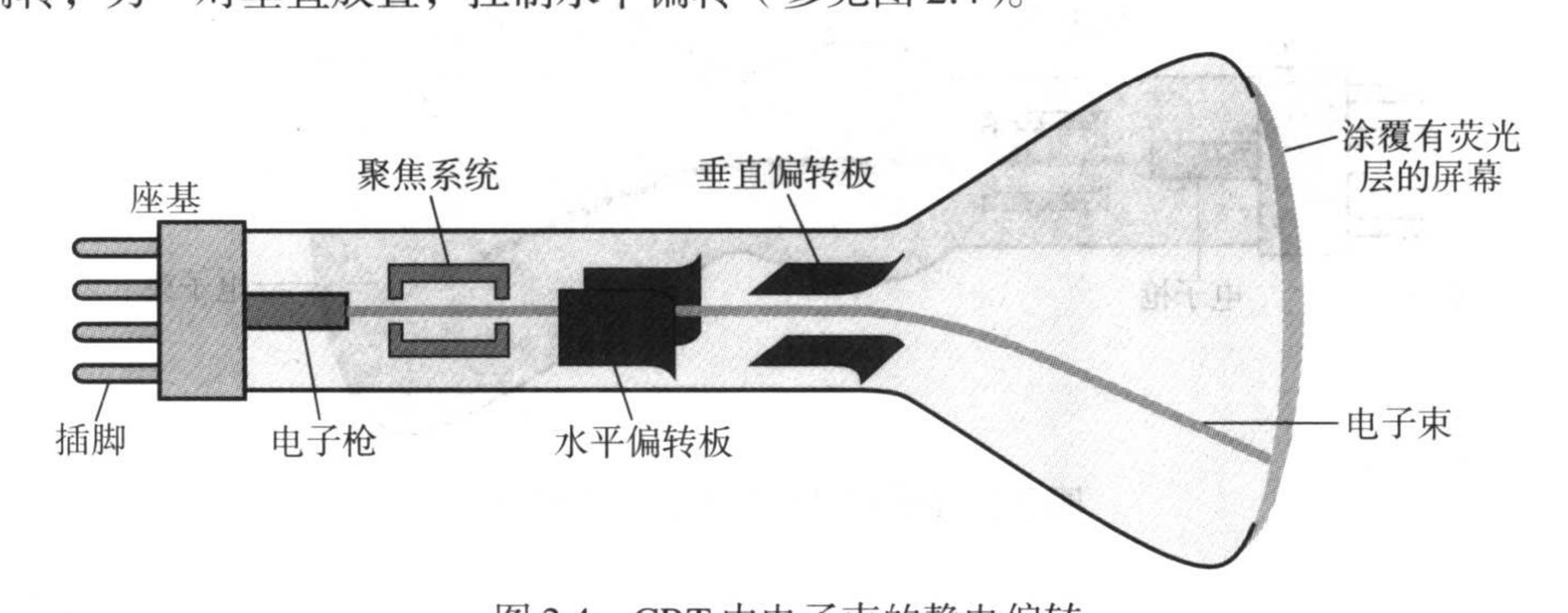


图2.4 CRT内电子束的静电偏转

通过将CRT电子束的能量转移到荧光层，就可以在屏幕上形成亮点。当电子束的电子撞击到荧光层并停止运动时，其动能被荧光层吸收。电子束能量的一部分因摩擦而转换为热能，余下部分导致荧光层原子的电子跃迁到较高的量子能级。经过一段短暂的时间之后，“激活”的荧光层电子释放了较小的量子光能，开始回落到自身的稳定状态。我们在屏幕上看到的是所有的电子光发射的组合效应：发光点随所有激活的荧光层电子转移到自身的基本能级后，会很快衰减。荧光层发射光线的频率（或颜色）同被激活量子态与基本状态之间的能级差成正比。

CRT采用的荧光层有着不同的类型。除了颜色之外，这些荧光层之间的主要差异是它们的**余辉**（persistence）时间：CRT电子束移走后，它们将继续发光（即激活电子转为基本态）多长时间。余辉时间定义成从屏幕发光到衰减为其原亮度十分之一的时间。较短余辉时间的荧光层，需要较高的刷新速率来保持屏幕图形不闪烁。短余辉的荧光层用于动画，而长余辉荧光层则用于显示高复杂度的静态图形。虽然有的荧光层的余辉时间大于一秒，但是对于图形监视器，通常采用余辉时间为10～60 μs的材料制成。

图2.5表明了屏幕上一亮点的亮度分布。亮点中心位置的亮度最大，并按高斯分布向亮点的边缘衰减。这个分布依赖于CRT电子束横截面的电子密度分布。

图2.5 CRT屏幕上荧光层亮点的亮度分布

CRT无重叠显示的最多点数称为**分辨率**（resolution）。虽然它常常简述为每个方向的总点数，但更精确的分辨率定义是在水平和垂直方向上每厘米可绘制的点数。亮点的强度满足高斯分布（参见图2.5），因此要使两个相邻亮点保持区别，其间隔应大于亮点强度在最大强度值的60%时的直径。这种覆盖位置如图2.6所示。亮点尺寸也依赖于亮度。当每秒有更多的电子加速飞向荧光层时，CRT电子束的直径及发光亮点的面积增大。此外，增大的激活能量趋向于传播到邻近的荧光原子，而不是正对着电子束的路径，这就进一步加大了亮点直径。因此，CRT的分辨率取决于荧光层的类型、显示的亮度、聚焦系统及偏转系统。典型的高质量系统分辨率为1280 × 1024，在许多系统中还要用到更高的分辨率。高分辨率系统常常称为高清晰度系统（high-definition system）。图形监视器的物理尺寸是由屏幕对角线的长度给定的。可从12英寸（1英寸＝2.54厘米）到27英寸或更大一些。CRT监视器可与各类计算机系统相连接，因此可实际绘制的屏幕点数依赖于与它相连接的系统的能力。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

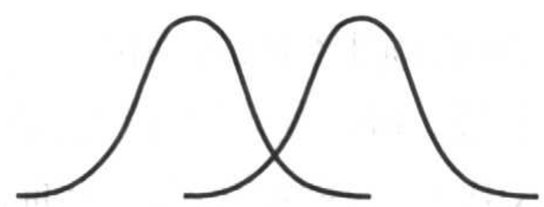


图 2.6　当两个荧光层亮点的间隔大于亮点亮度衰减到 60% 最大值的亮点直径时，两个亮点是可区分的

2.1.2　光栅扫描显示器

使用 CRT 的普通图形监视器是基于电视技术的**光栅扫描显示器**（raster-scan display）。在光栅扫描系统中，电子束横向扫描屏幕，一次一行，从顶到底依次进行。每一行称为一个**扫描行**（scan line）。当电子束横向沿每一行移动时，电子束的强度不断变化，从而建立亮点组成的一个图案。图形定义保存在称为**刷新缓存**（refresh buffer）或**帧缓存**（frame buffer）的存储器中，这里的**帧**（frame）是指整个屏幕范围。该存储器保存一组对应屏幕所有点的强度值。电子束在屏幕上逐点移动时由从刷新缓存取出的强度值控制其强度。这样，如图 2.7 所示，在屏幕上“画图”是每次一行。每个可由电子束点亮的屏幕点称为一个**像素**（pixel 或 pel，是 picture element 的简写）。由于刷新缓存用来存储屏幕颜色值，因此它也称为**颜色缓存**（color buffer）。除了颜色以外，像素的其他信息也存储在缓存中，因而不同的缓存区域有时统称为“帧缓存”。光栅扫描系统对于屏幕的每一点都有存储强度信息的能力，从而使之较好地适用于包含细微阴影和彩色模式的场景的逼真显示。家用电视和打印机是另一类使用光栅扫描方式的例子。

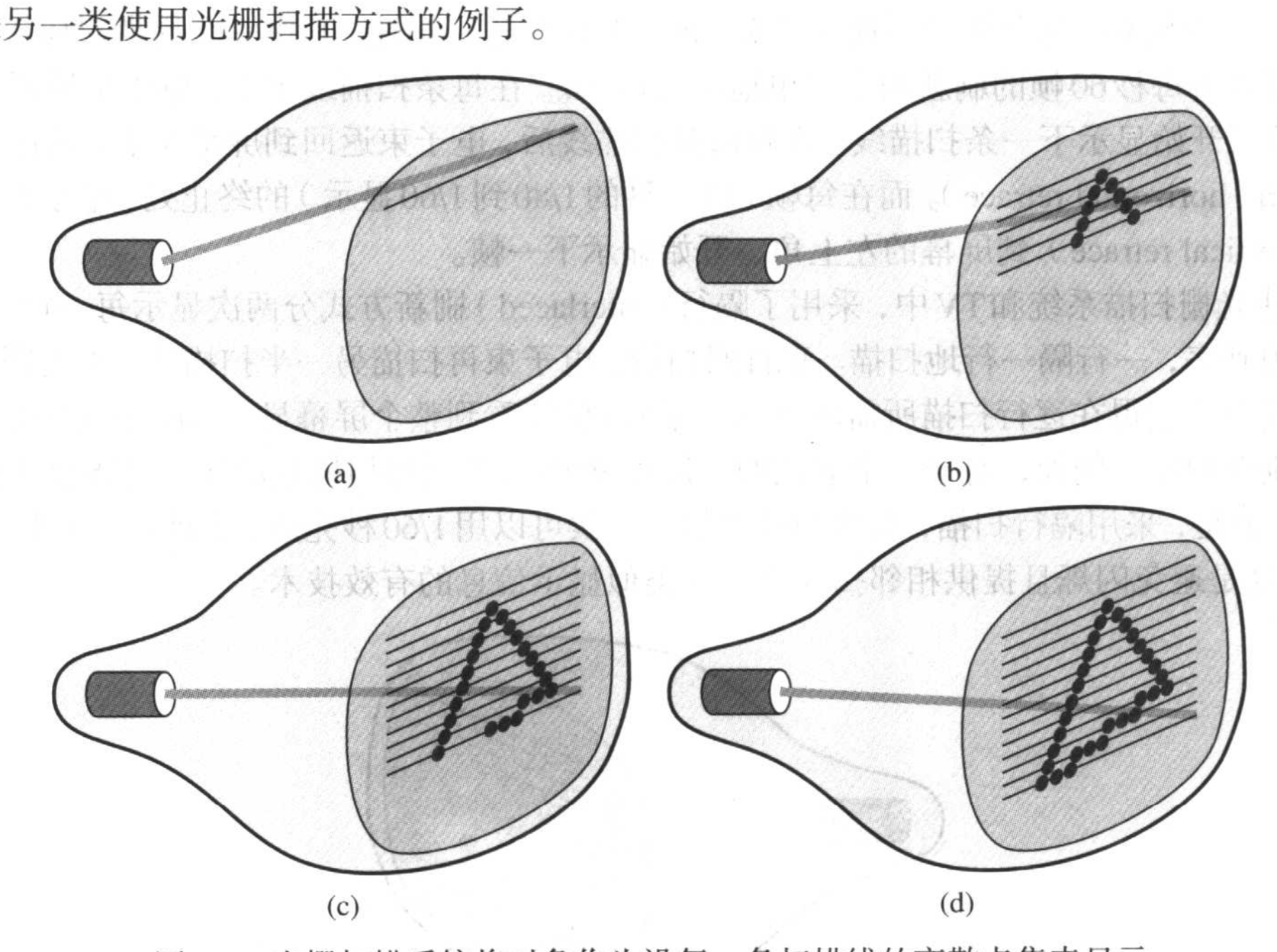


图 2.7　光栅扫描系统将对象作为沿每一条扫描线的离散点集来显示

光栅系统常用称为分辨率的像素个数作为其特征。视频显示器的另一特征是**纵横比**（aspect ratio），定义为系统能显示的像素列数除以行数的结果（有时术语纵横比用来表示扫描行数除以像素列数的结果）。纵横比还可用在屏幕上显示水平和垂直方向相同长度线段所需的点数之比来描述。因此，纵横比为 4/3 表示用 4 点绘出的水平线与用 3 点绘出的垂直线的物理长度相同。类似地，任一矩形（包括整个屏幕）的纵横比可用其宽度除以高度所得的结果来描述。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

光栅系统可显示的颜色或灰度等级依赖于CRT使用的荧光粉类型以及每一像素对应的帧缓存中的位数。对于一个简单的黑白系统来说，每一屏幕点或亮或暗，因此每个像素只需一位来控制屏幕位置上的亮度。该位取值为1，表示电子束在该位置时开通，取值为0表示电子束在该位置时关闭。如果要使电子束除了“开”、“关”两状态之外有更多的强度等级，那么就需要提供附加位。在高性能系统中每一像素可多达24位，这时分辨率为1024 × 1024的屏幕要使用3 MB容量的刷新缓存。每像素的位数有时也称为缓存**深度**（depth）或**位平面**（bit plane）数。同样，每像素一位的帧缓存通常称为**位图**（bitmap），而每像素多位的帧缓存称为**像素图**（pixmap）。

当刷新频率不太低时，我们会感觉到刷新过程中相邻两帧的内容是平稳过渡的。在每秒24帧以下时，我们会感觉到屏幕上相邻图像之间有间隙，即图像出现闪烁。例如，早期的无声电影以每秒16帧的速率拍摄，因而放映时有闪烁现象。在20世纪20年代开发有声系统时，电影速率已增加到每秒24帧，因而消除了闪烁及演员的不稳定运动。早期的光栅计算机系统按每次刷新30帧来设计，因而产生了较好的结果，但由于监视器上的显示技术与电影有着根本上的不同，因此图片质量的改善还依赖于监视器更高的刷新频率。电影放映机可以通过持续放映一帧直到下一帧开始放映来保持显示结果的连续性。但是在视频监视器上，荧光点在点亮后立即开始衰退。因此，现在多数扫描显示器使用每秒60 ~ 80帧的刷新频率，部分系统达到每秒120帧的刷新频率。有些图形系统设计成使用可变刷新频率。例如，为立体显示应用选择高刷新频率，使其在交替显示场景的两个视图时不会闪烁。这一类应用通常使用多个帧缓存的方法。

有时，刷新频率以每秒多少周期或赫兹（Hz）为单位进行描述，其中一个周期对应于一帧。因此，我们可以将每秒60帧的刷新频率简单地称为60 Hz。在每条扫描线末端，电子束返回到屏幕的左边，然后又开始显示下一条扫描线。刷新每条扫描线后，电子束返回到屏幕左端，这称为电子束的**水平回扫**（horizontal retrace）。而在每帧（以一秒的1/80到1/60显示）的终止处，电子束返回（**垂直回扫**，vertical retrace）到屏幕的左上角，开始显示下一帧。

在某些光栅扫描系统和TV中，采用了隔行（interlaced）刷新方式分两次显示每一帧。第一次，电子束从顶到底，一行隔一行地扫描。垂直回扫后，电子束再扫描另一半扫描线（参见图2.8）。这种隔行扫描方式使得在逐行扫描所需时间的一半时，就能看到整个屏幕显示。隔行扫描技术主要用于较慢的刷新频率。例如，对于一个老式的、每秒30帧的非隔行扫描显示器，可能会注意到它产生了闪烁。但是，采用隔行扫描，两次扫描中的每一次可以用1/60秒完成，也就是刷新频率接近每秒60帧。这是避免闪烁且提供相邻扫描线包含类似显示信息的有效技术。

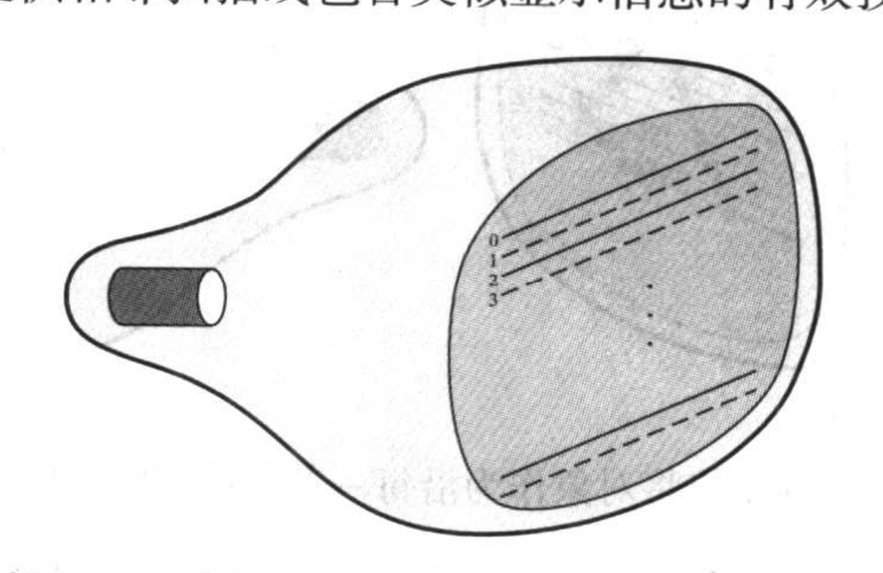


图 2.8 光栅扫描显示的隔行扫描线，首先显示偶数（实线）扫描线上所有的点，然后显示奇数扫描线（虚线）的所有点

2.1.3 随机扫描显示器

当CRT用于**随机扫描显示器**（random-scan display）时，其电子束只在屏幕上显示图形的部分移动。电子束逐条地跟踪图形的组成线条，从而生成线条图。因此，随机扫描显示器也称为**向量显**

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

示器（vector display）、**笔划显示器**（stroke-writing display）或**笔迹显示器**（calligraphic display）。图形的组成线条由随机扫描系统按任意指定的顺序绘制并刷新（参见图2.9）。笔式绘图仪也以类似的方式工作，它是随机扫描、硬拷贝设备的一个例子。

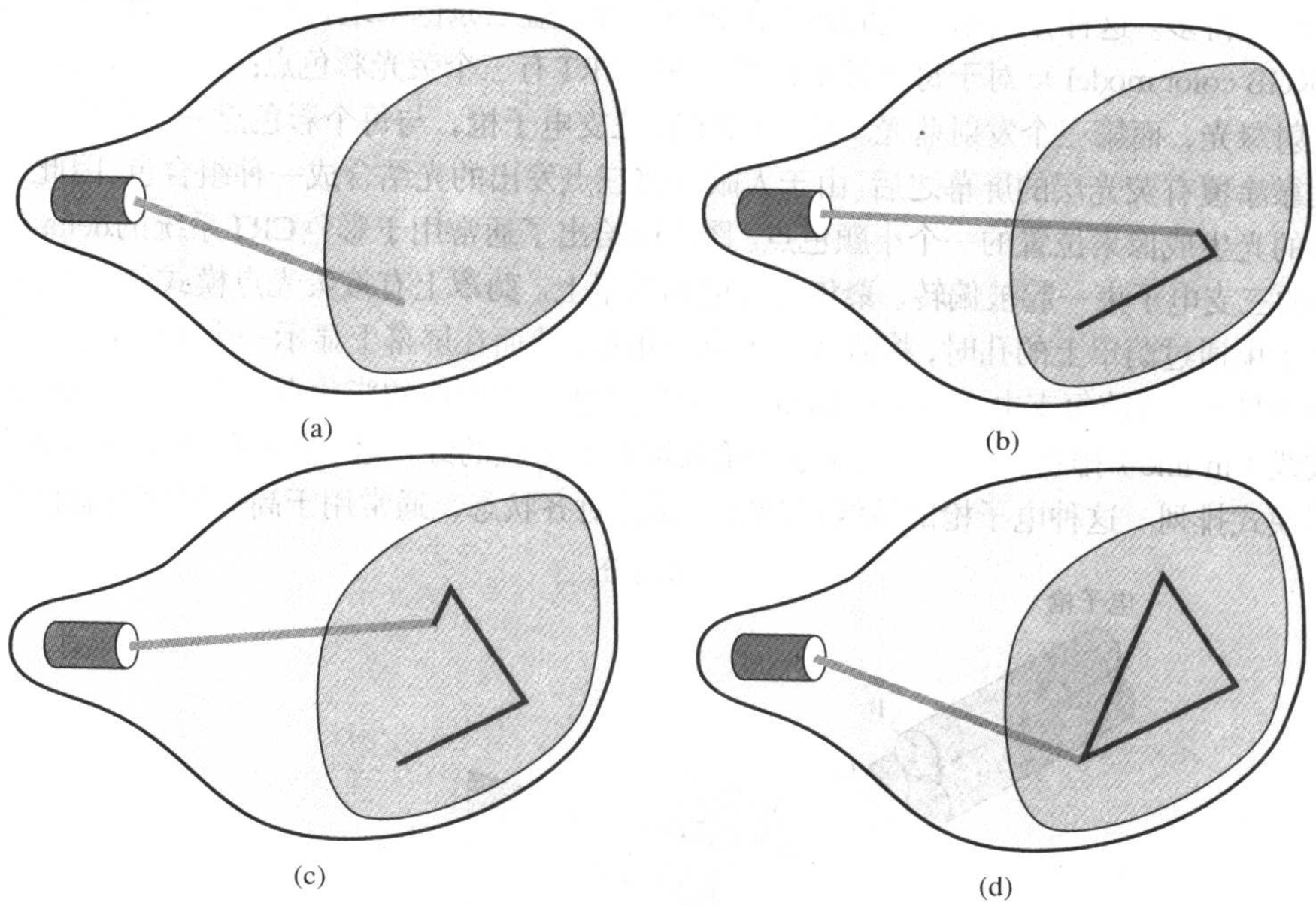


图 2.9　随机扫描系统以任意指定的顺序画出对象的组成线条

随机扫描系统的刷新频率依赖于显示的线数。这时图形的定义是存放在称为刷新显示文件存储区的一组画线命令。刷新显示文件称为**显示列表**（display list）、**刷新显示文件**（refresh display file）、**向量文件**（vector file）或**显示程序**（display program）。为了显示指定的图形，系统周期地按显示文件中的一组命令依次画出其组成线条。当所有画线命令处理完后，系统周期地返回到该列表的第一条画线命令。随机扫描显示器设计成每秒 30 ~ 60 次画出图形的所有线条。高性能的向量系统在这样的刷新速率中能处理约 100 000 条短线。当显示的线条很少时，则延迟每个刷新周期，以避免刷新速率超过每秒 60 帧。否则，线条的刷新过快，可能会烧坏荧光层。

随机扫描系统用于画线应用，如建筑和工程布局图等，它不能显示逼真的有阴影的场景。由于图形定义是作为一组画线命令来存储而非所有屏幕点的强度值，所以向量显示器一般具有比光栅系统更高的分辨率。另外，向量显示器的 CRT 电子束直接按线条路径画线，因而生成光滑线条。相比之下，光栅系统通过显示一组离散点来画线，因而生成锯齿状线条。但是，光栅系统极大的灵活性和提高的画线能力还是淘汰了向量技术。

2.1.4　彩色 CRT 监视器

CRT 监视器利用能发射不同颜色光的荧光层的组合来显示彩色图形。不同荧光层的发射光组合起来，可以生成一种按其比例而定的可见颜色。

显示彩色图形的一种方法是在屏幕上涂上多层不同的荧光粉。发射颜色由电子束在荧光层中的穿透深度决定。这种方法称为**电子束穿透法**（beam-penetration），它常用于红、绿两层结构。速度慢的电子束只激活外面的红色层，速度快的电子束能穿过红色层并激活里面的绿色层。而中速的电子束通过发射红、绿光的组合来生成两种另外的颜色：橙色和黄色。电子的速度，也就是屏幕上任

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

意一点的颜色，受电子束的加速电压控制。电子束穿透法是随机扫描监视器生成彩色图形的廉价途径，但是只可能有较少的颜色种类，而且图形质量不如其他方法的好。

荫罩法（shadow-mask）常用于光栅扫描系统（包括彩色电视机），因为它能产生的彩色范围比电子束穿透法大得多。这种方法基于我们熟悉的由红、绿、蓝三原色来组合颜色的原理，称为**RGB颜色模型**（RGB color model）。对于每个像素位置，荫罩CRT有三个荧光彩色点：一个荧光点发射红光，另一个发射绿光，而第三个发射蓝光。这类CRT有三支电子枪，与每个彩色点一一对应，而荫罩栅格位于紧靠涂覆有荧光层的屏幕之后。由于人眼可将三点发出的光结合成一种组合色，因此三种荧光粉发射出的光生成像素位置的一个小颜色点。图2.10给出了通常用于彩色CRT系统的delta-delta荫罩法。其中的三支电子束一起被偏转、聚焦并发射到荫罩上。荫罩上有按荧光点模式分布的一系列孔。当三支电子束通过荫罩上的孔时，将激活一个点三角形，从而在屏幕上显示一个小的彩色亮点。荧光点以三角形排列，并使每支电子束通过荫罩时，只能激活与之对应的彩色点。三支电子枪的另一配置结构是按线（in-line）排列。其中，三支电子枪及屏幕上相应的红－绿－蓝彩色点都沿扫描线而不是成三角形模式排列。这种电子枪的按线排列容易保持对齐状态，通常用于高分辨率的彩色CRT。

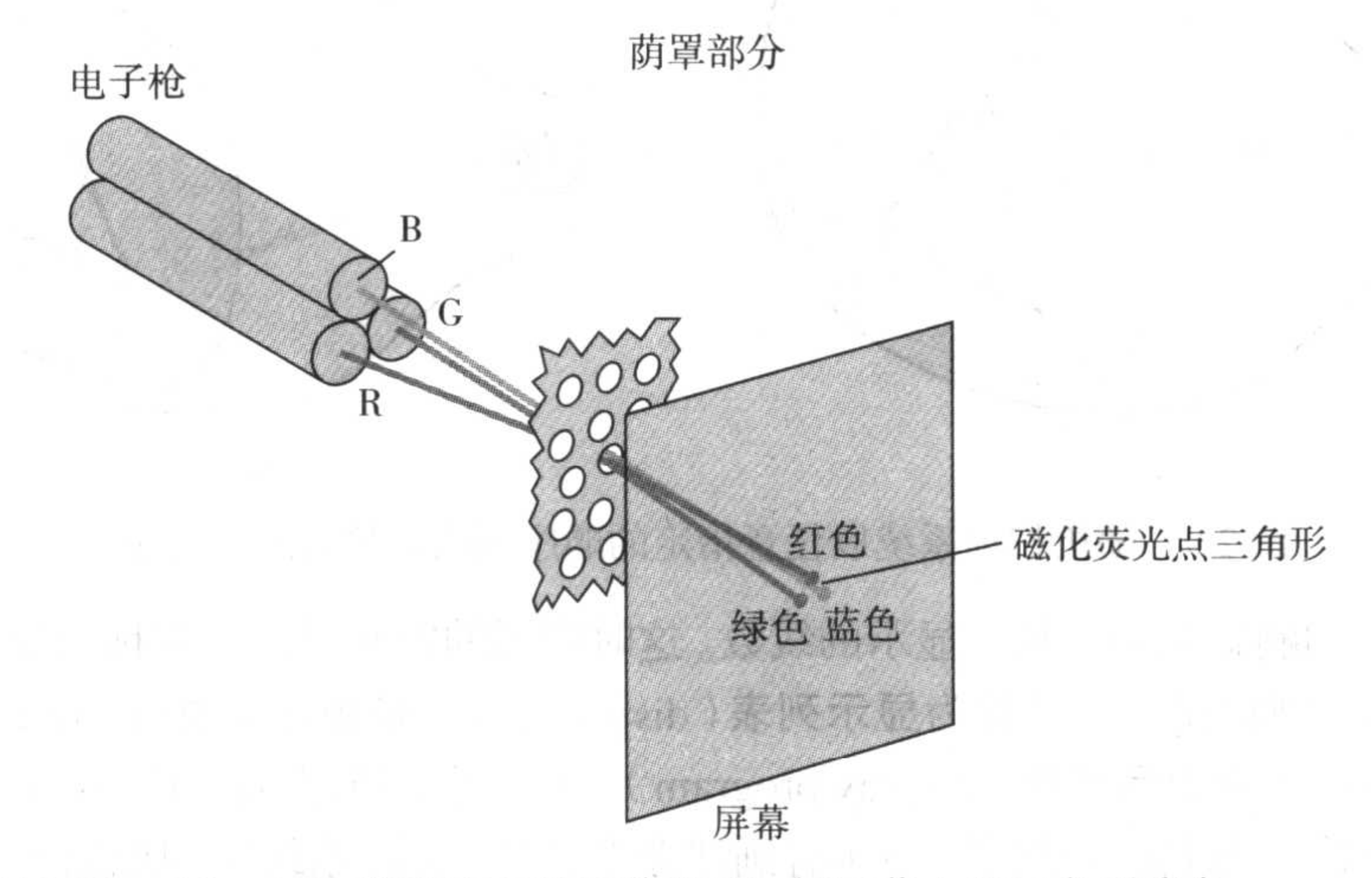


图 2.10　delta-delta荫罩CRT的工作原理。与屏幕上的三角形彩色点模式对应的三支电子枪由荫罩控制指向每个点三角形

改变三支电子束的强度等级，可以改变荫罩CRT显示的颜色。关掉三支枪中的两支，我们只能得到来自单个激活荧光点的颜色（红、绿、蓝）。在以相同的电子束强度激活三点时，我们将看到白色。黄色由相同强度的绿点和红点产生，品红由相同强度的蓝点和红点产生。而当蓝点和绿点的激活程度相同时，将呈现青色。在某些低价格系统中，电子束只能置为开或关，因此只能显示八种颜色。较高级的系统可以为电子束设置中间强度等级，这样就允许生成几百万种不同的颜色。

彩色图形系统可以根据配用多种CRT显示设备进行设计。某些廉价的家用计算机系统和电视游戏机则设计成能配用彩色电视机和RF（radio-frequency，无线电频率）调制器。RF调制器的作用是模拟广播电视台的信号。这意味着必须组合图形的颜色和亮度信息，并叠加到广播频率载波信号上来作为电视机的输入。然后，电视机中的电路从RF调制器接收这种信号，抽取图形信息，并在屏幕上进行显示。正如我们可以预料的，由于RF调制器和TV电路对图形信息的额外处理，将会降低图像显示的质量。

合成式监视器（composite monitor）是用于允许广播电路旁路的电视适配设备。这些显示设备仍然要求组合图形信息，但无需载波信号。它将图形信息组合为合成信号，然后由监视器分离，所得图形的质量仍然不是太好。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

人们将图形系统的彩色CRT设计成**RGB监视器**（RGB monitor）。这些监视器采用荫罩法且不经任何中间处理，直接从计算机系统取得每支电子枪（红、绿和蓝）的强度等级。在高质量的光栅图形系统的帧缓存中，每个像素对应24位，每支电子枪允许256级电压设置，因而每个像素有近1700万种颜色可供选择。每个像素具有24个存储位的RGB彩色系统通常称为**全彩色系统**（full-color system）或**真彩色系统**（true-color system）。

2.1.5　平板显示器

虽然多数图形监视器还是采用CRT构造，但是其他一些技术可能很快要替代CRT监视器。**平板显示器**（flat-panel display）代表一类相比CRT能减小体积、减轻重量并节省功耗的视频设备。平板显示器的一个有意义的特性是比CRT要薄，可以把它们挂在墙上或戴在手腕上。有些平板显示器上甚至还可以进行书写，因此它们可用于袖珍式笔记本。平板显示器另外还用于小型TV监视器、计算器、袖珍式视频游戏机、膝上型计算机、航空座椅上的电影屏幕、电梯内的告示牌，以及在要求不高的便携式监视器的应用场合中作为图形显示器。

我们可以把平板显示器分为两类：**发射显示器**（emissive display）和**非发射显示器**（nonemissive display）。发射显示器是将电能转换为光能的设备。等离子体显示板、薄膜光电显示器以及发光二极管都是发射显示器的实例。平板CRT也已发明出来，其中的电子束以平行于屏幕的方向加速，然后偏转90°轰击屏幕。但是，还未证实平板CRT同其他发射设备一样可以实际应用。非发射显示器利用光学效应将太阳光或来自某些其他光源的光转换为图形模式。液晶设备是非发射平板显示器的最重要的例子。

等离子体显示板（plasma panel）也称**气体放电显示器**（gas-discharge display），通过将通常包含氖气的混合气体充入两块玻璃板之间的区域而构成。一块玻璃板上放置一系列垂直导电带，而另一块玻璃板上构造一组水平导电带（参见图2.11）。在成对的水平和垂直导电带上施加点火电压，导致两导电带交叉点处的气体进入电子和离子的辉光放电等离子区。图形的定义存储在刷新缓存中，点火电压以每秒60次的速率刷新像素位置（导电带的交叉处）。使用交变电流方法快速提供点火电压，可以得到较亮的显示。像素之间的分隔是由导电带的电场提供的。图2.12给出一高分辨率等离子体显示板。等离子体显示板有一个缺点，即它是一种严格的单色设备，但现在已开发出能显示彩色和灰度等级的等离子显示器。

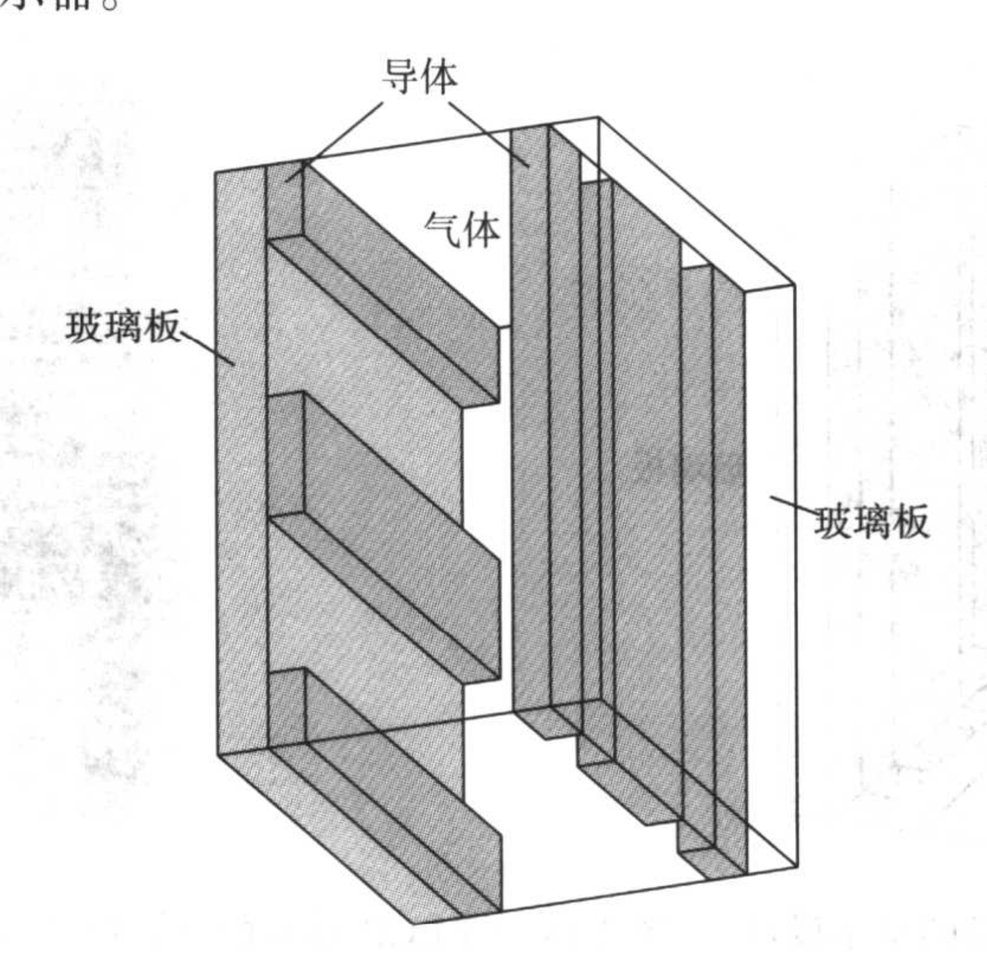


图 2.11　等离子体平板显示设备的基本设计

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

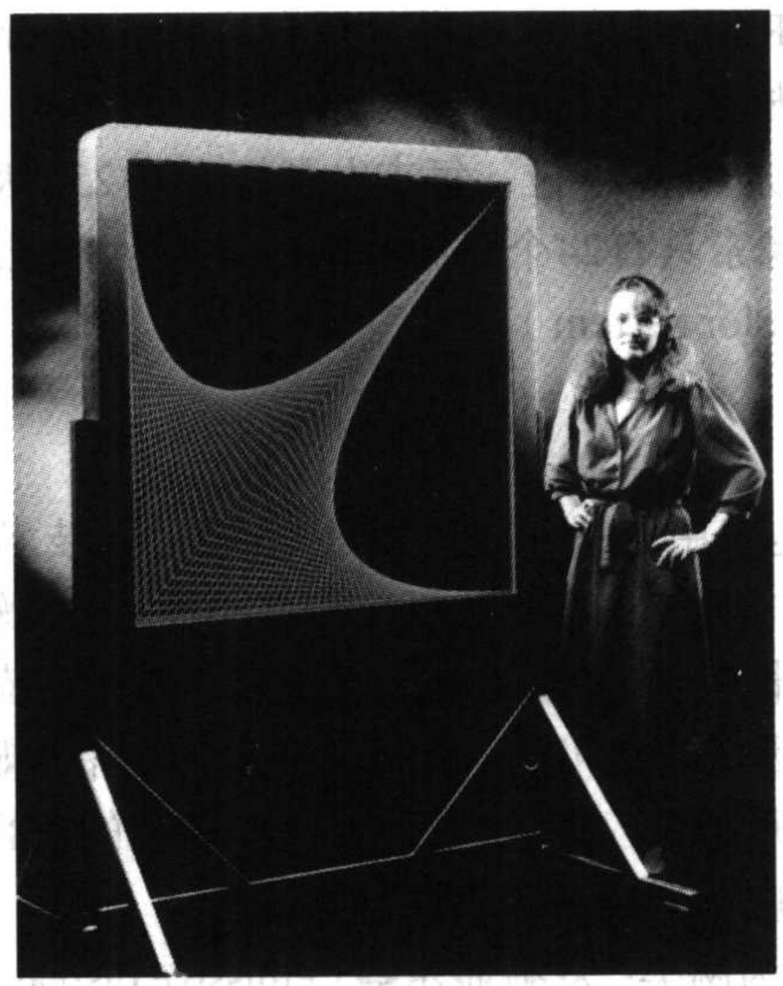


图 2.12　等离子体显示器，分辨率为 2048 × 2048，屏幕对角线为 1.5 米（Photonics Systems 提供）

薄膜光电显示器（thin-film electroluminescent display）具有与等离子体显示板类似的结构。不同之处是它在玻璃板之间的区域充以荧光物，诸如硫化锌与锰的胶状物，而不再是气体（参见图2.13）。当一个足够高的电压加到一对交叉的电极时，荧光层在两电极交叉区域成为一个导电体。电能由锰原子吸收，然后释放能量成为一发光亮点，这类似于等离子体显示板的辉光放电的等离子体效应。光电显示器比等离子体显示板需要更多的功耗，而且难以达到好的颜色和灰度等级显示。

第三类发射设备是**发光二极管**（light-emitting diode，LED）。二极管以矩阵排列形成显示器的像素位置，图形的定义存储在刷新缓存中。如同CRT的扫描线刷新一样，信息从刷新缓存读出，并转换为电压电平，然后应用于二极管，在显示器上产生发光图案。

液晶显示器（liquid-crystal display，LCD）通常用于小型系统，如膝上型计算机及计算器（参见图 2.14）。这些非发射设备生成图形的原理是，通过能阻塞或传递光的液晶材料，传递来自周围的或内部光源的偏振光。

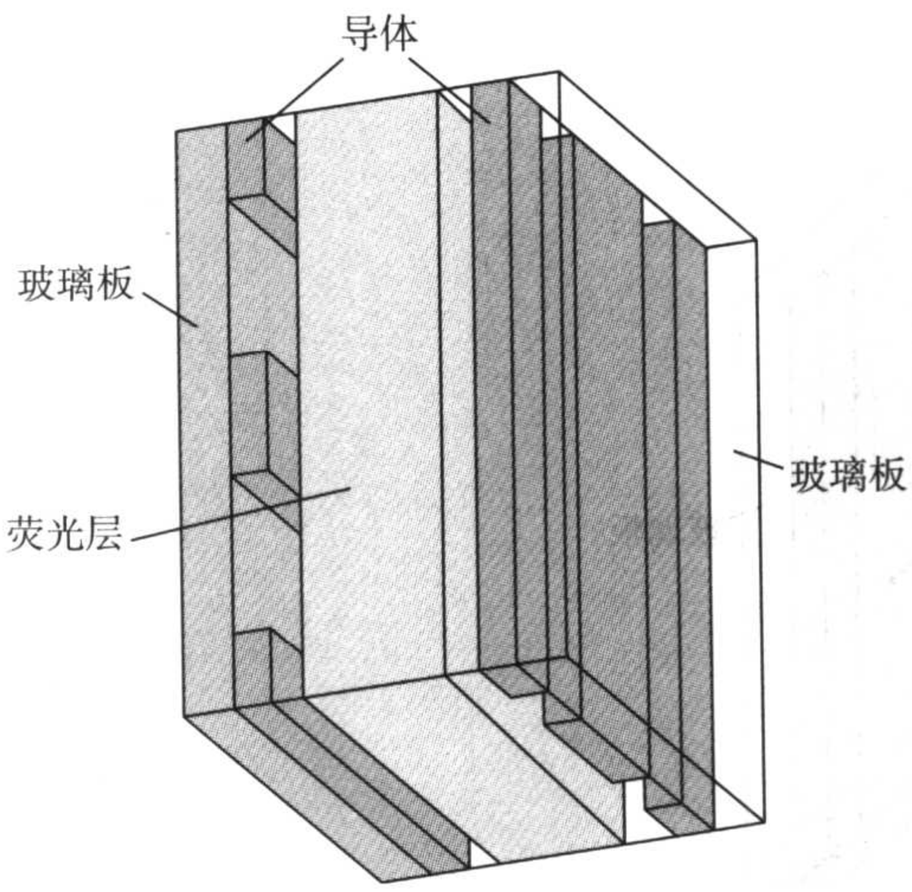


图 2.13　薄膜光电显示设备的基本设计

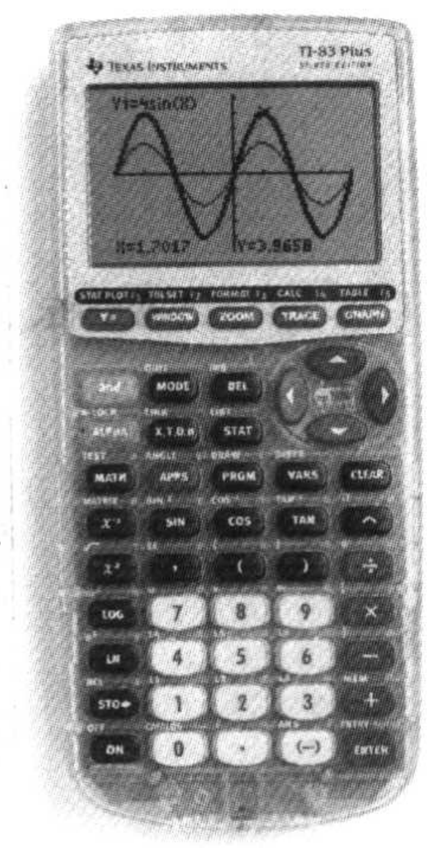


图 2.14　带 LCD 屏幕的掌式计算器（Texas Instruments 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

术语液晶（liquid crystal）是指这些化合物具有晶状结构的分子，并且可以像液体那样流动。平板显示器通常使用线状的液晶化合物，它们趋向于保持杆状分子的长轴排列。因此平板显示器可用线状的液晶构成，如图2.15所示。其中有两块玻璃板，每块都有一个光偏振器，与另一块形成合适的角度，内部充以液晶材料。在一块板上排放水平透明导体行，而另一块板上则放置垂直透明导体列。行、列导体的交叉处定义一个像素位置。通常，分子按图2.15中所示的“开态”排列。经过该材料的偏振光被扭曲，使之通过对面的偏振器，从而将光反射给观察者。如果要关掉像素，我们可以将电压置于两交叉导体，使分子对齐，从而不再扭曲偏振光。这类平板显示设备可视为**无源矩阵**（passive-matrix）LCD。图形的定义存储在刷新缓存，以每秒60帧的速率刷新屏幕，与发射设备相同。使用固态电子设备时，通常也利用背光，因而系统不完全依赖于外部光源。可以使用不同的材料或染料来显示颜色，并在每个屏幕位置放置一个三合一的彩色像素。构成LCD的另一种办法是在每个像素位置放置一个晶体管，并采用薄膜晶体管技术。晶体管用来控制像素位置的电压，并阻止液晶单元慢性漏电。这些设备称为**激活矩阵**（active-matrix）显示器。

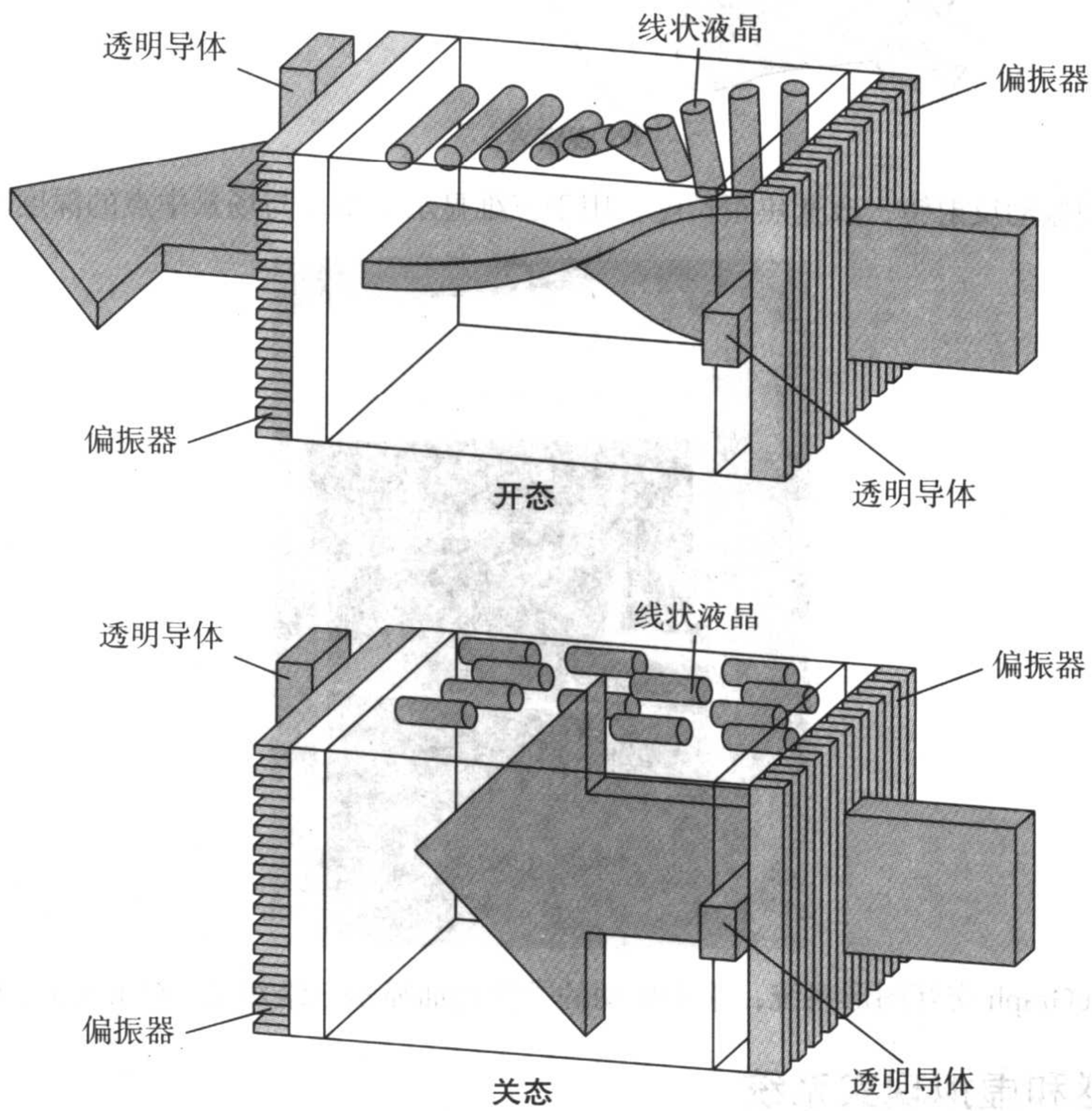


图 2.15　多数液晶显示设备使用的光扭曲、快门效应

2.1.6　三维观察设备

显示三维场景的图形监视器的设计，采用了从振动的柔性镜面反射CRT图像的技术。此类系统的操作原理如图2.16所示。当变焦反射镜振动时改变焦距长度。这些振动是同CRT上对象的显示同步的。因此，将该对象上的每一点从镜面反射到空间位置，对应于该点到指定观察位置的距离。这样就允许我们围绕着一个对象或场景行走，并从不同的角度进行观察。

图2.17给出了Genisco SpaceGraph系统，它采用振动镜将三维对象投影进25 cm × 25 cm × 25 cm空间。这个系统也能显示选定对象在不同深度横截面的二维“切片”。这些系统已用于医学应用，分

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

析来自超声波造影和CAT扫描设备的数据；在地质应用中，可以分析地形的地震数据；以及用于分子系统和地形系统的三维仿真应用。

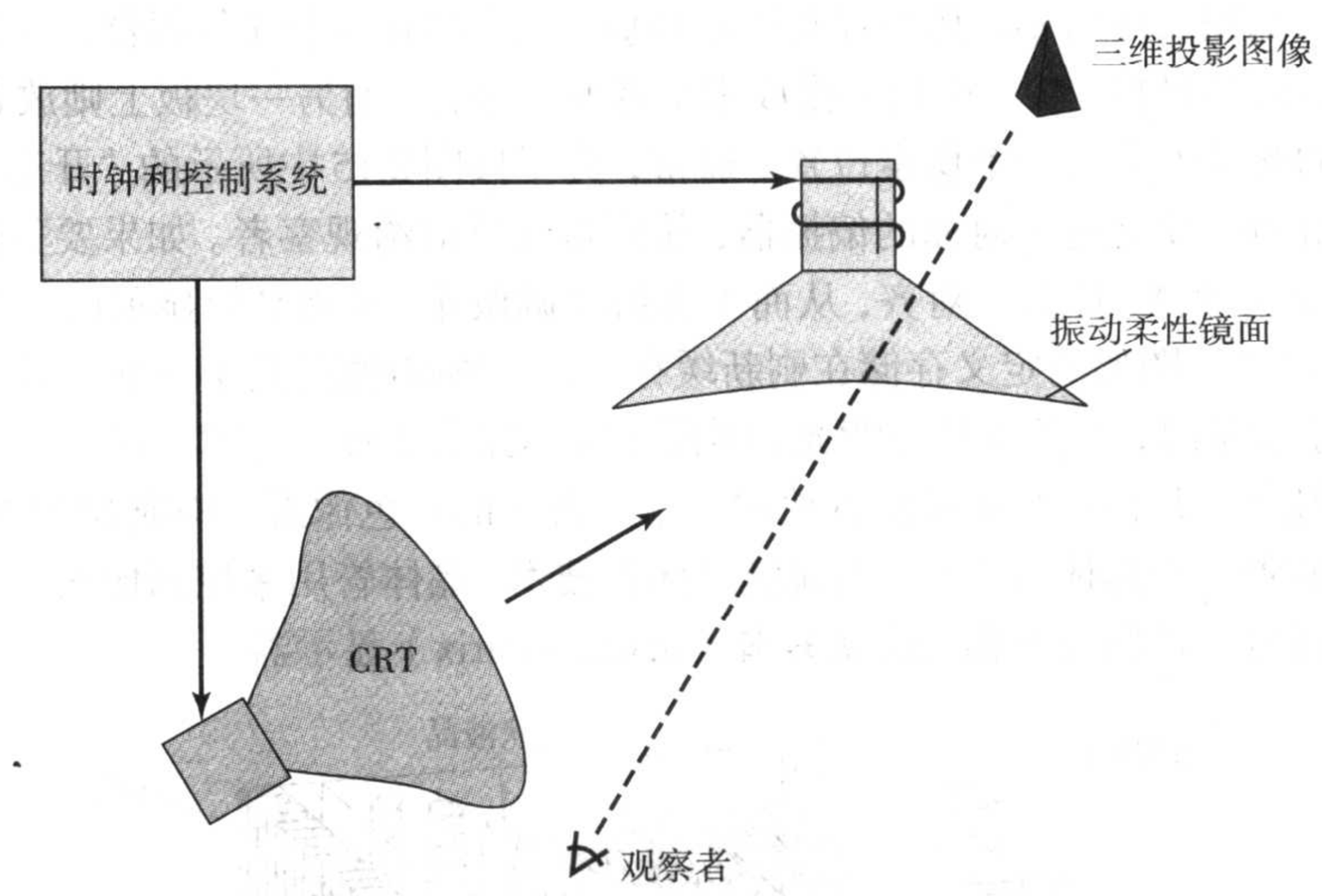


图 2.16　采用振动反射镜，改变焦距长度，用于三维显示系统（与场景中点的深度匹配）的操作

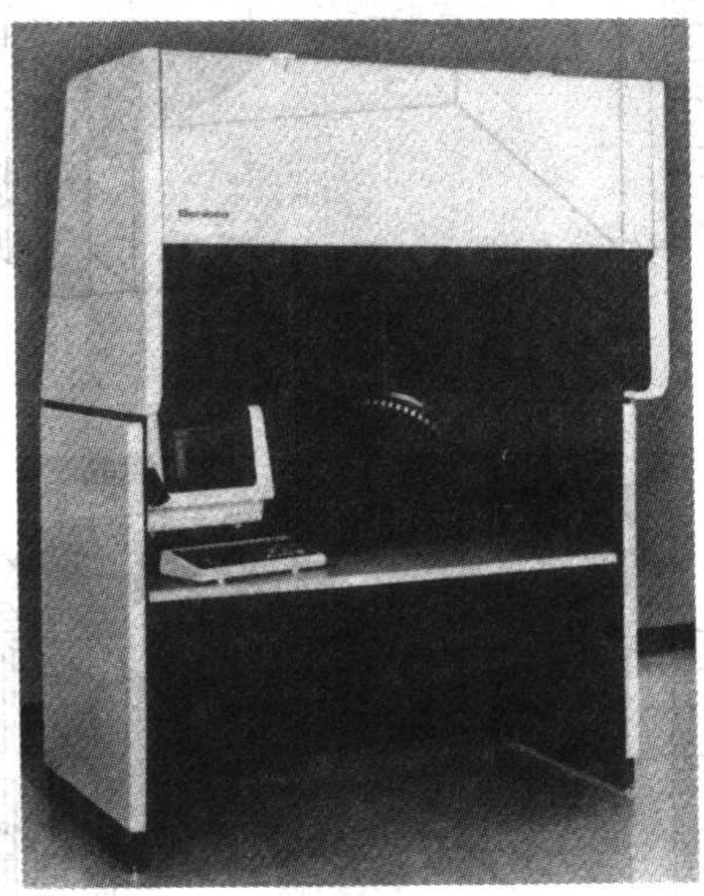


图 2.17　SpaceGraph 交互图形系统，采用振动的柔性镜面显示三维对象（Genisco 计算机公司提供）

2.1.7　立体感和虚拟现实系统

表示三维对象的另一种技术是显示具有立体感的视图。这种方法并不生成真实的三维图像，而是为观察者的每只眼睛给出不同的视图来提供三维效果，从而使场景带有深度（参见图 2.18）。

为得到具有立体感的投影，首先需要得到从相对于每只眼睛（左眼与右眼）的观察方向上产生的有关场景的两个视图。可以通过指定不同的观察位置，并由计算机生成场景来获得这两个视图，或者用一对立体照相机拍摄某些对象或场景来获得这两个视图。当我们同时用左眼得到左视图、右眼得到右视图时，则两个视图合成为单个图像，并感觉到场景带有深度。图2.19给出计算机生成的场景的两个立体投影视图。为了增加观察的舒适感，取消了该场景中仅由一只眼睛看到的左边和右边的区域。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

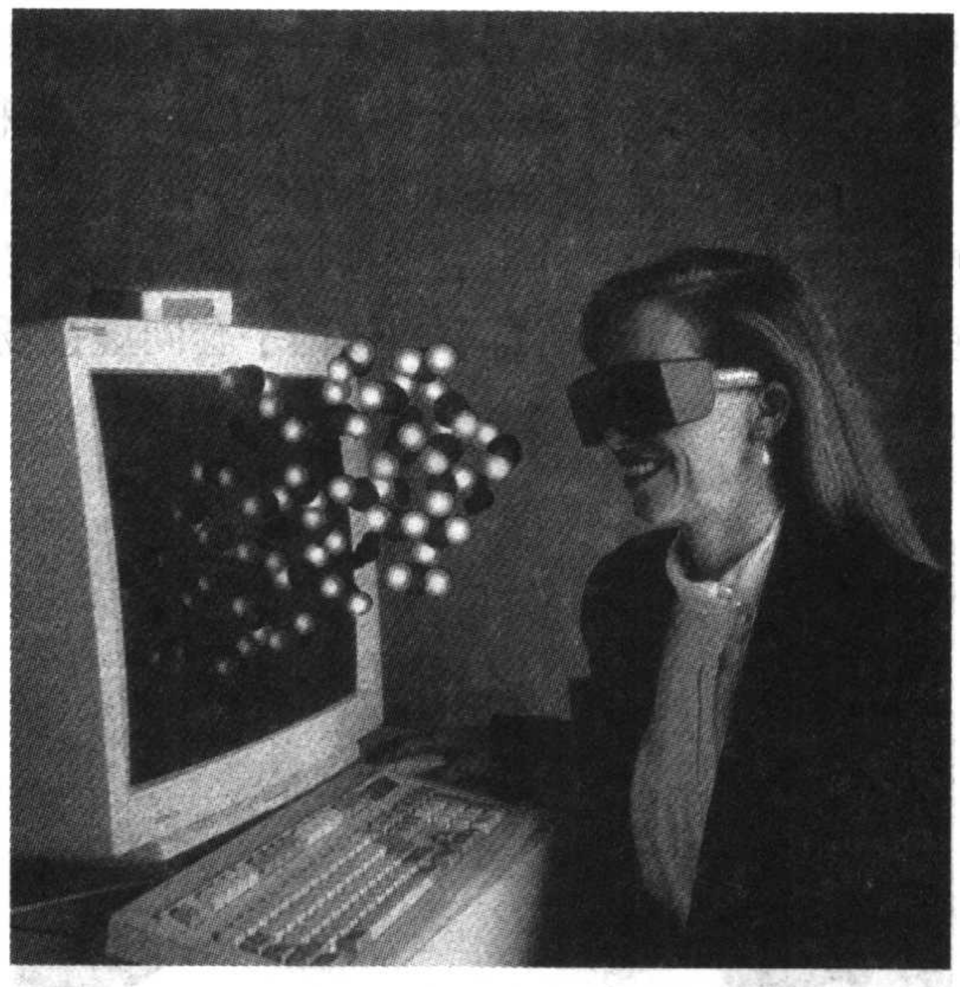
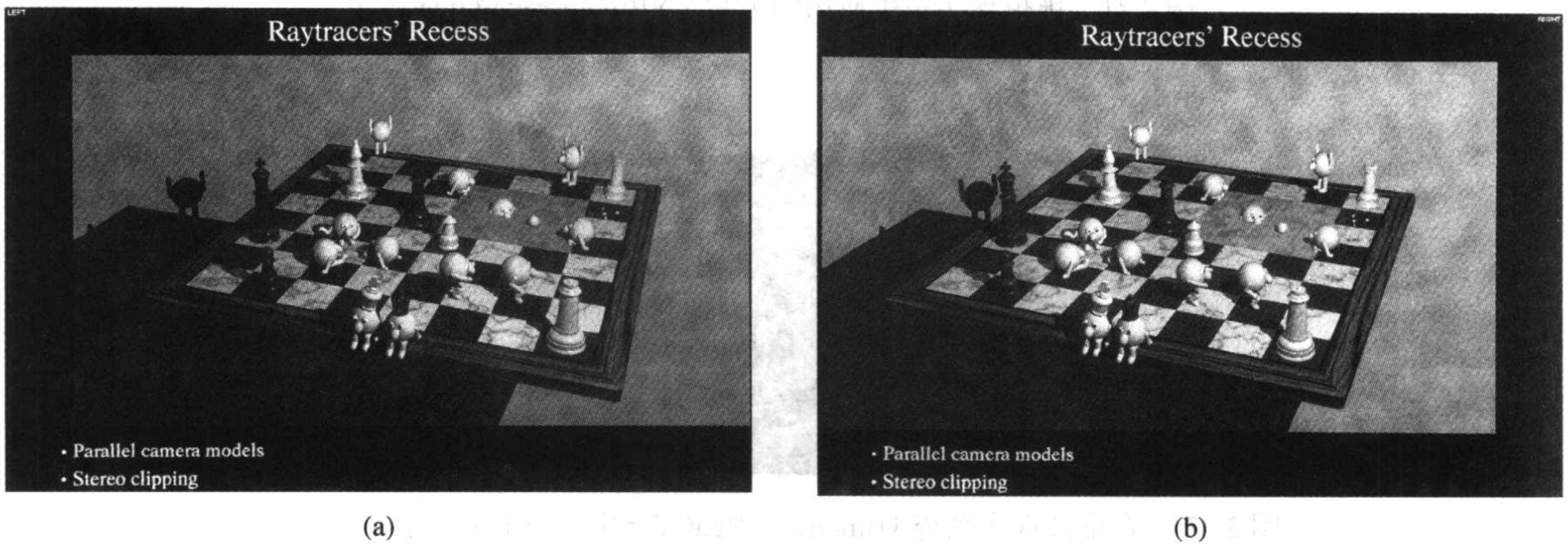


图 2.18　观察一个有立体感的投影（StereoGraphics 公司提供）



(a)　(b)

图 2.19　一对立体感视图（Jerry Farm 提供）

产生立体感效果的途径之一是使用光栅系统在不同的刷新周期交替显示两种视图。通过眼镜观察屏幕，每个透镜设计成高速交替的快门，这种快门能同步阻止另一视图的显示，图2.20给出了一副立体眼镜，它由液晶快门和使眼镜与屏幕视图同步的红外线发射器构成。

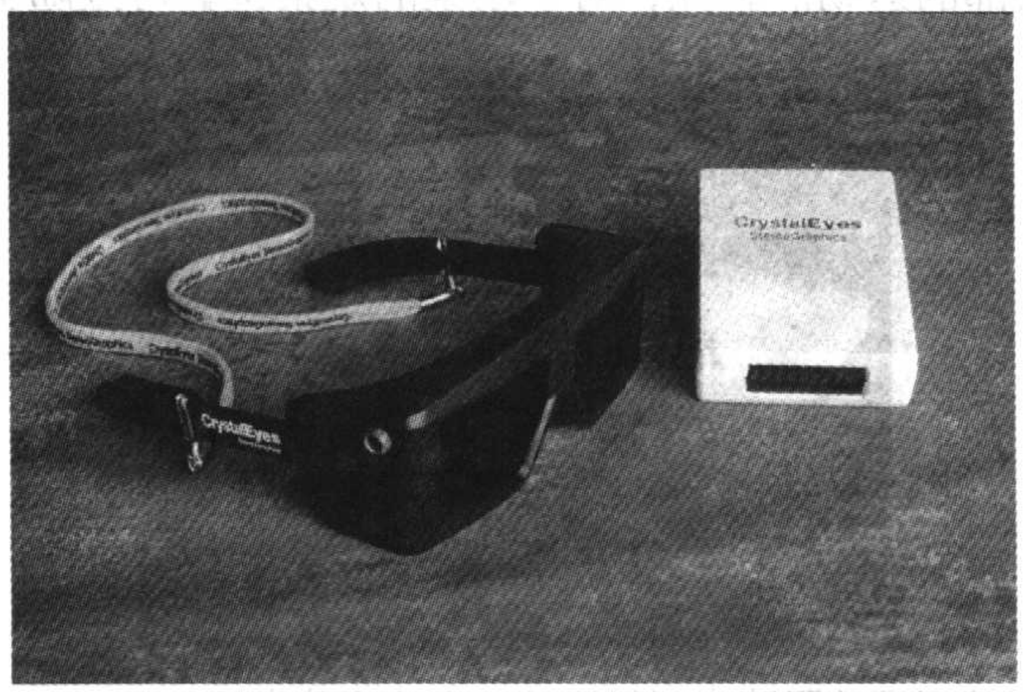


图 2.20　观察立体感场景的眼镜，并配有红外线同步发射器（StereoGraphics 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

立体感视图也是**虚拟现实**（virtual-reality）系统的一个组成部分。用户可以步入场景并同环境进行交互。带有生成立体感视图的光学系统的头套（参见图2.21）可用来连接交互输入设备，从而定位并操纵场景中的对象。头套内的传感系统跟踪观察者的位置，以便在观察者“走进”并同显示进行交互时，能看见对象的正面和背面。另一种生成虚拟环境的方法是使用投影仪在布局好的墙上生成场景，图2.22演示了使用立体眼镜和戴在右手的数据手套同虚拟场景的交互（参见2.4节）。

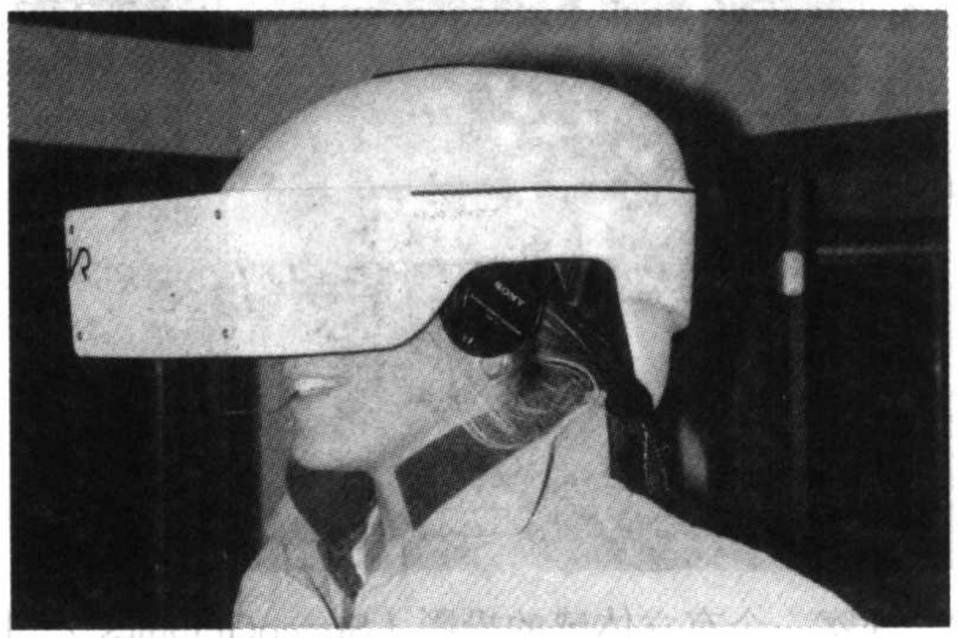


图2.21 虚拟现实系统使用的头套（Virtual Research提供）

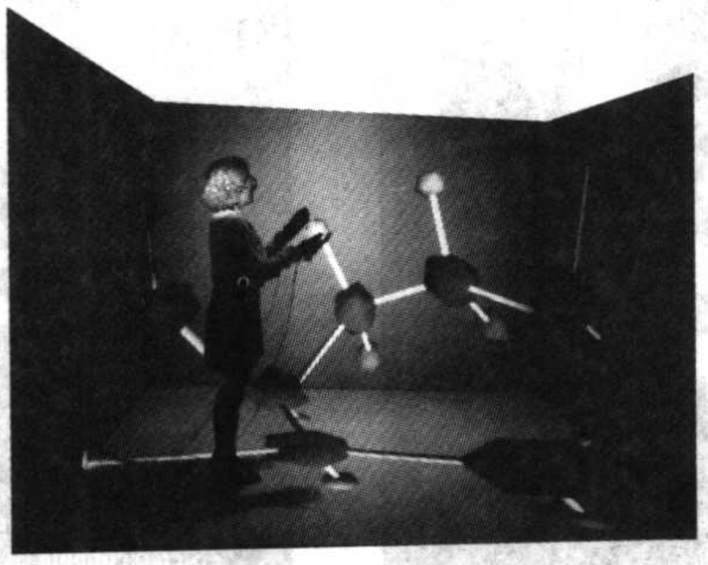


图2.22 在虚拟现实系统Trimension ReaCTor中，分子生物学家在分析分子结构。Fakespace Pinch手套使科学家可以在投影场景中抓住及重新安排虚拟对象（SGI公司提供）

低成本交互虚拟现实环境也可以不用头套而用立体眼镜和视频监视器进行观察，这便提供了一种廉价的虚拟现实系统。图2.23给出了有六个自由度的超声波跟踪设备。跟踪设备放置在视频显示器的顶部，并用来监视头部的运动。因此，对于场景的观察位置可跟随头部位置的变化而变化。

图2.23 与立体眼镜配用的跟踪头部位置的超声波跟踪设备（StereoGraphics公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

2.2　光栅扫描系统

交互式光栅图形系统通常使用几个处理部件。除了中央处理器（CPU）以外，还使用一个**视频控制器**（video controller）或**显示控制器**（display controller）来控制显示设备的操作。简单光栅系统的组织结构如图2.24所示。其中，帧缓存可以在系统存储器的任意位置，视频控制器通过访问帧缓存来刷新屏幕。更高级的光栅系统除了使用视频控制器之外，还使用其他处理器作为协处理器和加速器来完成各种图形操作。

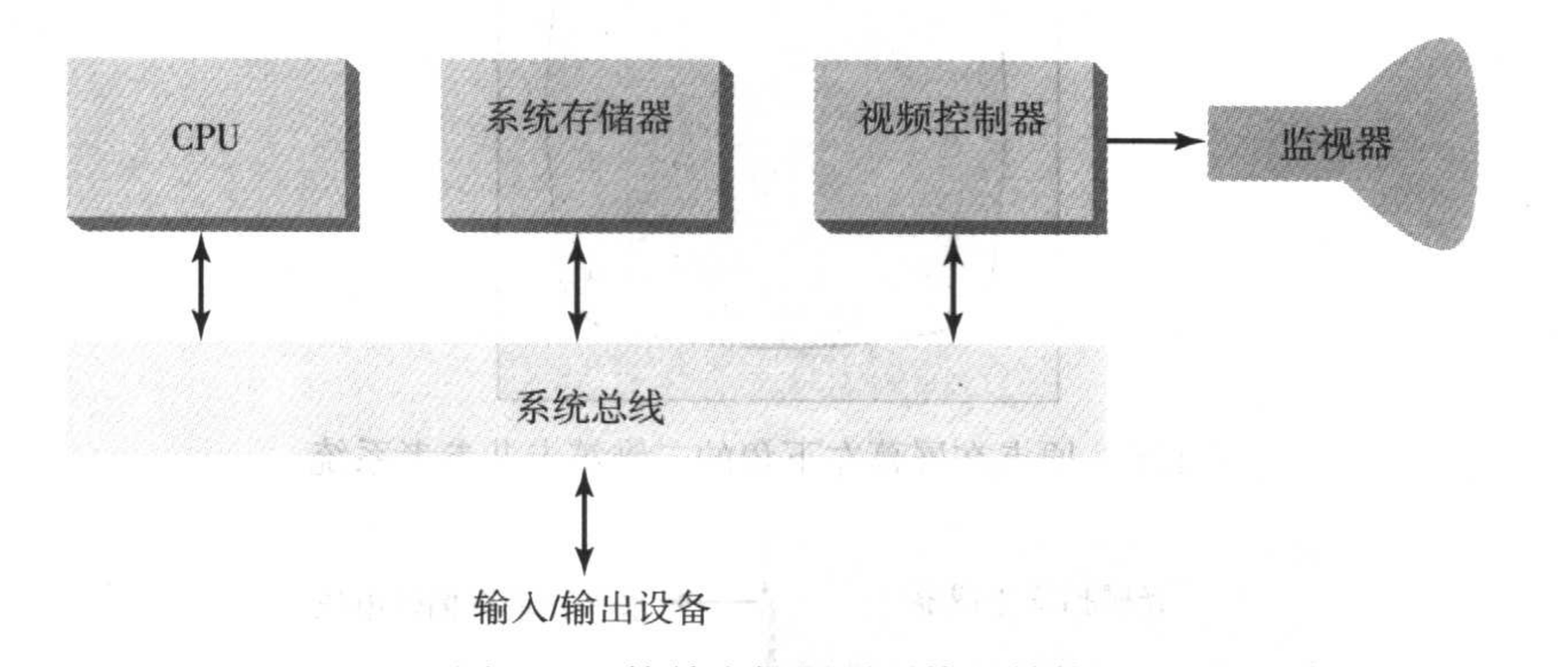


图 2.24　简单光栅图形系统的结构

2.2.1　视频控制器

图 2.25 给出了常用的光栅系统组织。帧缓存使用系统存储器的固定区域，并且由视频控制器直接访问。

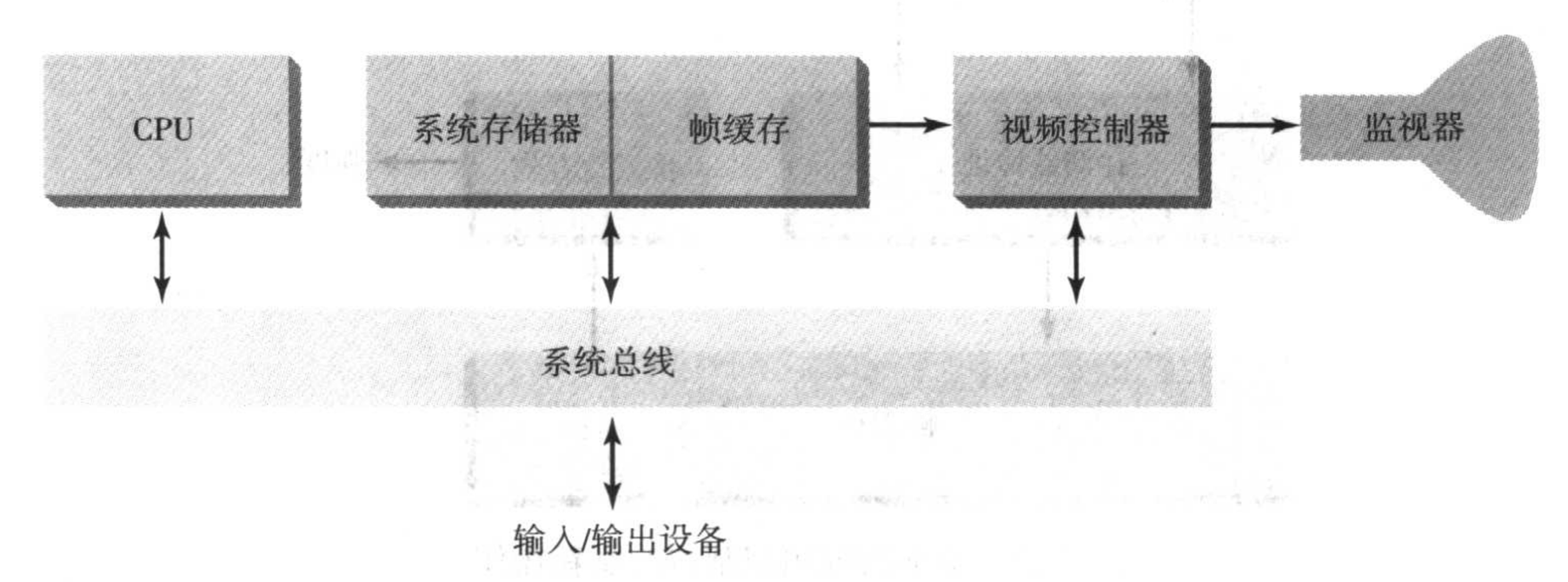


图 2.25　在系统存储器中有固定帧缓存入口的光栅系统的结构

帧缓存的位置以及相应的屏幕位置均使用笛卡儿（Cartesian）坐标。应用程序使用图形软件包的命令来设定显示对象相对于笛卡儿坐标系原点的坐标位置。尽管在特定系统中我们可以将原点设定在任一方便的位置，但多数情况下将坐标原点定义在屏幕的左下角。图2.26给出了一个原点在屏幕左下角的二维笛卡儿参考系统。屏幕表面则表示二维系统的第一象限，正 x 的值向右递增，正 y 的值从下到上递增。像素位置用整数 x 从屏幕左边的 0 到右边的 x_{max} 和整数 y 从底部的 0 到顶部的 y_{max} 来赋值。但在屏幕刷新等硬件处理以及某些软件系统中，像素位置以屏幕左上角为参考。

图 2.27 给出了视频控制器的基本刷新操作流程。有两个寄存器用来存放屏幕像素的坐标。开始时，为顶部扫描行将 x 寄存器置为 0、将 y 寄存器置为 y_{max}。存储在帧缓存中该像素对应位置的值被取出，并用来设置 CRT 电子束的强度值。然后，x 寄存器增加 1，且该过程对顶部扫描线上的下

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

一个像素重复执行，并沿该扫描线对每个像素重复执行。在处理完顶部扫描线的最后一个像素之后，x寄存器复位为0，y寄存器减1，指向顶部扫描行的下一行。然后，依次处理沿该扫描线的各像素，且该过程对每条后继的扫描线重复执行。当循环处理完底部扫描线的所有像素后，视频控制器将寄存器复位为最高行扫描线上第一个像素的位置，刷新过程重复开始。

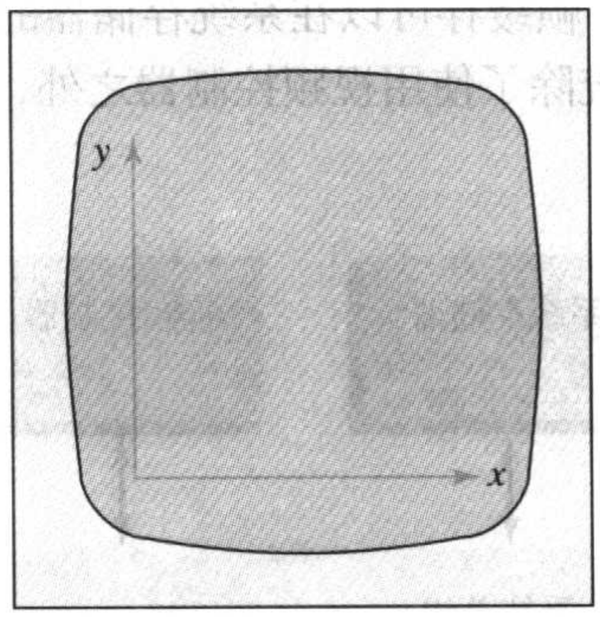


图 2.26　原点在屏幕左下角的二维笛卡儿参考系统

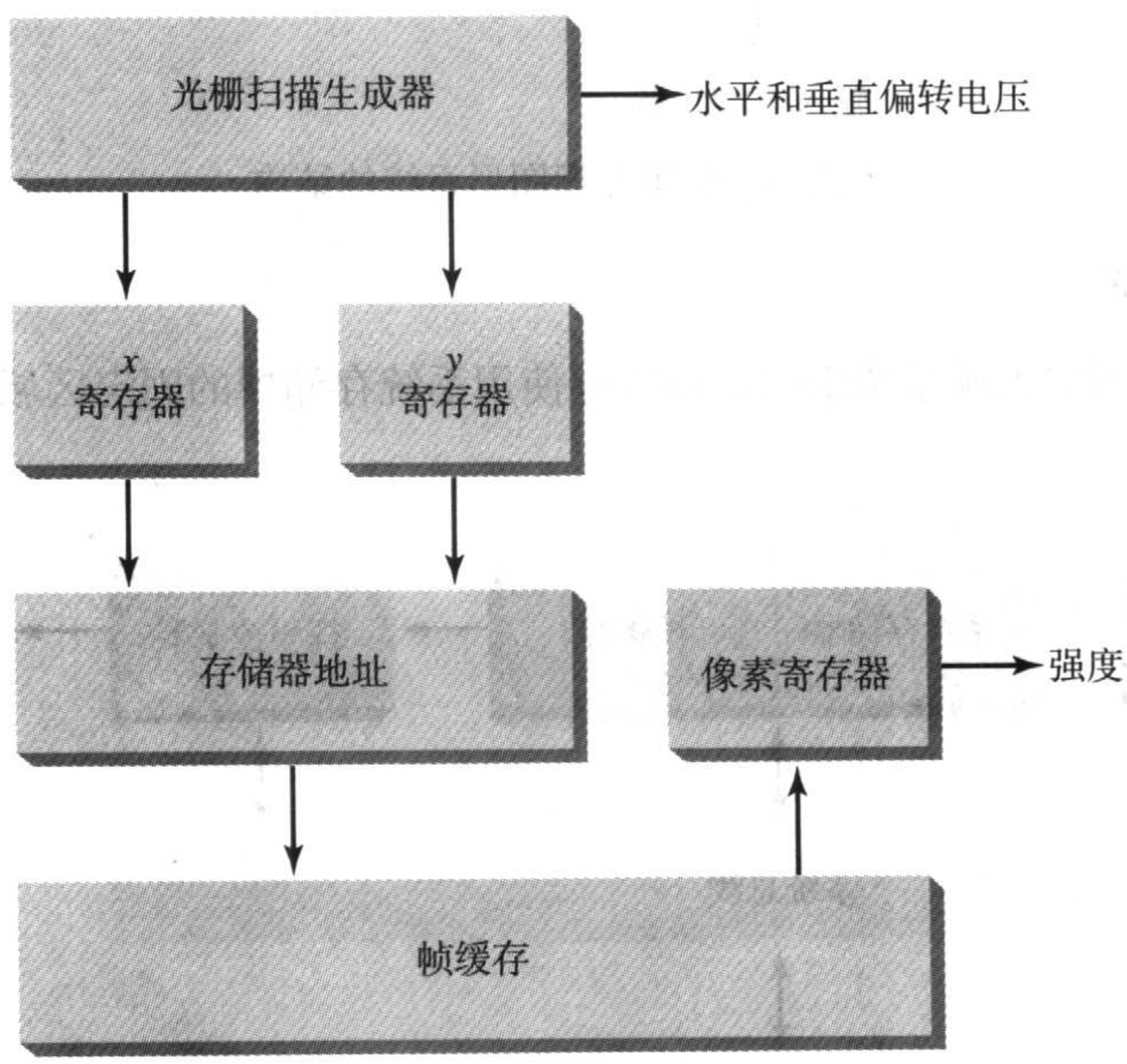


图 2.27　基本的视频控制器的刷新操作

因为屏幕必须按每秒最少60帧的频率刷新，所以图2.27所示的简单过程不能使用循环周期太慢的RAM芯片。为了加速像素处理，视频控制器每次从刷新缓存中取出多个像素值。这些像素强度则存放在单独的寄存器中，用来为一组相邻的像素控制CRT电子束的强度。当处理完该组像素后，从帧缓存取出下一块像素值。

视频控制器还能执行一些其他操作。对于多类应用，视频控制器在不同的刷新周期内可以从不同的存储区中取出像素强度值。例如，在高性能系统中，常常提供两个帧缓存，一个缓存用来刷新，另一个以强度值填充。然后，这两个缓存可以互换角色。这种方法提供了生成实时动画的快速机构，因为正在移动的对象的不同视图可以逐一装入刷新缓存中而不用中断刷新周期。同样，视频控制器可以完成像素块的变换。在一个刷新周期内，屏幕区域可以放大、缩小，或从一个位置移向另一个位置。此外，视频控制器常常包含一个查找表，帧缓存中的像素值用来访问查找表，而不是直接控

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

制CRT电子束强度。这提供了改变屏幕强度值的快速方法，我们将在第4章更详细地讨论查找表。最后，可以将某些系统设计成允许视频控制器将来自电视摄像机或其他输入设备的输入图像与帧缓存图像进行混合。

2.2.2　光栅扫描显示处理器

图2.28给出了建立光栅系统的一种方法，其中包含独立的**显示处理器**（display processor），有时也指**图形控制器**（graphics controller）或**显示协处理器**（display coprocessor）。显示处理器的用途是使CPU从图形的复杂处理中解脱出来。除了系统存储器以外，还可以提供独立的显示处理器的存储区域。

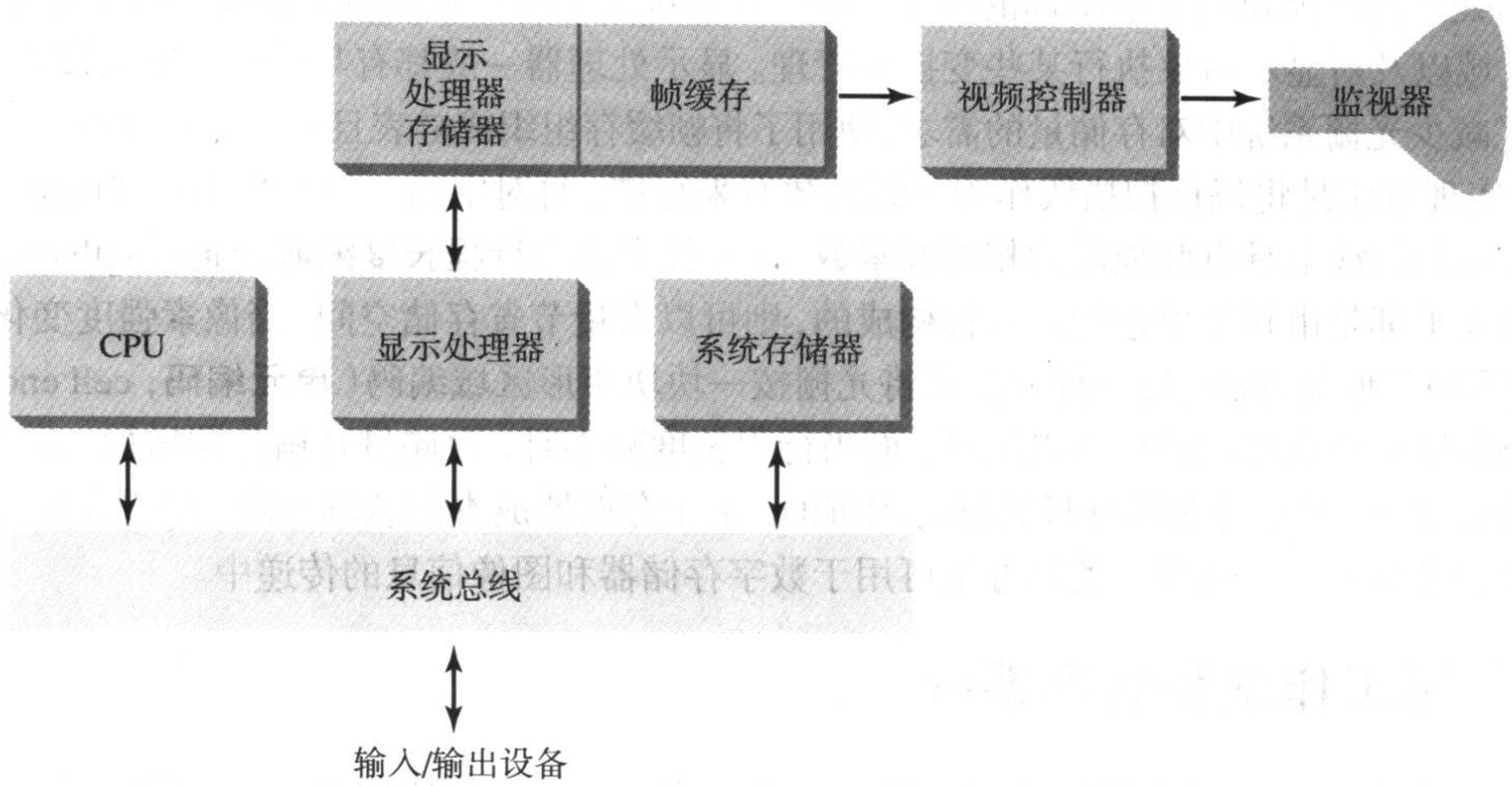


图 2.28　带有显示处理器的光栅图形系统的结构

显示处理器的主要任务是将应用程序给出的图形定义数字化为一组像素强度值，并存放在帧缓存中。这个数字化过程称为**扫描转换**（scan conversion）。扫描转换将给定直线和其他几何对象的图形命令转换为一组与屏幕像素位置对应的离散点。例如，直线段的扫描转换意味着必须确定最接近于直线段路径的像素位置，并把每个位置的强度值存入帧缓存。图形定义中其他对象的扫描转换也使用类似的方法。字符可以用如图 2.29 所示的矩形像素点阵进行定义，或者用如图 2.30 所示的曲线轮廓来定义。字符网格矩阵的大小可以为 5 × 7 到 9 × 12，对于高质量显示则还要大一些。字符的点阵显示是把矩形点阵模式附加到帧缓存中指定的坐标位置。对于使用曲线轮廓定义的字符，通过确定最接近轮廓的像素位置将字符形状扫描转换到帧缓存中。

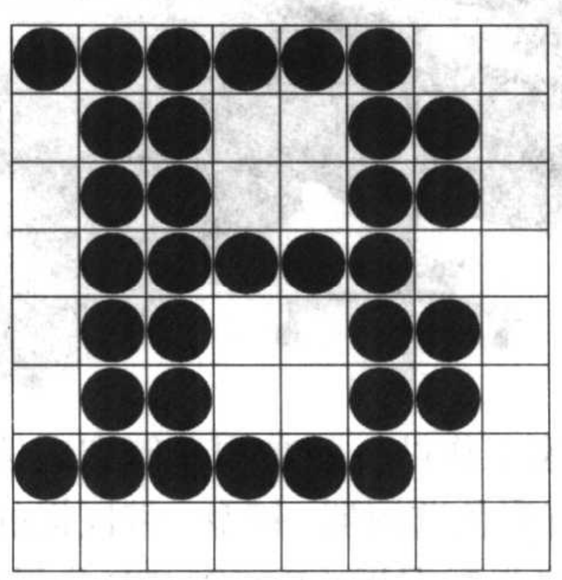


图 2.29　使用矩形像素点阵定义的字符

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

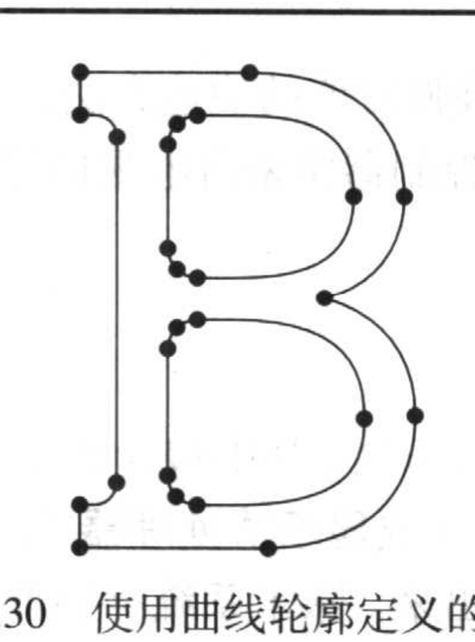


图 2.30 使用曲线轮廓定义的字符

显示处理器也能执行某些附加的操作。这些功能包括生成各种线型（虚线、点线或实线）、显示彩色区域以及对显示对象执行某些变换和管理。显示处理器一般都有与鼠标等输入设备的接口。

为了减少光栅系统中对存储量的需求，使用了将帧缓存组织成链表且对强度信息进行编码的方法。一种实现方法是将每行扫描线作为一组整数对来存储，每对中的一个整数指示强度值，另一个整数设定该扫描线上具有此强度的相邻像素数。这种技术称为**行程长度编码**（run-length encoding）。如果图形几乎都是由每个单色的长行程构成的，则可以大量节省存储空间。当像素强度变化为线性时，也能采用类似的方法。另一种方法是将光栅按一块块矩形区域编码（**单元编码**，cell encoding）。行程编码的缺点是强度的改变难以记录，而当行程长度减小时，存储量开销急剧增加。此外，当包括许多短行程时，显示控制器处理光栅是困难的。由于存储器成本的急剧下降，帧缓存的大小已不再是主要的考虑因素。然而，编码方法可用于数字存储器和图像信息的传递中。

2.3 图形工作站和观察系统

目前，大多数图形监视器以光栅扫描显示的方式工作，一般使用 CRT 或平板系统。图形系统的范围从小型通用计算机系统到多监视器以及超大幅面观察屏幕。个人计算机的屏幕分辨率从 640 × 480 到 1280 × 1024，屏幕对角线从 12 英寸到 21 英寸。现在多数通用系统的彩色能力相当强，许多具备全彩色功能。为图形应用专门设计的桌面工作站的屏幕分辨率从1280 × 1024到1600 × 1200，屏幕对角线为 18 英寸或更大些。商用工作站常配套专门应用的各种设备。图 2.31 给出了一个艺术家工作站的配置。

图 2.31 一个艺术家工作站，除数据存储和通信设备外，还配有彩色监视器、键盘、带鼠标的图形数据板和光学数据板（DICOMED公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

分辨率为 2560 × 2048 的高性能图形系统常用于医学成像处理、空中交通控制、仿真和 CAD。图 2.32 给出了一种 2048 × 2048 的平板显示器。

许多高端图形工作站也带有一些特殊的大型观察屏幕。图 2.33 给出了用于立体视图的大屏幕系统，而图 2.34 是一个多通道宽屏系统。

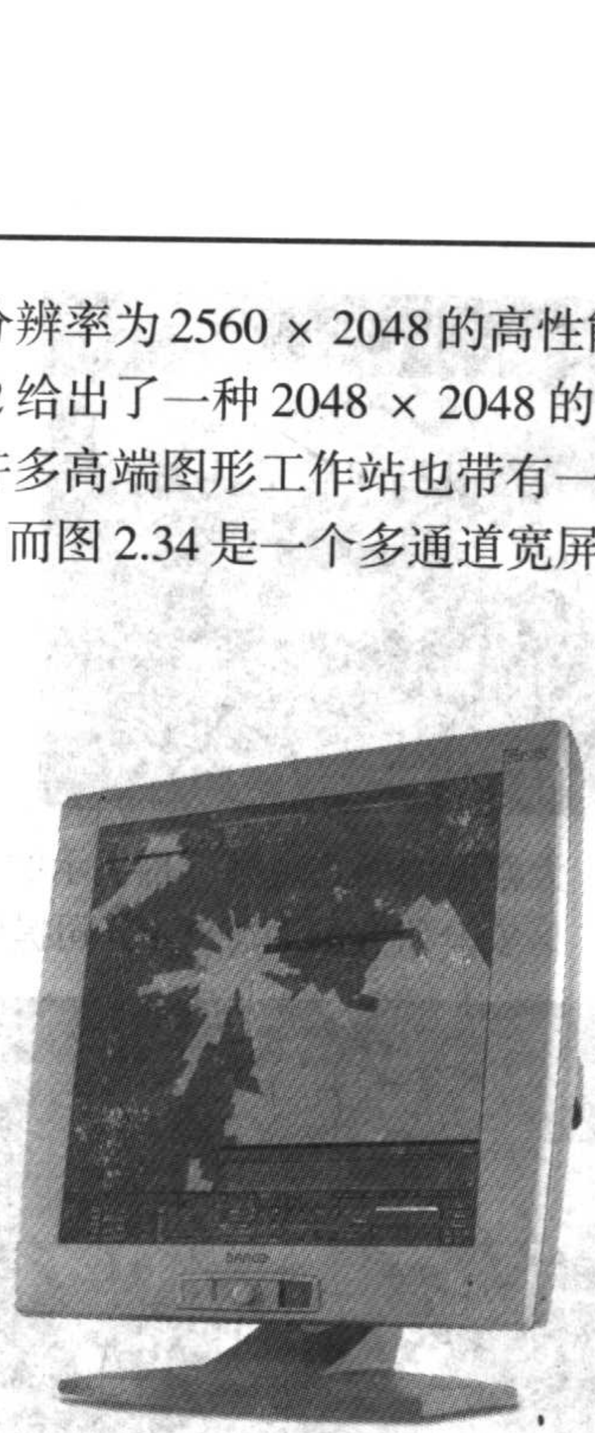


图 2.32　一种高分辨率（2048 × 2048）图形工作站（BarcoView 提供）

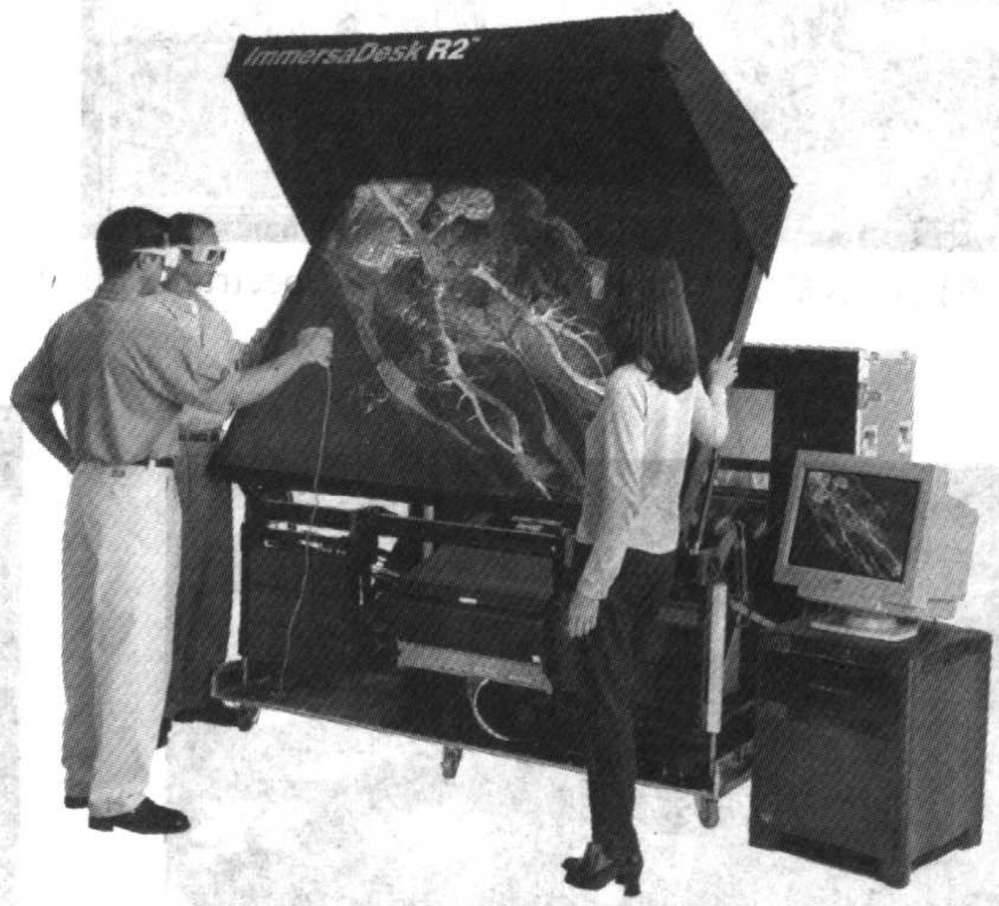


图 2.33　SGI Reality Center 2000D，配有 ImmersaDesk R2，显示体绘制解剖数据集上添加的脉管血流仿真中的压力层位线大屏幕立体视图

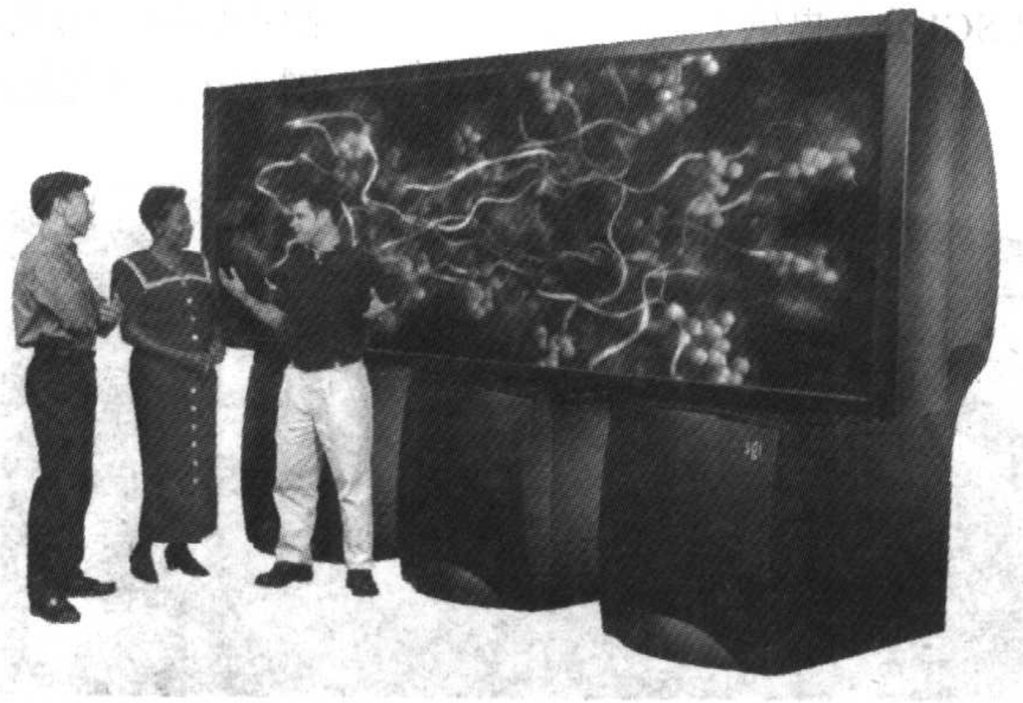


图 2.34　三通道 SGI Reality Center 3300W 上显示的宽屏分子系统视图

多板显示屏幕用在需要“墙幅面”观察区域的各种应用中。这些系统专为会议、学术交流、集会、交易展示、百货商店、博物馆和旅客候机厅等地方显示图形而设计。多板显示用来给出单个场景的大幅面显示或多个独立图像。如图 2.35 所示，系统中每一块板显示整个图形的一部分。大幅面图形显示也可出现在图 2.36 所示的曲面式观察屏幕上。大幅面、曲面式屏幕系统在如图 2.37 和图 2.38 所示的许多人一起研究的特定图形应用中特别有用。配有一组标准监视器的控制中心允许操作员观察大幅面显示的各个部分，并通过触摸屏菜单来控制语音、视频、灯光及投影系统。系统投影机提供了不常见的多通道显示，其中包括边融合、变形校正和彩色平衡。环绕音响系统用来提供语音环境。图 2.39 给出了 NASA 控制塔模拟器中的 360° 观察系统，它用于训练和测试以解决飞机场空中交通和跑道问题。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

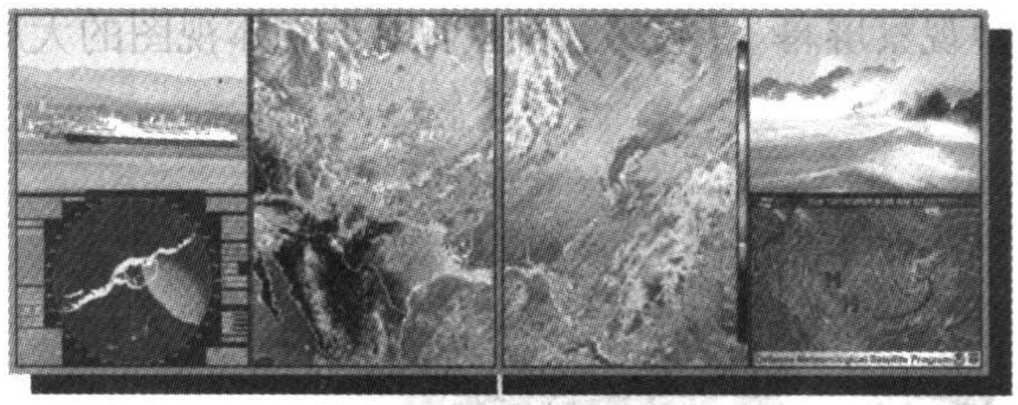


图 2.35 多板显示系统“Super Wall”（RGB Spectrum 提供）

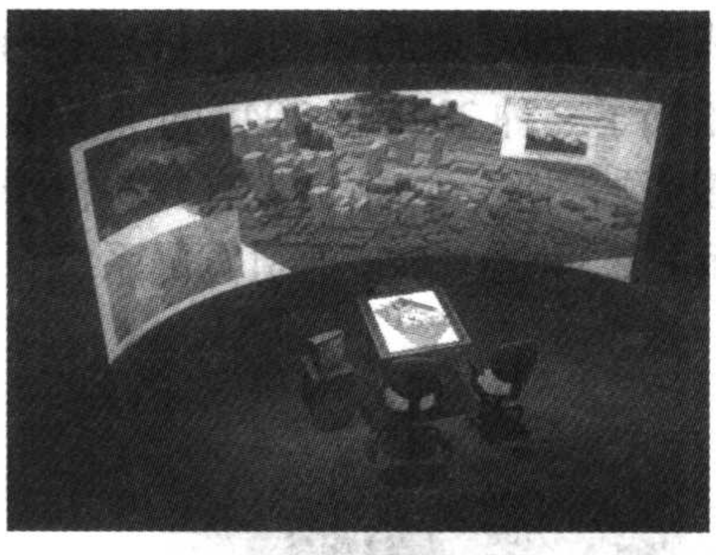


图 2.36 大幅度曲面式观察屏幕显示的地区安全研究画面（SGI 公司提供）

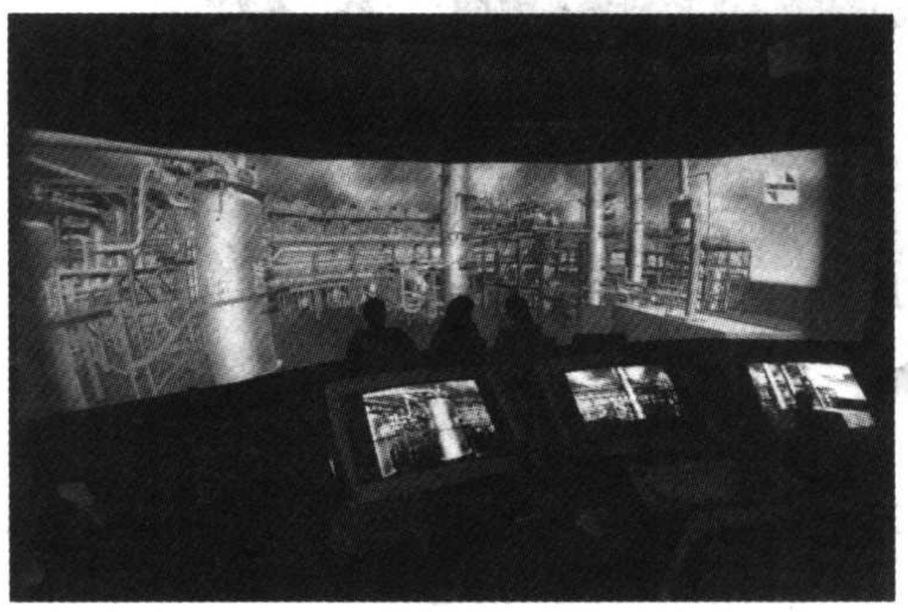


图 2.37 显示天然气工厂交互式“walk-through”的曲面式屏幕图形（SGI 公司提供）

图 2.38 展示在 25 英尺（1 英尺 = 0.3048 米）半圆形屏幕上的地球物理可视化，水平范围 160°，垂直范围 40°（SGI 公司提供）

图 2.39 NASA 飞机场控制塔模拟器的 360° 观察屏幕，称为 FutureFlight Central Facility（SGI 公司提供）

2.4 输入设备

可用于图形工作站数据输入的设备有很多种。多数系统有一个键盘和一个或多个专门为交互输入而设计的其他设备。这些设备包括鼠标、跟踪球、空间球和操纵杆。适合特殊应用的其他输入设备有数字化仪、旋钮、按钮盒、数据手套、触摸板、图像扫描仪和语音系统。

2.4.1 键盘、按钮盒和旋钮

图形系统的字母数字**键盘**（keyboard）主要用于录入文本串、发布一定的命令和选择菜单项。键盘是输入那些与图形显示有关的图形标记等非图形数据的高效设备。键盘也能用来进行屏幕坐标的输入、菜单选择或图形功能选择。

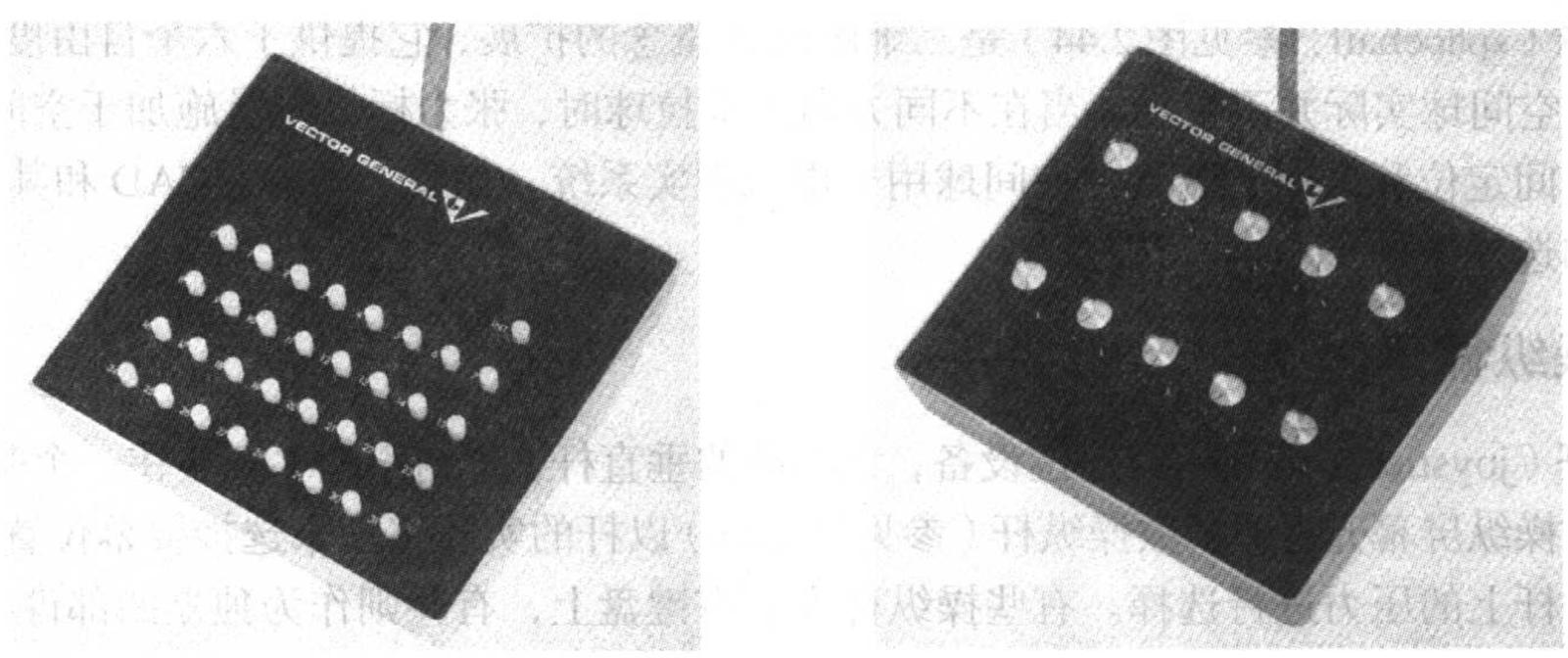
Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

光标控制键和功能键是通用键盘的组成部分。功能键允许用户以单一击键来输入常用的操作，而光标控制键可用来选择显示的对象，或通过定位屏幕光标来确定坐标位置。除了用来快速输入数值数据的数字键盒之外，键盘上可以包含其他类型的光标定位设备，如跟踪球或操纵杆。此外，某些键盘按人体工程学原理进行设计（参见图 2.40），从而可以为减少操作员的疲劳而进行调整。

图 2.40　按人体工程学设计的可移动手握式键盘，键盘两半边的倾斜角度可分开调节。键盘前的单键鼠标通过电缆与主机相连（Apple 公司提供）

对于某些特殊任务，图形应用的输入可能来自一组按钮、旋钮，或者是选择数据值或自定义的图形操作的开关。图 2.41 给出**按钮盒**（button box）和一组**输入旋钮**（input dial）的例子。按钮和开关常用来输入预定的功能，而旋钮是用于输入标量值的常用设备，某些定义范围内的实数可以通过旋钮的旋转来选择输入。首先用电压计量器来测量旋钮的旋转量，然后再转变为相应的数值。



(a)　(b)

图 2.41　(a)按钮盒；(b)一组输入旋钮（Vector General 提供）

2.4.2　鼠标设备

鼠标（mouse）是一个通过在平板上移动而给光标定位的小型手持盒，图 2.40 给出了单键鼠标的一种代表性设计。鼠标底部的转轮或滚轮可用来记录移动的总量和方向。另一种检测鼠标运动的办法是使用光学感应器。某些光学鼠标在特殊的有水平线和垂直线网格的鼠标衬板上移动。光学感应器检测跨越网格线的移动量。另一些光学鼠标可在任意的表面上工作。有的无绳鼠标通过数字无线电技术与计算机进行通信。

由于鼠标可以在某一位置提起并在另一位置放下却不会改变光标移动，因而鼠标可用于控制屏幕光标位置的相对变化。鼠标顶部通常有一个、两个或三个按钮，用来给出某些操作的执行信号，如记录光标位置或调用某个功能。现在，多数通用图形系统以鼠标和键盘作为主要的输入设备。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

基本的鼠标中也可以包括一些附加设备，从而增加允许的输入参数数量。图 2.42 的 Z 鼠标包含三个按钮、侧边的拇指转轮、顶部的跟踪球以及底部的标准鼠标球。这种设计提供了六个自由度来选择空间位置、旋转和其他参数。使用Z鼠标可以拾取监视器上显示的一个对象，使之旋转，并按任意方向移动。也可以利用Z鼠标在三维场景中控制观察位置和方向。Z 鼠标的应用包括虚拟现实系统、CAD 和动画。

图 2.42　Z鼠标有三个按钮，底部为鼠标球，侧边为拇指转轮，顶部为跟踪球（Multipoint Technology 公司提供）

2.4.3　跟踪球和空间球

跟踪球（trackball）是一个球设备，可以通过手指或掌心对其旋转而使屏幕光标移动。与球相连的电压计量器测量球的旋转量和方向。膝上型键盘中常配有跟踪球来取代鼠标所需的空间。跟踪球也可以安装在如图 2.42 所示的Z 鼠标等其他设备上，或用做包含两个或三个控制键的附加部件。

空间球（spaceball，参见图 2.44）是二维跟踪球概念的扩展，它提供了六个自由度。与跟踪球不同的是，空间球实际并不移动。当在不同方向上推拉球时，张力标尺测量施加于空间球的压力，从而提供空间定位和方向的输入。空间球用于虚拟现实系统、建模、动画、CAD 和其他应用中的三维定位和选择操作。

2.4.4　操纵杆

操纵杆（joystick）是另一类定位设备，它由小的垂直杆（称为手杆）安装在一个基座上构成。操纵杆用于操纵屏幕光标。多数操纵杆（参见图 2.43）以杆的实际移动来选择屏幕位置，而其他操作则根据对杆上的压力进行选择。有些操纵杆安装在键盘上，有些则作为独立的部件进行设计。

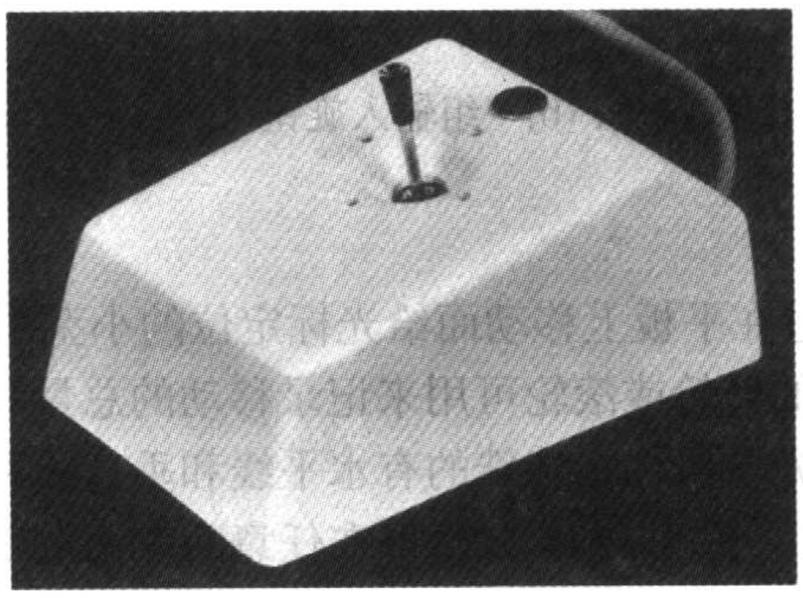


图 2.43　可移动的操纵杆（CalComp Group, Sanders Associates 公司提供）

手杆从其中心位置向任意方向移动的距离对应于屏幕光标向该方向的移动量。安装在操纵杆底部的电压计量器用来测量移动量，弹力将被释放的手杆弹回到中心位置。可以通过编程将一个或多个按钮用做输入开关，从而在选定屏幕位置时给出某些操作信号。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在另一类可移动操纵杆中，手杆用来激活开关，从而引起屏幕光标在选定方向上以恒定速度移动。有时提供八个开关并排成一圈，从而可以使用手杆选择八个方向中的任意方向来移动光标。压力感应式操纵杆（也称等轴操纵杆）有一个不可移动的手杆。由张力标尺测量施于手杆的压力或拉力，并将其转换为指定方向的屏幕光标移动量。

2.4.5　数据手套

图 2.44 给出了一个可以用来抓住“虚拟对象”的**数据手套**（data glove）。手套由一系列检测手和手指运动的传感器构成。发送天线和接收天线之间的电磁耦合用来提供手的位置和方向等信息。发送和接收天线各由一组三个互相垂直的线圈构成，形成三维笛卡儿坐标系统。来自手套的输入可用来定位或操纵虚拟场景中的对象。该场景的二维投影可在视频监视器上观察，而三维投影则使用头套观察。

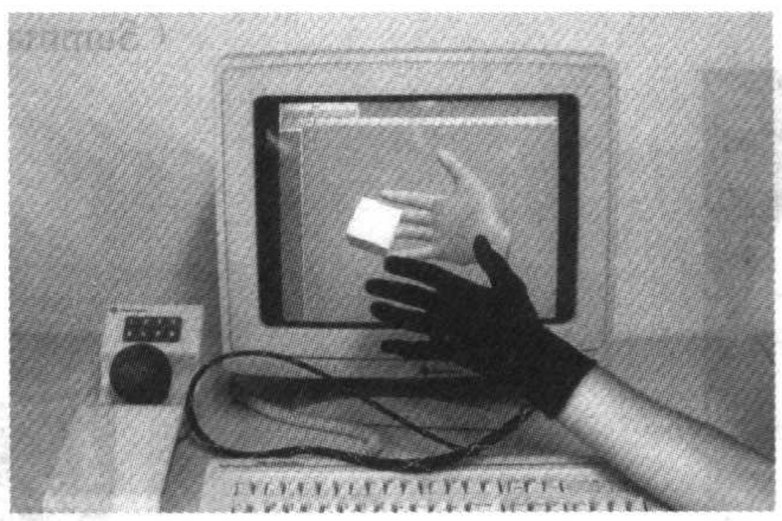


图 2.44　二维视频监视器显示的虚拟现实场景，输入来自数据手套和空间球（The Computer Graphics Center 提供）

2.4.6　数字化仪

数字化仪（digitizer）是绘画、着色或交互式选择坐标位置的常用设备。这类设备可用来输入二维或三维空间的坐标值。在工程和建筑应用中，数字化仪常用来描绘一张工程图或一个对象并输入一组离散的坐标位置。这些输入位置按直线段相连来逼近曲线或表面形状。

图形数据板（graphics tablet，也称数据板）是一种类型的数字化仪，用来在平板表面选定位置时，通过手持光标或触笔来激活二维坐标的输入。手持光标使用交叉发丝指示位置，而触笔是在数据板上指示位置的笔形设备。图 2.45 和图 2.46 给出了台式和落地式数据板的例子，所用的手持光标可取 2、4 或 16 个按钮。图 2.47 和图 2.48 给出了由数据板触笔输入的例子。图 2.48 的艺术家数字化仪系统利用电磁共振来检测触笔的三维位置。这样允许艺术家通过不同的压力对数据板表面产生不同的绘画笔法。数据板的尺寸有许多种，台式为 12 英寸 × 12 英寸，落地式可为 44 英寸 × 60 英寸或更大些。图形数据板提供选择坐标位置的高精度方法，精度可从台式的约 0.2 mm 到较大型号的约 0.05 mm 或更小。

许多图形数据板是在板表面铺设矩形网格线而构成的。沿金属线生成电磁脉冲序列，激活触笔或手持光标，在线圈内感应出电信号，从而记录数据板的位置。采用这种技术，可按信号强度、编码脉冲或相位移来确定在数据板上的位置。

声学（acoustic）或声音（sonic）数据板利用声波来检测触笔位置。可以使用条式麦克风或点式麦克风检测来自触笔端的电火花发出的声音。触笔的位置由其声音到达不同麦克风的时间来计算。二维声学数据板的好处是，麦克风可放置在任何表面位置以形成数据板工作区域。例如，可将麦克风放在一张书页上，从而在手指碰到该页时对其进行数字化。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

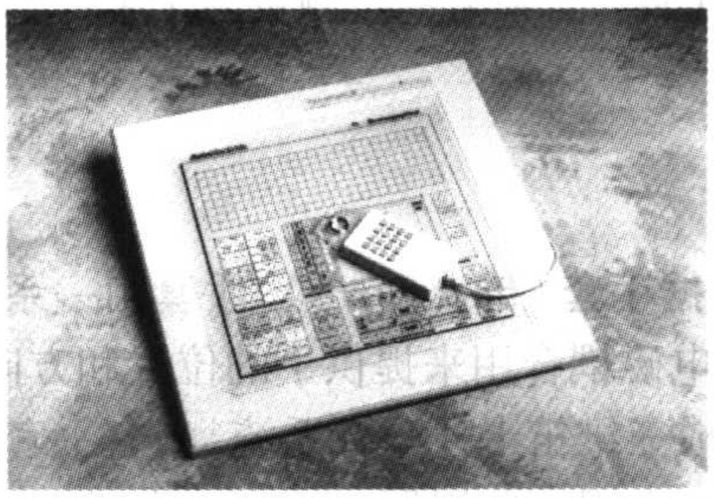


图 2.45 SummaSketch III 型台式数据板，带 16 个按钮的手持光标（Summagraphics 公司提供）

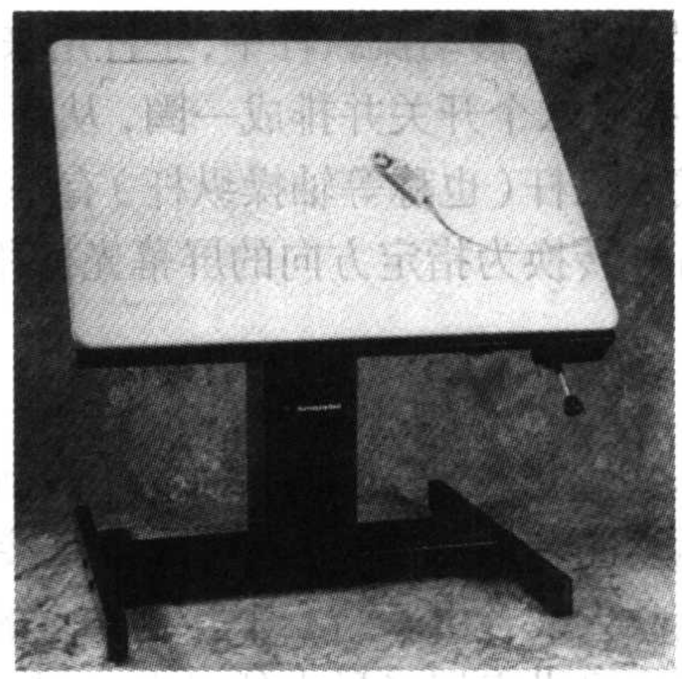


图 2.46 Microgrid 图型数据板，带有 16 个按钮的手持光标，为数字化更大的图形而设计的（Summagraphics 公司提供）

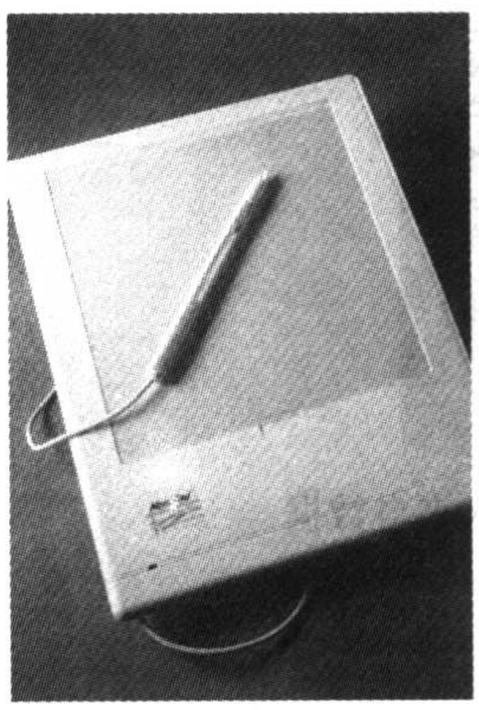


图 2.47 带触笔的 NotePad 台式数据板（CalComp 公司提供）

图 2.48 艺术家数字化仪系统，带压力传感的无绳触笔（Wacom Technology 公司提供）

三维数字化仪使用声音或电磁传播来记录位置。电磁传播的方法之一与数据手套所使用的方法相类似：发送器和接收器之间的耦合用来计算触笔在对象表面移动时的位置。图 2.49 给出了为 Apple Macintosh 计算机设计的三维数字化仪。在非金属物体表面选择点的时候，表面的线框轮廓显示在计算机屏幕上。一旦形成表面轮廓，可以利用光照效应进行绘制，从而产生该物体的逼真显示。该系统的分辨率从 0.8 mm 到 0.08 mm，根据型号的不同而不同。

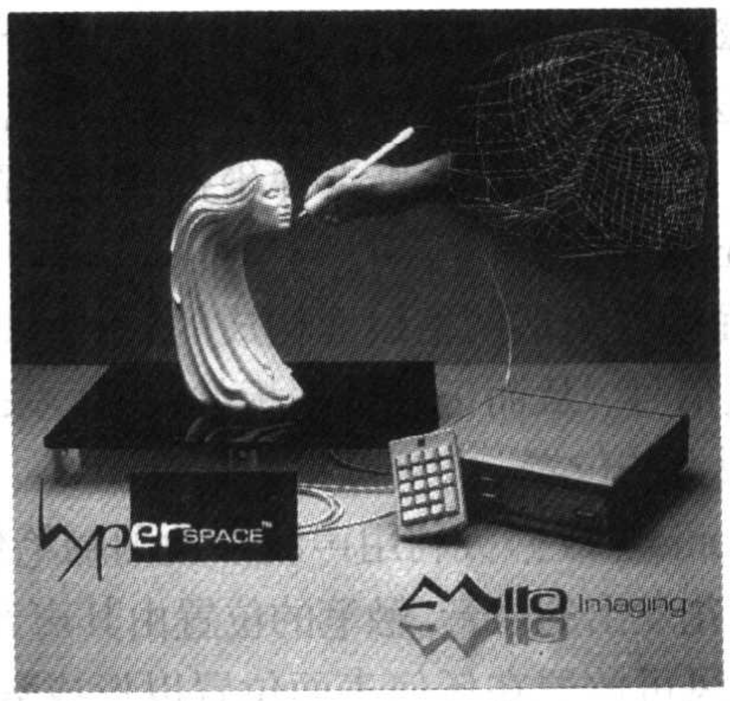


图 2.49 Apple Macintosh 计算机采用的三维数字化系统（Mira Imaging 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

2.4.7　图像扫描仪

工程图、表格、照片或文本可以使用**图像扫描仪**（image scanner）中的光学扫描装置通过扫描来存入计算机。灰度或彩色等级被记录成一个阵列。一旦获得了图形的内部表示，就可以施加变换、旋转、按比例缩放等操作，或者剪辑该图形使之适合特定的屏幕区域。我们也可以使用各种图像处理方法，从而修改该图形的阵列表示。对于扫描输入的文本可以进行各种编辑操作。扫描仪有不同的尺寸和功能。小型的手持扫描仪如图 2.50 所示，而大型的扫描仪如图 2.51 和图 2.52 所示。

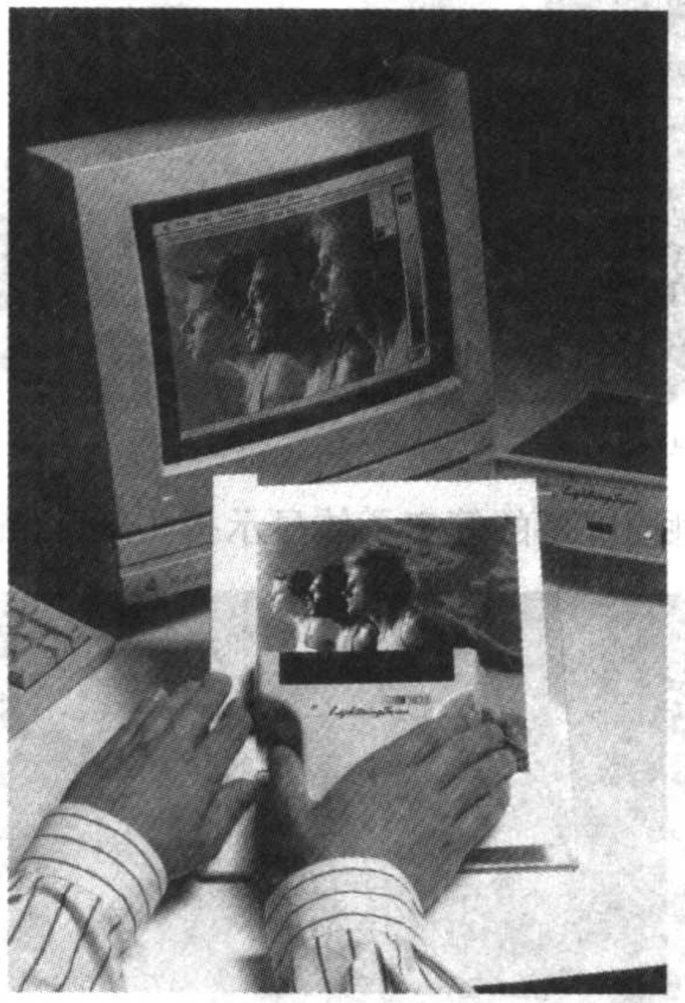
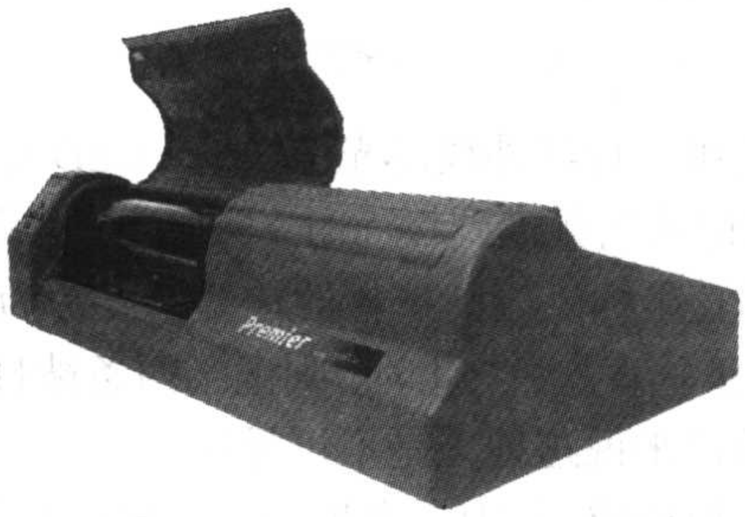
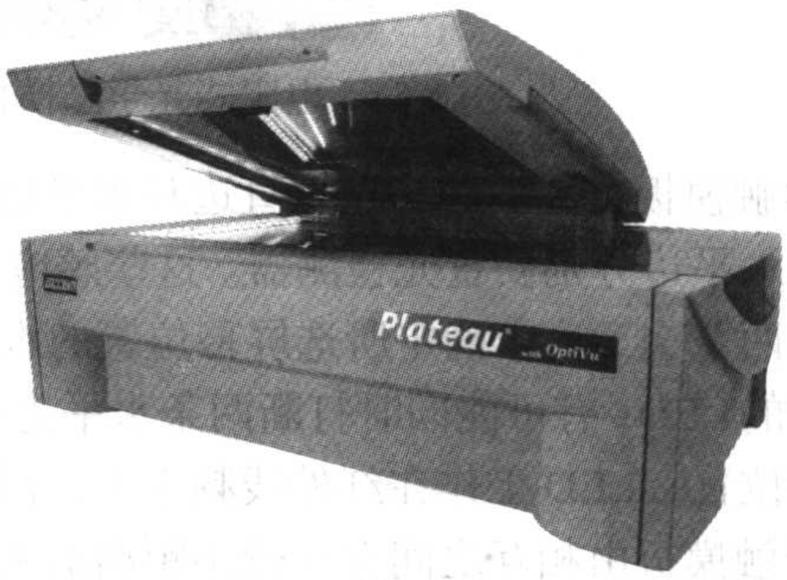


图 2.50　手持扫描仪，可以用于输入文本或图形图像（Thunderware 公司提供）



(a)



(b)

图 2.51　台式扫描仪：(a) 鼓式扫描仪；(b) 平板式扫描仪（Aztek 公司提供）

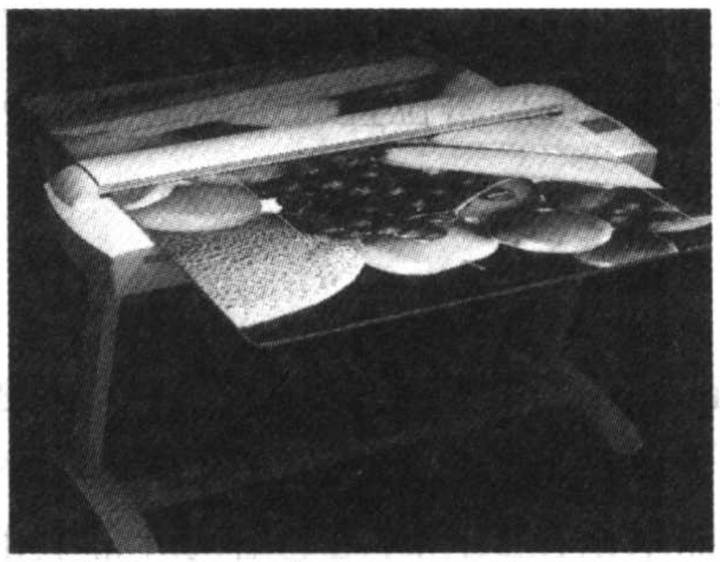


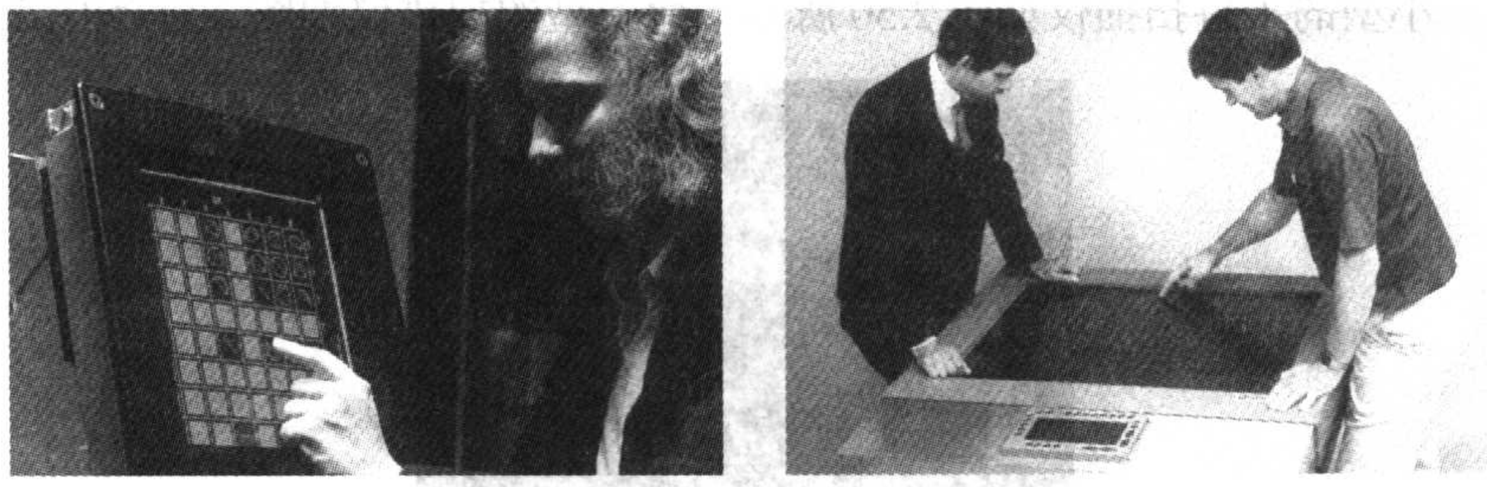
图 2.52　宽幅面扫描仪（Aztek 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

2.4.8 触摸板

顾名思义，**触摸板**（touch panel）允许用手指触摸来选择显示的物体或屏幕位置。触摸板的典型应用是对用图形符号菜单表示的处理选项进行选择。有些监视器，像图2.53所示的等离子体显示板设计成具有触摸屏功能。其他一些系统则通过在视频监视器的屏幕上贴上一个透明的、有触感机能的设备（参见图 2.54）来进行触摸输入。触摸输入可以使用光学、电子或声学方法进行记录。



(a) (b)

图 2.53 带有触摸屏幕的等离子体显示板（Photonics Systems 提供）

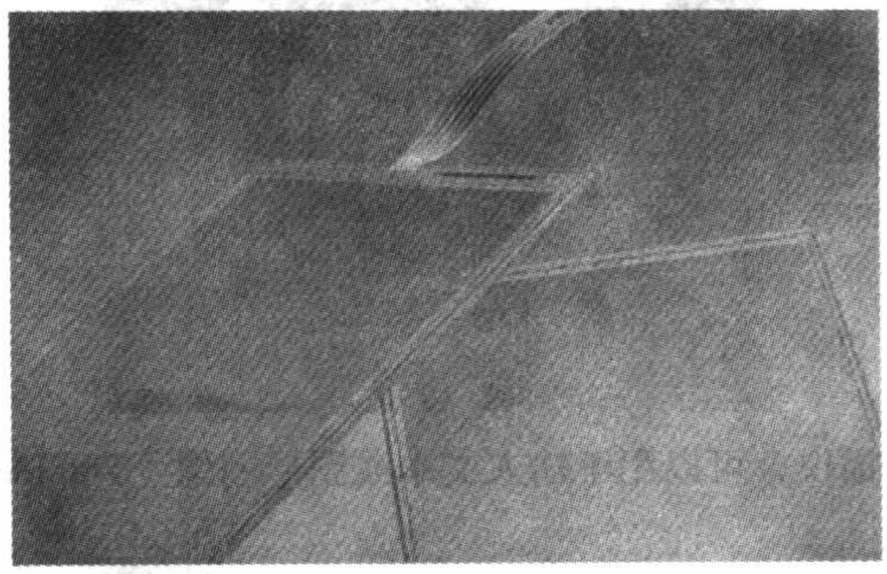


图 2.54 电阻式光学触摸板（Elo TouchSystems 公司提供）

光学触摸板在沿框的一条垂直边和水平边上各使用一行红外线发光二极管（LED），而相对的垂直边和水平边分别安置光检测器。这些感应器用来记录当触摸到板面时，实际打断了哪些光束。被打断的两条交叉光束确定所选屏幕位置的水平和垂直坐标。当前选择的位置精度约为 1/4 英寸。对于密排的LED，有可能同时打断两条水平光束或两条垂直光束。此时，则记录两条被打断光束之间的平均位置。LED 工作在红外线频率上，因而对用户来说这种光是不可见的。

电子触摸板由相互之间有一较小距离的两块透明板构成。其中一个板面涂以导电材料，另一板面涂以电阻材料。当外面一块被触摸时会触及到里面一块，这将引起沿电阻板的电压降低，该压降转换为所选屏幕位置的坐标值。

在声学触摸板中，沿一块玻璃板的水平方向和垂直方向产生高频声波。触摸屏幕引起每个声波的一部分从手指反射到发射器。接触点的屏幕位置通过测量每个声波发送与反射到发射器的时间间隔进行计算。

2.4.9 光笔

图 2.55 给出了一类**光笔**（light pen）的设计。这些笔型设备通过检测来自 CRT 屏幕上某一点的光来选定屏幕位置。它们对电子束打到特定点时荧光层瞬时发射的突发光很敏感。但光笔检测不到房间里的背景光等其他光源。如果正在工作的光笔指向屏幕上正被电子束点亮的点，则生成一个

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

电子脉冲，从而记录该电子束的坐标位置。当光笔作为光标定位设备使用时，记录的光笔坐标可用来定位对象或选择处理选项。

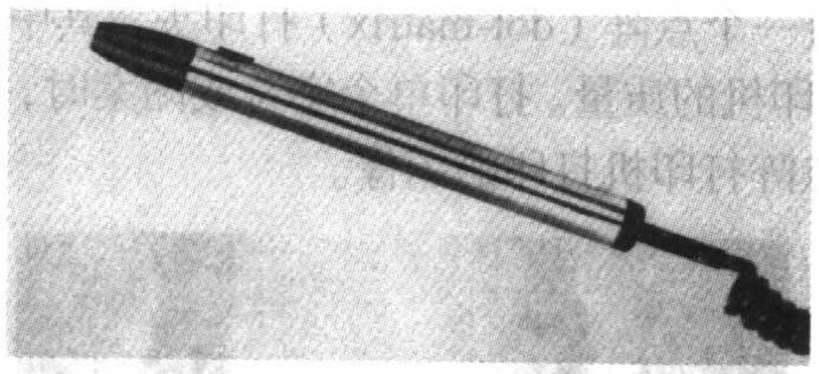


图 2.55　由接钮开关激活的光笔（Interactive Computer Products 提供）

虽然现在光笔仍有人使用，但它们不再像以前那样流行。因为同其他已开发的输入设备相比较，光笔有其不足之处。其一，当光笔指向屏幕时，手和笔将遮挡屏幕图像的一部分。而且长时间使用光笔，会造成手臂的疲劳。对于某些应用，光笔需要经过特殊的加工，因为它们不能检测黑暗区域内的位置。为了使光笔能选择任何屏幕区域的位置，我们应该将每个屏幕像素设定为一些非零亮度；另外，有时因房间发光背景的影响，光笔会产生误读现象。

2.4.10　语音系统

语音识别器在某些图形工作站中是用于接收声音命令的输入设备。**语音系统**（voice system）的输入可用于图形操作的初始化或输入数据。这些系统通过将输入与预定义的字典中的单词和词组进行匹配来进行识别。

字典通过重复多次读出命令单词而建立。系统分析每一单词并将单词的频率模式与将要执行的相应功能一起放到字典中。然后，在给出一个语音命令时，该系统检索字典中与其匹配的频率模式及其对应的内容。对每一个使用该系统的用户有必要建立一个单独的字典。语音输入一般对着如图2.56所示的安装在耳机上的麦克风进行。麦克风设计成将其他背景声音的输入降为最小。语音系统具有某些优于其他输入设备的长处，因为在输入命令时，操作员的注意力不需要从一个设备切换到另一个。

图 2.56　语音识别系统（Threshold Technology 公司提供）

2.5　硬拷贝设备

我们可以使用几种格式来获得图像的硬拷贝输出。为了演示或存档，可以将图像文件传送到生成投影胶片、35 mm幻灯片或电影胶片的设备与服务部门。也可将图形输出到打印机或绘图仪上，从而在纸上打印出图像。

输出设备生成图片的质量依赖于可显示的点的大小和每英寸的点数或每英寸的行数。为了打印精细的图案，高质量的打印机要移动点的位置，使相邻点之间有部分重叠。

打印机以击打式或非击打式产生输出。击打（impact）式打印机隔着色带将某种格式的字符压在纸上。行式打印机是击打设备的一个例子，其字样安装在色带、链条、磁鼓或滚轮上。非击

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

打（nonimpact）式打印机和绘图仪使用激光技术、喷墨技术、静电方式和热转印方式，从而把图像打印在纸上。

字符击打式打印机常常有一个点阵（dot-matrix）打印头，其中包含一组矩形阵列结构的伸出的金属针，针的总数决定着打印机的质量。打印单个字符或图案时，可以缩回某些针而让余下的针进行打印。图 2.57 给出了用点阵打印机打印的图像。

图 2.57　点阵式打印机打印的图像，展示如何改变点图案的密度来造成明暗区域（Apple Computer 公司提供）

在激光（laser）设备里，激光束在涂覆光电材料（如硒）的旋转鼓上建立电荷分布。调色剂施加于旋转鼓，然后转印到纸上。喷墨（ink-jet）法产生的输出，是沿着裹在鼓上的纸卷，逐行水平地将墨水喷于纸上而实现的。带有电荷的墨水流受到电场影响而偏转，从而产生点阵图案。静电（electrostatic）设备每次一整行地将负电荷置于纸上。然后，这张纸将面对带正电荷的调色剂曝光。这使得调色剂附加到负电荷区域，从而产生指定的输出。热转印（electrothermal）打印是另一种输出技术。该系统将点阵打印头加热，在热感应纸上输出图案。

采用不同颜色的色带，可以在击打式打印机上得到有限的彩色输出。非击打式设备利用各种技术，组合三种彩色颜料（青、品红和黄）来产生一定范围的彩色图案。激光和复印设备分几次沉积三种颜料，喷墨法则将三种颜色沿每个打印行同时喷射在纸上。

布局草图和其他工程图一般由喷墨或笔式绘图仪生成。笔式绘图仪中的一支或多支笔安装在横跨纸的笔架或滑杆上，各种颜色和不同粗细的笔用来产生各种阴影和线型。湿墨水笔、圆珠笔和毡尖笔都可用于笔式绘图仪。绘图仪的纸可以铺在平板上或卷在旋转鼓或色带上。滑杆可以是移动的或固定的，笔可沿杆前后移动。通过使用夹板、真空或静电荷来保持纸的定位。图 2.58 给出了一个台式笔式绘图仪的例子。图 2.59 给出了更大的滚动传送的笔式绘图仪的例子。

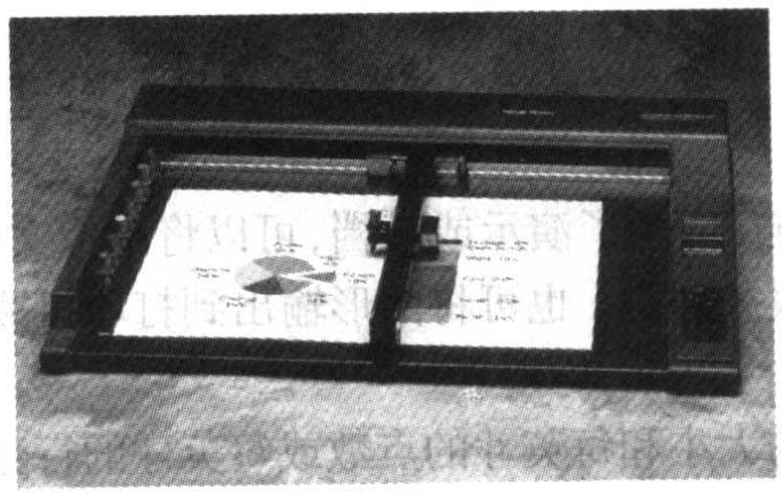


图 2.58　台式笔式绘图仪，分辨率为 0.025 mm（Summagraphics 公司提供）

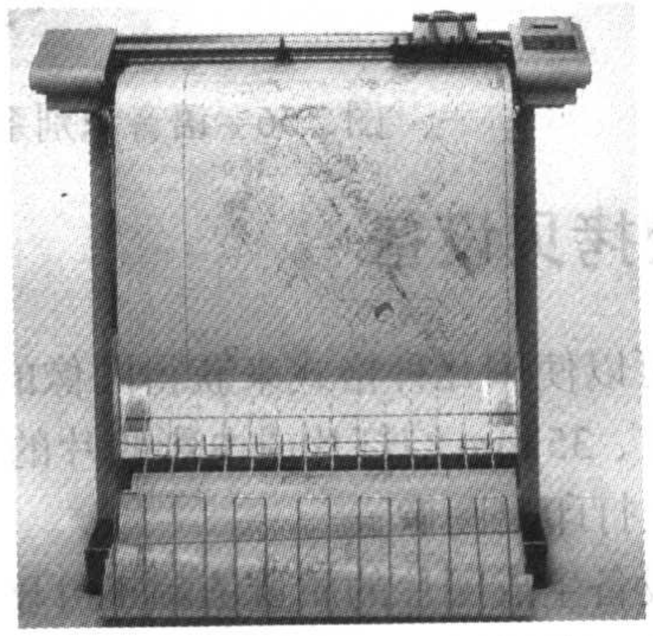


图 2.59　大型的滚动传送绘图仪，带有自动的彩色转换器，分辨率为 0.0127 mm（Summagraphics 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

2.6　图形网络

到目前为止，我们主要考虑了单用户独立系统中的图形应用。但是，多用户环境和计算机网络是目前许多图形应用的普遍特点。处理器、打印机、绘图仪和数据文件等许多资源都分布在网络中并为多个用户所共享。

图形监视器在网络中一般称为**图形服务器**（graphics server），或简称**服务器**。通常，监视器包含有标准的输入设备，如一个键盘和一个鼠标或跟踪球。此时，该系统除了是一个输出服务器外还可以输入。网络上正在运行图形应用程序的计算机称为**客户**（client），其程序的输出在服务器上显示。包含处理器、监视器和输入设备的工作站可以完成服务器和客户两种功能。

在网络上工作时，客户计算机可以向监视器（服务器）发送要求显示一图形的指令。一般要将指令打包后传送，而不是在网络上逐条地传递指令。因此，图形软件包除了有生成图形的命令外还会包含与打包传送有关的命令。

2.7　因特网上的图形

现在大量的图形开发工作在全球性计算机网络——**因特网**（Internet）上进行。因特网上的计算机使用 TCP/IP（传输控制协议 / 网际互连协议）进行通信。**万维网**（World Wide Web）提供了超文本系统，使用户可以放入或阅读包含文本、图形和视频的文档。图形文件等资源由统一资源定位器（uniform resource locator，URL）来识别。有时称为全球资源定位器的 URL 包含两个部分：(1)传送文档的协议；(2)包含文档和可选的服务器位置（目录）的服务器。例如，URL http://www.siggraph.org 指出要用超文本传送协议（hypertext transfer protocol，http）传送的一个文档，并且服务器是 www.siggraph.org，它是计算机学会图形学专业委员会（Special Interest Group in Graphics，SIGGRAPH）的主页。URL 的另一常用类型以“ftp://”开头。这表明一个可以使用文件传送协议（file-transfer protocol）下载程序和其他文件的“ftp 场所”。

因特网上的文档可用超文本标记语言（Hypertext Markup Language，HTML）来组织。HTML 的开发提供了一种简单方法来描述包含文本、图形和对其他文档的引用（超链接）。尽管使用HTML 和 URL 寻址可以找到资源，但在因特网上寻找信息还是很困难的。因此，美国超级计算应用国家中心（National Center for Supercomputing Application，NCSA）开发了一个称为Mosaic的浏览器（后来发展成为 Netscape Navigator），可以使用户比较容易地查找 Web 资源。

超文本标记语言提供了一种在因特网上开发图形应用的简单方法，但其功能有限。因此，其他一些用于因特网图形应用的语言也已经开发出来，2.8 节中将讨论这些语言。

2.8　图形软件

图形软件有两个大类：通用编程软件包和专用应用软件包。专用应用图形软件包是为非程序员设计的，使得他们在某些应用中能生成图形、表格而不用关心显示所需的图形函数。专用软件包的接口通常是一组菜单，用户通过菜单按自己的概念与程序进行通信。这类应用的例子包括艺术家绘画程序和各种建筑、商务、医学及工程 CAD 系统。相反，通用图形编程软件包提供一个可用于 C、C++、Java 或 Fortran 等高级程序设计语言的图形函数库。典型的图形库中的基本函数用来描述图元（直线、多边形、球面和其他对象）、设定颜色、观察选择的场景和进行旋转或其他变换等。通用图形程序设计软件包有 GL（Graphics Library）、OpenGL、VRML（Virtual-Reality Modeling Language，虚拟现实建模语言）、Java 2D 和 Java 3D 等。由于图形函数库提供了程序设计语言（如 C++）和硬

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

件之间的软件接口，所以这一组图形函数称为**计算机图形应用编程接口**（computer-graphics application programming interface，CG API）。在我们使用C++编写应用程序时，可以使用图形函数进行组织并在输出设备上显示图形。

2.8.1 坐标表示

使用程序设计软件包生成图形时，首先需要给出显示对象的几何描述。该描述确定对象的位置和形状。例如，一个立方体由它的顶点位置来描述，一个球由其中心位置和半径来定义。除了少数的例外情况，通用图形软件包要求在标准的、右手系的、笛卡儿坐标参考系统（参见附录A）中给出几何描述。如果一个图形的坐标值是在某个其他参照系（球面坐标、双曲坐标等）中指定的，那么它们必须先转换为笛卡儿坐标再输入图形软件。某些针对专门应用的图形软件包允许采用适合该应用的其他坐标系。

通常，在构造和显示一个场景的过程中会使用几个不同的笛卡儿参照系。首先在各自的坐标系统中构造每一对象的形状，比如树或家具。这些坐标系称为**建模坐标系**（modeling coordinate），有时也称为**局部坐标系**（local coordinate）或**主坐标系**（master coordinate）。一旦指定了单个物体的形状，我们可将对象放到称为**世界坐标系**（world coordinate）的场景参照系的适当位置。这一步涉及从单独的建模坐标系到世界坐标系的指定位置和方向的变换。作为一个例子，我们可以在各个独立的建模坐标系中定义自行车的零件（车轮、车架、坐垫、车把手、齿轮、链条、踏板等），然后将这些零件在世界坐标系中装配起来。如果两个车轮尺寸相同，我们只需要在局部坐标系中定义一个车轮。该车轮装配到世界坐标系的两个位置。如果场景不是很复杂，对象的各部分可以直接在世界坐标系中建立，从而跳过建模坐标和建模变换两步。在建模坐标系和世界坐标系中可使用任何浮点数或整数值来给出几何描述，而不受特定输出设备的约束。对于某些场景，可能要用一英尺的分数值来指定物体尺寸；而对于其他的应用场合，我们可能要用毫米、千米或光年等单位。

在描述好场景的所有部分之后，要将该场景的世界坐标描述经各种处理变换到一个或多个输出设备参照系来显示。这个过程称为**观察流水线**（viewing pipeline）。世界坐标系位置首先转换到与我们要对场景进行的观察所对应的观察坐标系（viewing coordinate），该转换依据假想照相机的位置和方向而进行。然后，对象位置变换到该场景的一个二维投影，这将对应于我们在输出屏幕上看到的结果。然后将该场景存入**规范化坐标系**（normalized coordinate），其坐标范围从 -1 到 1 或从0到1，这依赖于不同的系统。规范化坐标系也称为规范化设备坐标系，使用该表示可使图形软件包与任何特定输出设备的坐标范围无关。我们还要识别可见面并清除在显示设备上观察边界之外的图形部分。最后，图形经扫描转换到光栅系统的刷新缓存中进行显示。显示设备的坐标系称为**设备坐标系**（device coordinate）或对视频监视器而言称为**屏幕坐标系**（screen coordinate）。规范化坐标系和屏幕坐标系都是左手系，即从 xy 平面增加的正距离方向可解释为远离观察位置而去。

图 2.60 给出了对于一个三维对象从建模坐标到设备坐标的坐标变换序列。图中初始的建模坐标位置(x_{mc}, y_{mc}, z_{mc})变换为世界坐标系，接着是观察和投影坐标系，然后是左手、规范化坐标系，最后到设备坐标系位置(x_{dc}, y_{dc})。其序列为

$$(x_{mc}, y_{mc}, z_{mc}) \rightarrow (x_{wc}, y_{wc}, z_{wc}) \rightarrow (x_{vc}, y_{vc}, z_{vc}) \rightarrow (x_{pc}, y_{pc}, z_{pc}) \rightarrow (x_{nc}, y_{nc}, z_{nc}) \rightarrow (x_{dc}, y_{dc})$$

对于特定的输出设备，设备坐标 x_{dc} 和 y_{dc} 是整数，范围为(0，0)到(x_{max}, y_{max})。除了观察表面的二维坐标(x_{dc}, y_{dc})，还保存每一设备坐标位置的深度信息，用于各种可见性和面处理算法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

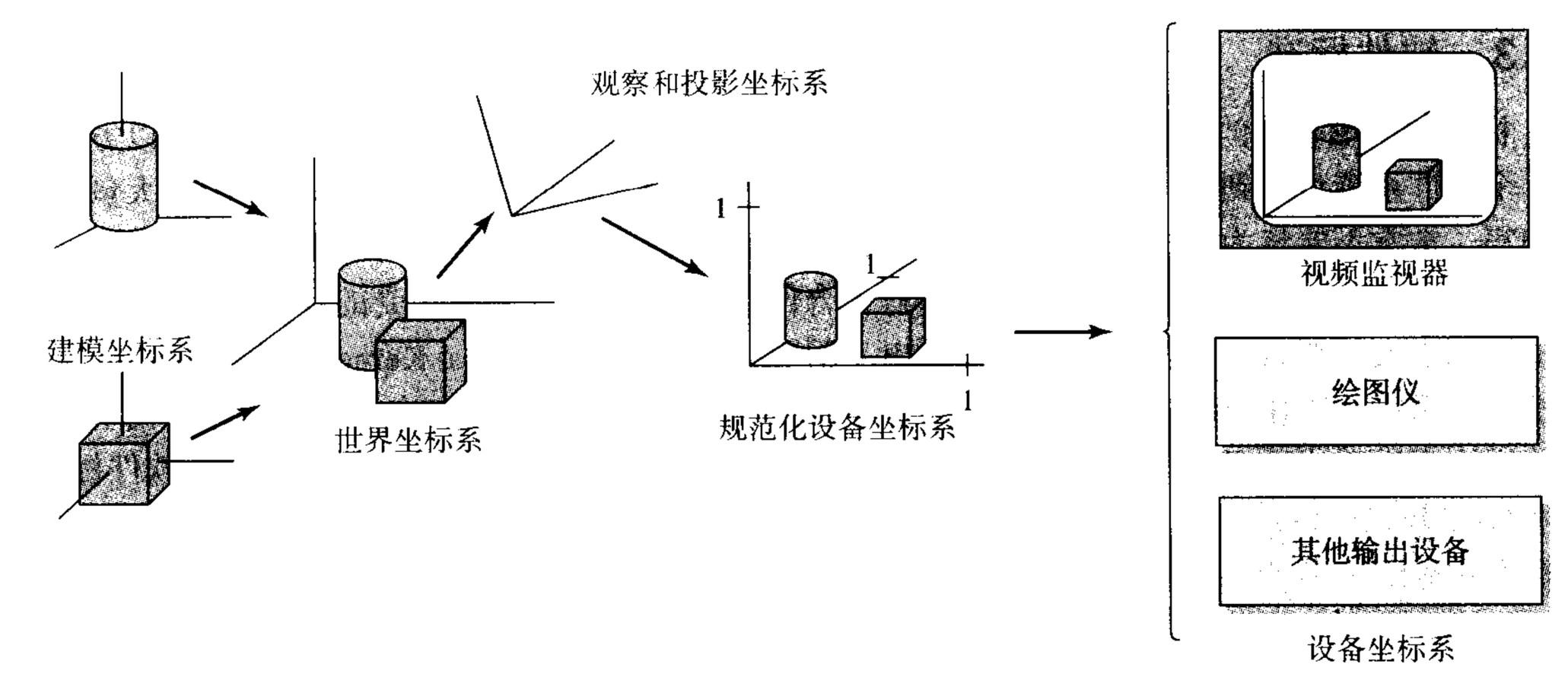


图 2.60　三维场景从建模坐标到设备坐标的变换序列。对象形状定义在单独的建模坐标系定义。然后该形体定位到总的世界坐标场景内。接着，世界坐标设定又变换到观察坐标系和投影坐标系，然后到规范化坐标系。最后一步，独立的设备驱动器将该场景的规范化坐标表示变换到输出设备，并进行显示

2.8.2　图形功能

通用图形软件包为用户提供建立和管理图形的各种功能。这些子程序可以按照它们是否处理输出、输入、属性、变换、观察、分割图形或一般的控制而进行分类。

图形的基本构造块称为**图形输出图元**（graphics output primitive）。它们包括字符串和几何成分，如点、直线、曲线、填充区域（通常为多边形）以及由彩色阵列定义的形状。此外，有些图形软件包提供对复杂形体（如球体、锥体和柱体）的显示函数。生成输出图元的函数提供了构造图形的基本工具。

属性（attribute）是输出图元的特性。也就是说，属性描述一个特定图元是怎样显示出来的。它们包括颜色设定、线型或文本类型以及区域填充图案等。

我们可以使用**几何变换**（geometric transformation）来改变场景中一个对象的大小、位置或方向。某些图形包给出一组函数实现**建模变换**（modeling transformation），将建模坐标系中给出的对象描述组织成场景。这些图形软件包通常提供描述复杂对象（如电子线路或自行车）的树形结构。另外一些软件包仅简单地提供几何变换函数，而将建模细节留给了程序员。

利用对象形状及其属性的描述函数构造场景之后，图形软件包将选定视图投影到输出设备。**观察变换**（viewing transformation）用来指定将要显示的视图、使用的投影类型以及在输出显示区域出现的范围。另有一些函数通过指定位置、大小和结构来管理屏幕显示范围。对于三维场景还要判定可见对象并应用光照条件。

交互式图形应用使用多类输入设备，如鼠标、数据板或操纵杆。**输入函数**（input function）用于控制和处理来自这些交互设备的数据流。

有些图形软件包也提供将一个图形描述分割成一组命名的组成部件的函数。另外有一些函数以各种方式管理这些图形部件。

最后，图形软件包常常包含许多事务性任务，如将显示屏清成指定颜色和初始化参数。我们可以将这类处理事务性任务的功能归入**控制操作**（control operation）类。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

2.8.3 软件标准

标准化图形软件的最主要目标是可移植性。当软件包按标准图形功能设计时，软件可以方便地从一个硬件系统移植到另一个，并且用于不同的实现和应用。如果没有标准，那么不经过大量的重新编写，常常不能将为一个硬件系统设计的程序移植到另一个系统。

国际组织和许多国家的标准化组织进行了合作，努力开发能被大家接受的计算机图形标准。在付出了相当大的努力后，最终在1984年推出了**图形核心系统**（Graphical Kernel System，GKS）。该系统成为国际标准化组织（International Standards Organization，ISO）和许多国家的标准化组织包括美国国家标准化组织（American National Standards Institute，ANSI）接受的第一个图形软件标准。虽然GKS最初的设计是一个二维图形软件包，但三维GKS扩展随后就开发出来。已制定出来并得到标准化组织批准的第二个图形软件标准是PHIGS（Programmer's Hierarchical Interactive Graphics Standard，程序员级的分层结构交互图形标准），它是对GKS的扩充。PHIGS提供了层次式对象建模、颜色设定、表面绘制和图形管理等功能。此后，PHIGS的扩充称为PHIGS+，用于提供PHIGS所没有的三维表面明暗处理功能。

随着GKS和PHIGS得到开发，SGI公司的图形工作站逐渐流行。这些工作站使用称为GL（Graphics Library）的函数集，GL很快成为图形界广泛使用的图形软件包。因此GL成为事实上的图形标准。GL函数为快速、实时绘制而设计，很快便扩展到其他硬件系统中。结果，作为GL的与硬件无关的版本，OpenGL在20世纪90年代早期就制定出来。这一图形软件包现在由代表许多图形公司和组织的**OpenGL结构评议委员会**（OpenGL Architecture Review Board）进行维护和更新。OpenGL函数库专为高效处理三维应用而设计，但它也能按 z 坐标为零的三维特例来处理二维场景描述。

图形函数定义为独立于任何程序设计语言的一组规范。**语言绑定**（language binding）则是为特定的高级程序语言而定义的。它给出该语言访问各种图形函数的语法。每一个语言绑定以最佳地使用有关的语言能力及处理好数据类型、参数传递和出错等各种语法问题为目标来定义。图形软件包在特定语言中的实现描述由国际标准化组织来制定。OpenGL的C和C++语言绑定也一样如此。Open GL的Ada及Fortran等语言绑定也已经问世。

在下面几章，我们把OpenGL的C和C++语言绑定作为讨论基本图形概念和图形软件包设计及应用的框架，并且使用C++语言的程序例子来给出OpenGL的应用及图形函数的实现算法。

2.8.4 其他图形软件包

已开发的计算机图形程序库有很多，有些提供通用的图形函数，有些则以专门应用或动画、虚拟现实或因特网图形等计算机图形学的特定应用方面为目标。

Open Inventor软件包给出一组用来描述场景的面向对象函数，其描述的场景通过OpenGL来显示。虚拟现实建模语言（Virtual-Reality Modeling Language，VRML）最初是Open Inventor的一个子集，可用来建立因特网上虚拟世界的三维模型。我们也可以使用为Java 3D语言开发的图形库来构造Web图形。还可以用Java 2D创建Java applet中的二维场景，或者使用Java 3D生成三维Web显示。Pixar公司的Renderman Interface可用来生成各种光照模型下的场景。最后，Mathematica、MatLab和Maple等另一类系统中也经常提供图形库。

2.9 OpenGL简介

OpenGL基本函数库用来描述图元、属性、几何变换、观察变换和进行许多其他的操作。如上节所指出的，OpenGL被设计成与硬件无关，因此输入和输出函数等许多操作均不包括在其基本库中。但在为OpenGL开发的辅助库中有输入和输出函数及许多附加函数。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

2.9.1　基本的 OpenGL 语法

OpenGL 基本库（也称为 OpenGL 核心库）中的函数名要以 gl 为前缀，且函数名中每一组成词的第一个字母要大写。下列例子给出了这种命名规范。

```
glBegin,    glClear,    glCopyPixels,    glPolygonMode
```

有些函数要求一个（或多个）变量用符号常量赋值，如参数名、参数的值或特定的模式。所以这些常量均以大写字母 GL 开头。另外，常量名中各组成词均采用大写，单词之间用下划线（_）分隔开。下面列出 OpenGL 函数使用的几百个符号常量中的几个例子。

```
GL_2D,    GL_RGB,    GL_CCW,    GL_POLYGON,    GL_AMBIENT_AND_DIFFUSE
```

OpenGL 函数也要求专门的数据类型。例如，OpenGL 函数的参数可以要求一个 32 位整数类型的值。但是，不同机器上的整数描述范围可能有所不同。OpenGL 采用专门的内置数据类型名来描述数据类型，例如

```
GLbyte,    GLshort,    GLint,    GLfloat,    GLdouble,    GLboolean
```

每个数据类型名以大写字母 GL 开头，名字中其余部分是用小写字母表示的标准数据类型名。

OpenGL 函数的某些变量可以采用数组赋值，从而列出一组数据的值。这是作为指向数组的指针来指定一组数值而不是作为显式变量指定该组数据中每一个数据的替代方法。指定 *xyz* 坐标值就是该方法的典型例子。

2.9.2　相关库

除 OpenGL 基本（核心）库之外，还有一些用于处理专门操作的附加库。**OpenGL 实用函数**（OpenGL Utility，GLU）提供了一些例程，可以设置观察和投影矩阵，利用线条和多边形近似法来描述复杂对象，使用线性近似法显示二次曲线和样条曲线，处理表面绘制操作，以及完成其他的复杂任务。每一个 OpenGL 实现中都包括 GLU 库，所有 GLU 函数名均用前缀 glu 开头。还有一个称为 Open Inventor 的基于 OpenGL 的面向对象工具包，它为交互式三维应用提供函数和预定义的对象形状。该工具包采用 C++ 编程。

为了使用 OpenGL 建立一个图形，必须首先在视频屏幕上设置显示窗口。它是一个屏幕上的简单矩形，图形将在其中显示。我们不能直接使用基本的 OpenGL 函数来创建显示窗口，因为该库中只有与设备无关的函数，且窗口管理操作依赖于所用的计算机。但是，有多个支持各种计算机上的 OpenGL 函数的窗口系统库。**OpenGL 的 X 窗口系统扩充**（OpenGL Extension to the X Window System，GLX）提供了一组以 glX 为前缀的函数。Apple 系统可使用 Apple GL（AGL）接口进行窗口管理操作，该库的函数名以 agl 为前缀。对于 Microsoft 的 Windows 系统，WGL 函数提供了窗口系统到 OpenGL 的接口，这些函数以 wgl 为前缀。Presentation Manager to OpenGL（PGL）是一个用于 IBM OS/2 的接口，它使用 pgl 作为库函数的前缀。**OpenGL 实用函数工具包**（OpenGL Utility Toolkit，GLUT）提供了与任意屏幕窗口系统进行交互的函数库。GLUT 库函数以 glut 为前缀，该库中也包含了描述与绘制二次和样条曲线及曲面的方法。

由于 GLUT 是一个与其他依赖于设备的窗口系统之间的接口，我们可以利用 GLUT 使得程序成为与设备无关的。关于 GLUT 最新版本的信息和源程序下载过程请参见 Web 页面：

```
http://reality.sgi.com/opengl/glut3/glut3.html
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

2.9.3 头文件

在我们所有的程序中，需要包含一个头文件以引入OpenGL核心库。在许多应用中，我们都需要GLU，需要包含头文件来引入窗口系统。例如，对于Microsoft的Windows系统，存取WGL函数的头文件是windows.h。该头文件必须列在OpenGL和GLU头文件之前，因为它包含了OpenGL库的Microsoft版本所需的宏。因此，源程序的开头几行为

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
```

但是，如果我们使用GLUT处理窗口管理操作，就不需要引入gl.h和glu.h，因为GLUT保证了它们的正确引入。因此，我们可以使用：

```
#include <GL/glut.h>
```

来代替OpenGL和GLU的头文件。我们也可以再引用gl.h和glu.h，但这将造成冗余且影响了程序的可移植性。

此外，我们总是要引入C++程序所需的头文件。例如：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

对应于ISO/ANSI新的C++标准，这些头文件称为cstdio、cstdlib和cmath。

2.9.4 使用GLUT进行显示窗口管理

我们从使用简化的最少操作来显示一个图开始。使用OpenGL实用库的第一步是初始化GLUT。该初始化函数也能处理任何命令行变量，但我们不需要在第一个例子程序中使用参数。完成GLUT初始化的语句是

```
glutInit (&argc, argv);
```

接着，需要说明的是显示窗口在创建时要给定一个标题。这是用下列语句实现的：

```
glutCreateWindow ("An Example OpenGL Program");
```

这里的单一变量可以是用做显示窗口标题的任意字符串。

下面，我们需要指定显示窗口中要显示什么内容。为此，我们用OpenGL函数创建一个图并将图的定义传递给GLUT函数glutDisplayFunc，即将图赋给显示窗口。作为一个例子，假定我们在称为lineSegment的过程中已经有了线段的OpenGL描述程序。则调用下列函数就将线段描述送到显示窗口。

```
glutDisplayFunc (lineSegment);
```

但是显示窗口还未出现在屏幕上。我们需要另一个GLUT函数来完成窗口处理操作。在执行下列语句后，所有已创建的显示窗口连同其中的图形内容将被激活。

```
glutMainLoop ();
```

该函数必须是程序中的最后一个。它显示初始图形并使程序进入检查鼠标或键盘等设备输入的无穷循环之中。我们的第一个例子不是交互式的，所以程序仅仅显示其中的图形直到显示窗口关闭。在后面的几章里，我们将考虑怎样修改OpenGL程序，使之能处理交互输入。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

尽管我们创建的显示窗口有默认的位置和大小，但还是可以使用另外的GLUT函数来设定这些参数。glutInitWindowPosition可用来给出显示窗口左上角的初始位置。该位置用以屏幕左上角为原点的整数坐标来表示。例如，下面的语句指定了显示窗口左上角应该在屏幕左边界向右50个像素、屏幕上边界之下100像素的位置上。

```
glutInitWindowPosition (50, 100);
```

类似地，glutInitWindowSize函数用来设定显示窗口的初始宽度和高度的像素数。因此，要指定一个宽度为400像素、高度为300像素（如图2.61所示）的显示窗口，相应的语句为

```
glutInitWindowSize (400, 300);
```

在显示窗口已出现在屏幕上之后，我们可重新设定它的位置和大小。

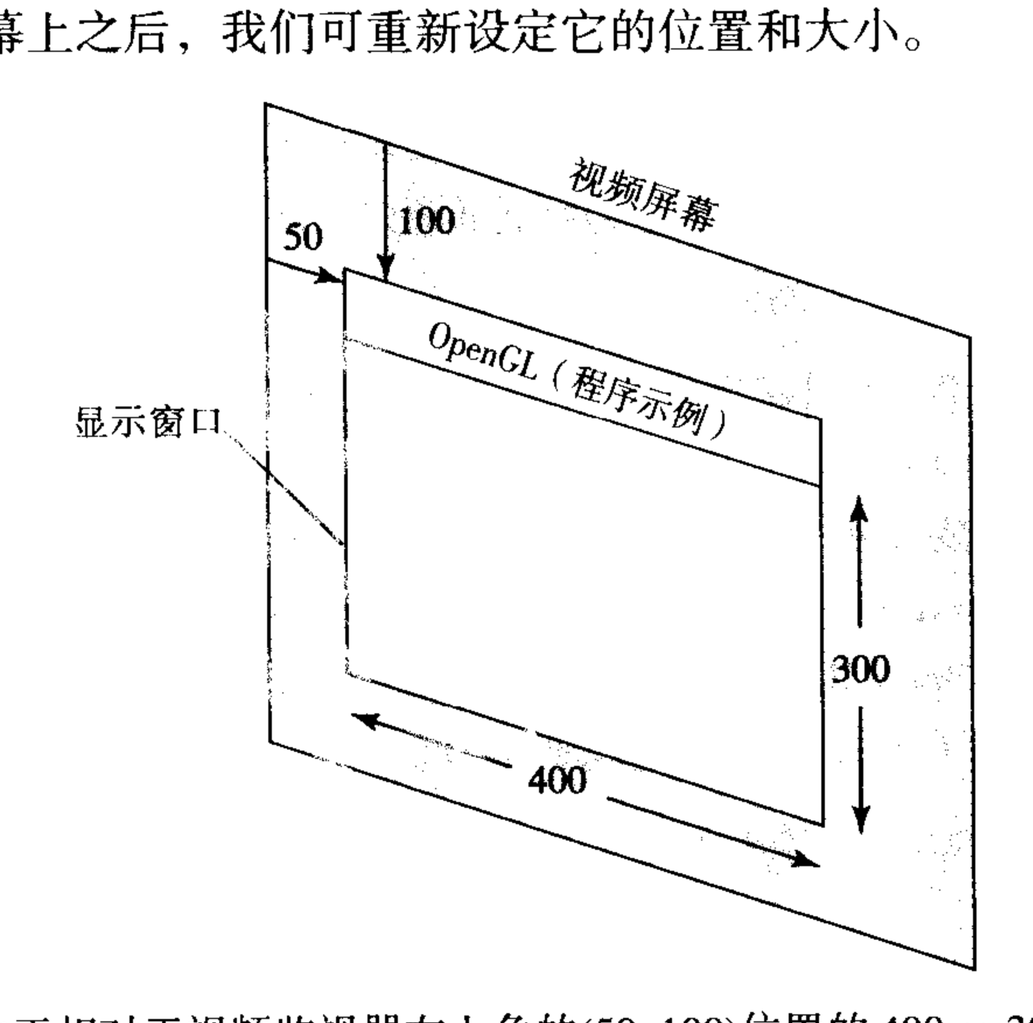


图 2.61　位于相对于视频监视器左上角的(50, 100)位置的 400 × 300 显示窗口

我们还可以使用glutInitDisplayMode函数来设定显示窗口的缓存和颜色模型等选项。该函数的变量使用符号化GLUT常量来赋值。例如，下面的命令指出显示窗口使用单个缓存且使用RGB（红、绿、蓝）颜色模型来设定颜色值。

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
```

传送给该函数的常量值利用逻辑或操作组合起来。实际上，单缓存和RGB颜色模型是默认的选项。但现在使用该函数是为了强调要使用这些选项来设定我们的显示。后面将会更详细地讨论颜色模型及动画应用的双缓存和观察三维场景的参数选项等其他显示选项。

2.9.5　完整的 OpenGL 程序

在给出构成一个完整程序的所有部分之前还是有一些任务需要完成。对于显示窗口，我们可以选择背景颜色。我们需要组织一个过程来包含创建显示的图形所必需的 OpenGL 函数。

要像图 2.61 那样将显示窗口的背景颜色设定为白色，可以使用语句：

```
glClearColor (1.0,  1.0,  1.0,  0.0);
```

该函数的前面三个变量将红、绿、蓝三个颜色分量设定为 1.0。这样就得到了白色的显示窗口。如果不是 1.0，而是将这些颜色分量都设定为 0.0，则得到黑色的背景。如果红、绿、蓝三分量的每

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

一个设定为0.0到1.0之间的中间值，将得到某种灰色。glClearColor函数的第四个参数称为指定颜色的α（alpha）值。α值的一个用途是作为“调和”参数。在激活OpenGL调和参数时，α值用来为两个重叠对象确定结果颜色。α值为0.0表示完全透明的对象，而α值为1.0表示不透明的对象。调和操作暂时不会使用，因此α值与我们前面提到的程序无关。现在，我们简单设定α值为0.0。

尽管glClearColor命令将某颜色赋给显示窗口，但它不能让显示窗口在屏幕上出现。要使赋值的窗口得到显示，必须引入下面的OpenGL函数。

```
glClear (GL_COLOR_BUFFER_BIT);
```

变量GL_COLOR_BUFFER_BIT是一个OpenGL符号常量，用来指定它是颜色缓存（刷新缓存）中的位值，该缓存将使用glClearColor函数中指定的值来设定。（在以后的几章中将讨论其他缓存。）

除了设定显示缓存的背景色，还可以为要显示的场景中的对象选择各种颜色模型。对于最初的程序设计例子，我们简单地把对象颜色设定为红色而把各种颜色选项的讨论放到第4章。

```
glColor3f (1.0,  0.0,  0.0);
```

glColor函数的后缀3f表示我们在指定三个RGB颜色分量时使用浮点数。这些值必须在0.0到1.0的范围内，这里设定R = 1.0而G = B = 0.0。

在第一个程序中，我们要显示一条简单的二维线段。为此，需要告诉OpenGL怎样将图形投影到显示窗口中，因为在OpenGL中把生成二维线段看作为生成三维线段的特例。因此，尽管我们只要生成很简单的二维线段，OpenGL还是采用完整的三维观察操作来处理该图形。我们可以使用下面两个函数来设置投影类型（模式）和其他观察参数。

```
glMatrixMode (GL_PROJECTION);
gluOrtho2D (0.0,  200.0,  0.0,  150.0);
```

这表示使用正投影将世界坐标系二维矩形区域的内容映射到屏幕上，区域的x坐标值从0.0到200.0，y坐标值从0.0到150.0。只要是在该矩形内定义的对象，都会出现在显示窗口中。任何在坐标范围外的内容都不会显示出来。因此，GLU函数gluOrtho2D定义了显示窗口以(0.0, 0.0)为左下角，以(200.0, 150.0)为右上角。由于我们仅仅描述了一个二维对象，正投影只是将我们前面定义的图形“贴”到显示窗口中。现在我们使用与显示窗口具有一样纵横比的世界坐标矩形，从而使图形不产生变形。后面将考虑如何在不依赖显示窗口描述的情况下保持纵横比。

最后，要调用合适的函数来建立线段。下面的程序定义了一个从整数笛卡儿端点坐标(180, 15)到(10, 145)的二维直线段。第3章将给出这些函数以及用来生成图元的其他OpenGL函数的详细解释。

```
glBegin (GL_LINES);
   glVertex2i (180, 15);
   glVertex2i (10, 145);
glEnd ();
```

现在，我们已经可以将各部分组合起来。下面的OpenGL程序按三个过程来组织。将所有初始化和有关的一次性设定参数的函数放在过程init中。要显示图形的几何描述放在过程lineSegment中，该过程将由GLUT函数glutDisplayFunc调用。main过程包含设定显示窗口及将线段送到屏幕的GLUT函数。图2.62给出了显示窗口及由该程序生成的红色线段。

```
#include <GL/glut.h>      // (or others, depending on the system in use)

void init (void)
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
{
    glClearColor (1.0, 1.0, 1.0, 0.0);  // Set display-window color to white.

    glMatrixMode (GL_PROJECTION);       // Set projection parameters.
    gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void lineSegment (void)
{
    glClear (GL_COLOR_BUFFER_BIT);  // Clear display window.

    glColor3f (1.0, 0.0, 0.0);      // Set line segment color to red.
    glBegin (GL_LINES);
        glVertex2i (180, 15);       // Specify line-segment geometry.
        glVertex2i (10, 145);
    glEnd ();

    glFlush ();     // Process all OpenGL routines as quickly as possible.
}

void main (int argc, char** argv)
{
    glutInit (&argc, argv);                          // Initialize GLUT.
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);    // Set display mode.
    glutInitWindowPosition (50, 100); // Set top-left display-window position.
    glutInitWindowSize (400, 300);    // Set display-window width and height.
    glutCreateWindow ("An Example OpenGL Program"); // Create display window.

    init ();                          // Execute initialization procedure.
    glutDisplayFunc (lineSegment);    // Send graphics to display window.
    glutMainLoop ();                  // Display everything and wait.
}
```

过程 lineSegment 的最后是函数 glFlush，我们还未讨论它。该函数强制由计算机系统存放在不同位置的缓存中的 OpenGL 函数执行，其位置依赖于 OpenGL 的实现。例如在繁忙的网络中，可能因处理某些缓存而出现延缓现象，但 glFlush 的调用将强制清空所有缓存来处理 OpenGL 函数。

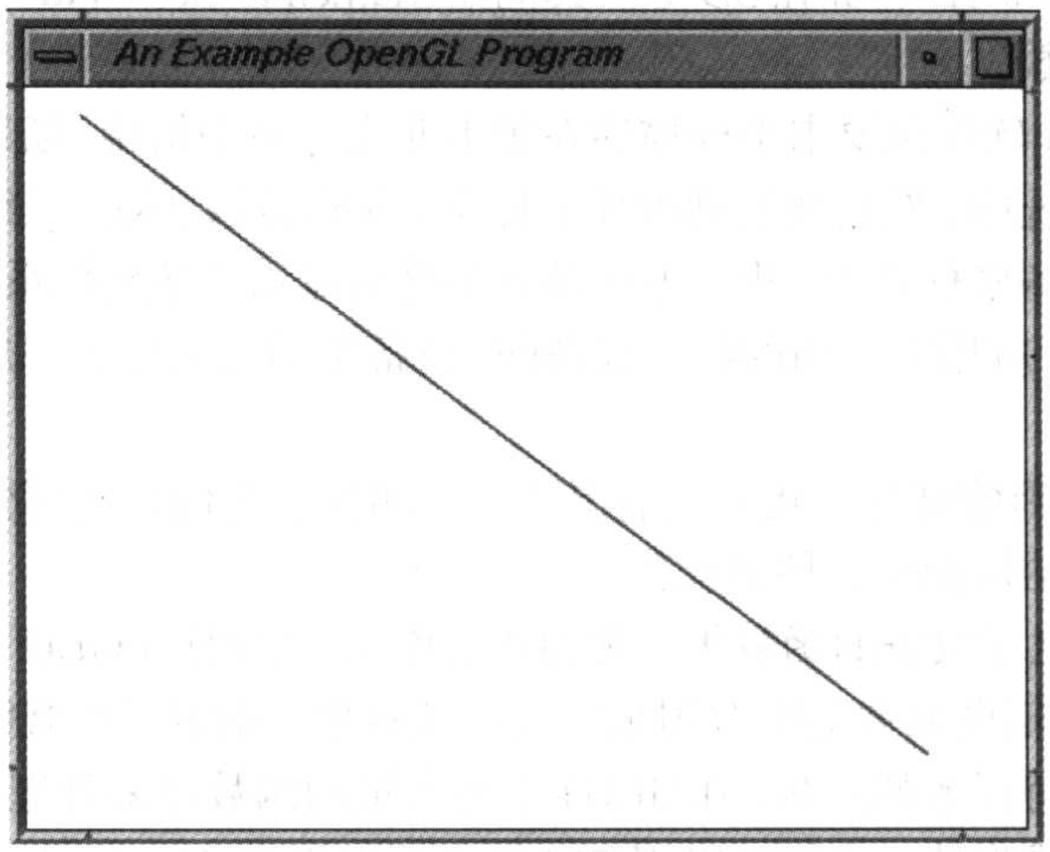


图 2.62　显示窗口和由示例程序生成的线段

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

描述图形的过程 lineSegment 称为一个显示回调函数（display callback function）。该过程由 glutDisplayFunc 作为在显示窗口需要重新显示时引入的函数来“注册”。例如，显示窗口移动时会出现这种情况。在后面几章中，我们将看到其他类型的回调函数及辅助的对它们注册的 GLUT 函数。一般情况下，OpenGL 程序组织成一组在一定行为发生时的回调函数的集合。

2.10 小结

在这一章中，我们简要介绍了计算机图形系统的主要硬件和软件的配置。硬件部分有视频监视器、硬拷贝输出设备、各种输入设备及与虚拟环境交互的部件。有些软件系统（如 CAD 软件包和绘画程序等）为特定应用而设计；另一些软件系统则提供通用的图形函数库，用于 C++ 等程序设计语言中为各种应用生成图形。

基于电视技术的光栅刷新监视器是主流的显示设备。光栅系统使用帧缓存来存储每一屏幕位置（像素）的颜色值。CRT 电子束从上到下扫描屏幕每一行时从帧缓存（也称为刷新缓存）中取出这些信息，从而在屏幕上画出图形。老式的向量显示器通过在指定的端点之间画出直线段来构成图形。图形以一组画线指令的形式存放。

现在已经出现了许多其他的视频显示设备。尤其是平板显示技术正在快速发展，而这些设备现在已应用于包括桌面型和膝上型计算机的各种系统中。等离子和液晶设备是平板显示器的两个例子。其他的技术有三维和立体观察系统等。虚拟现实系统使用立体显示头套或标准的视频监视器。

对于图形输入，可以选择的设备很多。键盘、按钮盒和旋钮用于输入文字、数值或程序选项。鼠标是最流行的“指点”设备，而跟踪球、空间球、控制杆、光标键和拇指转轮也用来对屏幕光标进行定位。在虚拟现实环境中经常使用数据手套。其他一些输入设备有图像扫描仪、数字化仪、触摸板、光笔和语音系统。

图形工作站的硬拷贝设备包括标准打印机和绘图仪，还有能生成幻灯片胶片和影片输出的设备。打印方法包括点阵、激光、喷墨、静电和热转印。图表可用喷墨式绘图仪或混合打印－绘图功能的设备生成。

ISO 和 ANSI 开发和批准的标准图形程序设计软件包有 GKS、3D GKS、PHIGS 和 PHIGS+。另外已经成为标准的有 GL 和 OpenGL。还有许多图形库可用于程序设计语言，包括 Open Inventor、VRML、RenderMan、Java 2D 和 Java 3D 等。其他如 Mathematica、MatLab 和 Maple 等系统通常提供一组图形程序设计函数。

通常，图形程序设计软件包要求坐标描述在笛卡儿坐标系中给出。场景的每个对象可定义在单独的建模笛卡儿坐标系统中，然后映射到世界坐标系，并构造该场景。三维对象从世界坐标系投影到二维平面的规范化设备坐标系中，然后再变换到最终的显示设备坐标系。从建模坐标到规范化设备坐标的变换，是独立于应用中使用的特定设备的。设备驱动器则用于将规范化坐标变换到整数设备坐标。

图形编程软件包的函数可分为几类：输出图元、属性、几何和建模变换、观察变换、结构操作、输入函数、图形－结构操作和控制操作。

OpenGL 系统由独立于设备的函数集（称为核心库）、实用库（GLU）和实用工具包（GLUT）三部分组成。在 GLU 的辅助函数集中有用来生成复杂对象、指定二维观察应用的参数、处理表面绘制操作和完成其他支持任务的函数。在 GLUT 中有大量的函数用来管理显示窗口、与屏幕窗口系统的交互及生成某些三维形体。我们可以用 GLUT 也可以用 GLX、Apple GL、WGL 或其他专用系统的软件包与任意计算机系统接口。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

参考文献

电子显示器的一般处理可参见Tannas(1985)和Sherr(1993)。平板设备在Depp and Howard(1993)中讨论。关于光栅图形结构的其他信息可参见 Foley, van Dam, Feiner, and Hughes (1990)。三维和立体显示的讨论请参见 Johnson(1982)和 Grotch(1983)。头套显示器和虚拟现实环境在 Chung, et al. (1989)中讨论。

OpenGL 的标准信息源请参见 Woo, Neider, Davis, and Shreiner(1999)和 Shreiner(2000)。Open Inventor在Wernecke(1994)中讨论。VRML的讨论请参见McCarthy and Descartes(1998)。Upstill(1989)展示了RenderMan。Knudsen(1999)、Hardy(2000)和Horstmann and Cornell(2001)给出了使用Java 2D进行图形编程的例子。Sowizral, Rushforth, and Deering(2000)、Palmer(2001)、Selman(2002)和Walsh and Gehringer(2002)探讨了使用 Java 3D 进行图形编程。

关于 PHIGS 和 PHIGS+ 的信息请参见 Howard, Hewitt, Hubbold, and Wyrwas(1991)、Hopgood and Duce(1991)、Gaskins(1992)和 Blake(1993)。二维 GKS 标准及图形标准的进展可参见 Hopgood, et al.(1983)。GKS 的其他信息参考见 Enderle, Kansy, and Pfaff(1984)。

练习题

2.1　列出下列显示技术的工作特性：光栅刷新系统、向量刷新系统、等离子体显示板和LCD。

2.2　列出练习题 2.1 所描述的每种显示技术适合的某些应用。

2.3　如果使用读者系统中的视频监视器，请确定 x 和 y 方向上的分辨率（每厘米像素数）。确定纵横比，并说明该系统怎样保持对象的相对比例。

2.4　考虑三个不同的光栅系统，分辨率依次为 640 × 480、1280 × 1024、2560 × 2048。如果每个像素存储12位，那么这些系统各需要多大的帧缓存（字节数）？如果每个像素存储24 位，这些系统各需要多大的存储量？

2.5　假设 RGB 光栅系统的设计采用 8 英寸 × 10 英寸的屏幕，每个方向的分辨率为每英寸 100 个像素。如果每个像素占6位，并存放在帧缓存中，则帧缓存需要多大的存储量（字节数）？

2.6　如果每秒能传输 10^5 位，每个像素有 12 位，则装入 640 × 480 的帧缓存需多长时间？若每个像素有 24 位，分辨率为 1280 × 1024，用同样的传输速率装入帧缓存要多长时间？

2.7　假设计算机字长为 32 位，传输速率为 1 mip（每秒 100 万条指令）。如果使用 300 dpi（每英寸点数）的激光打印机，那么在页面大小为 8.5 英寸 × 11 英寸时，要填满帧缓存需要多长时间？

2.8　考虑分辨率为 640 × 480 和 1280 × 1024 的两个光栅系统。若显示控制器刷新屏幕的速率为每秒 60 帧，那么在各个系统中，每秒应访问多少像素？各个系统访问每个像素的时间是多少？

2.9　假设视频监视器的显示区域为 12 英寸 × 9.6 英寸。如果分辨率是 1280 × 1024，纵横比为 1，屏幕上每一点的直径是多少？

2.10　有一种光栅系统的分辨率为 1280 × 1024，刷新速率为每秒 60 帧。如果在屏幕刷新期间，横向扫描每行像素，那么需要多长时间？

2.11　考虑一个非隔行光栅监视器，分辨率为 $n \times m$（m 扫描行，每扫描行有 n 个像素），刷新速率为每秒 r 帧，水平回扫时间为 t_{horiz}，而垂直回扫时间为 t_{vert}。那么电子束回扫时的开销占每帧总刷新时间的多少（分数）？

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

2.12 一个非隔行光栅系统，分辨率为1280 × 1024，刷新频率为60 Hz，水平回扫时间是5 μs，垂直回扫时间是500 μs，则电子束回扫的开销占每帧总刷新时间的多少（分数）？

2.13 假设某种全彩色（每像素24位）RGB光栅系统有512 × 512的帧缓存，那么可以使用多少种不同的颜色选择（亮度级）？在任意时刻可以显示多少种不同的颜色？

2.14 使用变焦镜的三维监视器与立体感系统相比的优点和缺点是什么？

2.15 列出一般在虚拟现实系统中使用的不同的输入和输出部件。然后说明用户如何与使用不同的输出设备（如二维和立体感监视器）显示的虚拟场景进行交互？

2.16 说明虚拟现实系统如何用于设计应用。虚拟现实系统还可用于其他什么应用？

2.17 列出大屏幕显示的几种应用。

2.18 说明为程序员设计的通用图形系统和作为专门应用的、为建筑设计而设计的软件包有什么不同？

2.19 说明OpenGL核心库、OpenGL实用库和OpenGL实用工具包之间的区别。

2.20 什么命令可用来设定OpenGL显示窗口的颜色为浅灰色？什么命令可用来设定OpenGL显示窗口的颜色为黑色？

2.21 列出将OpenGL显示窗口左下角设定到像素位置(200, 200)、窗口宽度为100像素且高度为75像素的语句。

2.22 解释术语“OpenGL显示回调函数”的含义。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第3章 输出图元

用于图形应用的通用软件包称为计算机图形应用编程接口（CG API），它提供可以在C++等程序设计语言中用来创建图形的函数库。如2.8节所指出的，函数库可以分成几种类型。创建图形时最先要做的一件事就是要描述显示场景的组成部分。图形的组成部分可以是树木和地形、家具和墙壁、商店铺面和街景、汽车和广告牌、原子和分子或者星星和银河。对于每一类场景，要描述每一对象的结构及其在场景中的坐标位置。图形软件包中用来描述各种图形元素的函数称为**图形输出原语**（graphics output primitive），或简称为**图元**（primitive）。描述对象几何要素的输出图元一般称为**几何图元**。点的定位和直线段是最简单的几何图元。图形软件包中包含的其他几何图元有圆和其他二次曲线、二次曲面、样条曲线和曲面及多边形填色区域。多数图形系统还提供显示字符串的某些函数。在选定的坐标系中指定一个图形的几何要素后，输出图元投影到与该输出设备显示区域对应的二维平面上，并扫描转换到帧缓存的整数像素位置。

本章将介绍OpenGL中的输出图元并讨论实现输出图元的设备级算法。对图形库实现算法的探讨将使我们了解这些图形软件包的功能，并且可以理解这些函数是如何工作的，也许会发现如何改进它们，以及如何自行为某些特定应用实现图形函数。计算机图形学方面的研究不断地为特定应用（如因特网图形等）发现新的和改进的技术，并且开发更快、更逼真的通用图形显示方法。

3.1 坐标系统

为了描述图形，必须首先确定一个称为世界坐标系的合适的二维或三维笛卡儿坐标系。接着通过给出世界坐标系中的位置等几何描述来定义图形中的对象。例如，通过两个端点定义一条直线段，通过一组顶点位置定义一个多边形。这些坐标位置与该对象的颜色、**坐标范围**（coordinate extent）即对象坐标的x、y、z最小/最大值等其他信息一起存储在场景描述中。坐标范围也称为对象的**包围盒**（bounding box）。对于二维图形来说，坐标范围也称为对象的**包围矩形**（bounding rectangle）。通过将场景信息传送给观察函数、由观察函数识别可见面、将对象映射到视频监视器上来实现对象的显示。扫描转换过程将颜色值等场景信息保存到帧缓存的相应位置，从而在输出设备上显示场景中的对象。

3.1.1 屏幕坐标

视频监视器上的位置使用与帧缓存中的像素位置相对应的整数**屏幕坐标**（screen coordinate）进行描述。像素的坐标值给出扫描行号（y值）和列号（扫描行的x值）。屏幕刷新等硬件处理一般从屏幕的左上角开始对像素进行编址。从屏幕最上面的0行到屏幕最下面的某整数值y_{max}行对扫描行进行编号，每一行中像素位置从左到右、从0到x_{max}进行编号。但是，使用软件命令可以按照任何方式设定屏幕位置的参考系统。例如，我们可以设定屏幕区域左下角为原点，用整数坐标（参见图3.1）或非整数笛卡儿坐标来描述图形。描述场景几何要素的坐标值由观察函数转换为帧缓存中的整数像素位置。

图元的扫描转换算法使用定义的坐标描述来确定要显示像素的位置。例如，给定一直线段的两个端点，其显示函数必须计算出两端点间位于线段上所有像素的位置。由于一个像素位置占有屏幕

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

上的一个有限范围，因此实现算法必须考虑像素的有限大小。目前，我们假设每一整数屏幕位置代表像素区域的中心。（我们将在 3.13 节考虑其他的像素编址方法。）

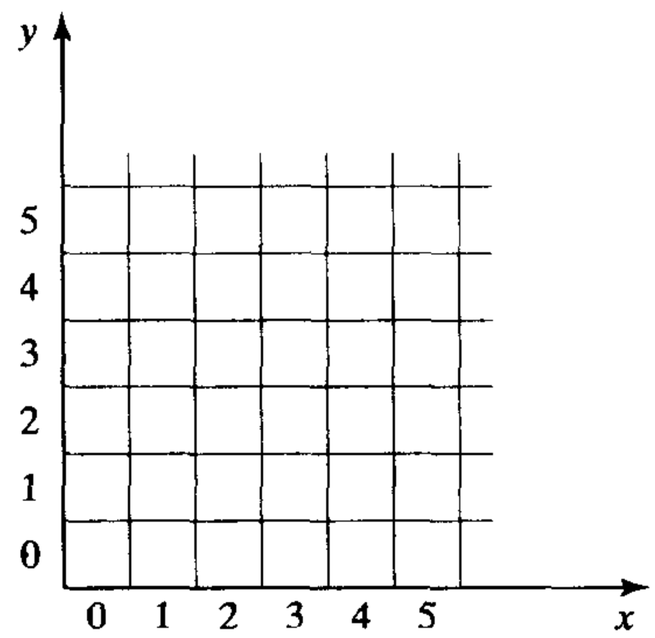


图 3.1 相对于屏幕区域左下角指定的像素位置

一旦确定了一个对象的像素位置，必须将合适的颜色值存入帧缓存。为此，我们要使用一个底层函数：

```
setPixel (x, y);
```

该函数将当前颜色设定值存入帧缓存的整数坐标位置(x, y)处，该位置相对于屏幕坐标原点而选定。有时我们也希望获得一个像素位置的当前帧缓存设置。这可使用下列底层函数来获得。

```
getPixel (x, y, color);
```

在这一函数中，参数 color 得到一个与存储在位置(x, y)的像素中的 RGB 组合对应的整数值。

对于二维图形来说，仅需在(x, y)位置指定颜色值；但是对于三维图形来说，还需要其他的屏幕坐标信息。这时，屏幕坐标按三维值来存储，第三维表示对象位置相对于观察位置的深度。在二维场景中，深度值均为 0。

3.1.2 绝对和相对坐标描述

到目前为止，我们讨论的坐标均为**绝对坐标**（absolute coordinate）值。这表示指定的值是所在坐标系统中的实际位置。

然而，有些图形软件包还允许使用**相对坐标**（relative coordinate）来描述位置。该方法在许多图形应用中很有用，比如用笔式绘图仪、艺术家绘画系统进行绘图及出版和印刷应用的图形软件包。使用这一方法，我们可以使用从离开最后一次引用的位置（称为**当前位置**，current position）的位移量来指定坐标位置。例如，如果位置(3, 8)是应用程序刚刚引用的位置，则相对坐标描述(2, -1)与绝对位置(5, 7)相对应。有一个函数专门用来在指定任何图元坐标前设定当前位置。在描述一串首尾相连的直线段场景时，我们可以在建立开始位置后仅给出一串相对坐标（位移）。图形系统中会给出指定定位中使用相对坐标还是绝对坐标的选项。在此后的讨论中，除非特别声明，我们假定都使用绝对坐标。

3.2 OpenGL 中指定二维世界坐标系统

第一个示例程序（参见 2.9 节）介绍了 gluOrtho2D 命令，我们可以利用该命令设定一个二维笛卡儿坐标系。该函数的变量是指定显示图形 x 和 y 坐标范围的四个值。由于 gluOrtho2D 函数指定正交投影，因此我们也要确定坐标值放进了 OpenGL 投影矩阵中。此外，我们可以将世界坐标范围设

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

定前的投影矩阵定义为一个单位矩阵。这样可保证坐标值不会受以前设置的投影矩阵的影响。因此，对于最初的二维例子，我们可以通过下列语句定义屏幕显示窗口的坐标系统。

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
gluOrtho2D (xmin, xmax, ymin, ymax);
```

如图 3.2 所示，显示窗口将被指定为其左下角位于坐标(x_{min}, y_{min})处，右上角位于坐标(x_{max}, y_{max})处。

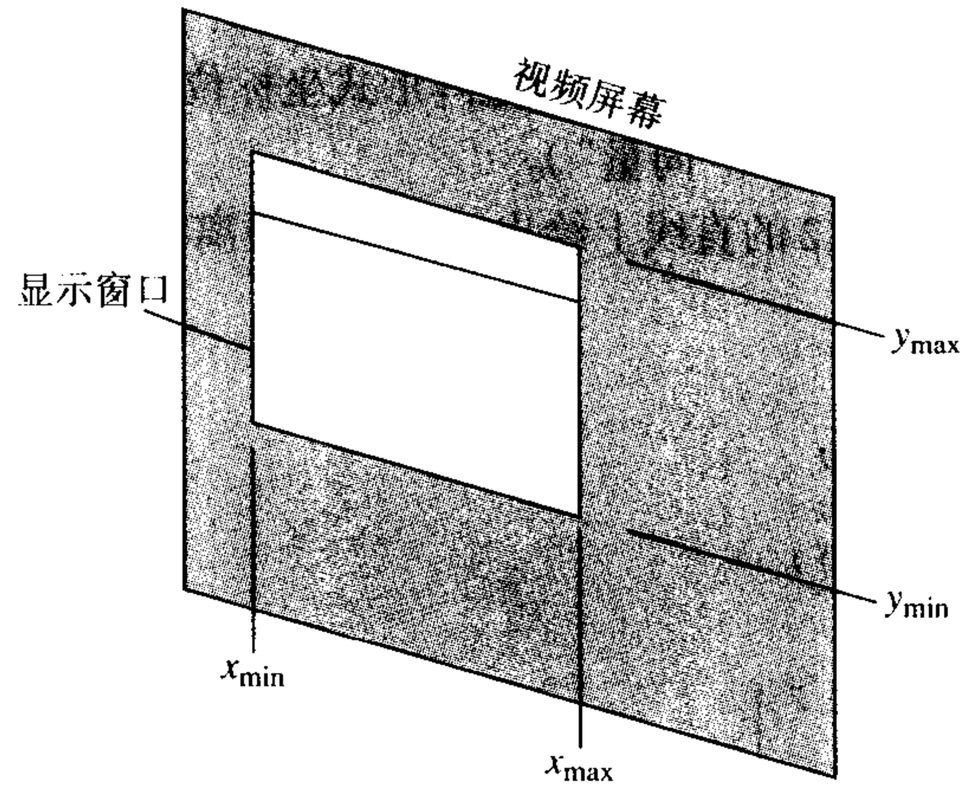


图 3.2　gluOrtho2D 函数指定的显示窗口的世界坐标范围

我们随后可使用 gluOrtho2D 语句描述的坐标系统来指定一个或多个要显示的图元。如果一个图元的坐标范围完全在显示窗口的坐标范围内，则该图元将完整地显示出来。否则，仅仅在显示窗口坐标范围内的图元部分被显示。同样，在建立图形的几何描述时，所有 OpenGL 图元的位置必须用 gluOrtho2D 函数定义的坐标系统中的绝对坐标给出。

3.3　OpenGL 画点函数

要描述一个点的几何要素，我们只需在世界坐标系中指定一个位置。然后该坐标位置和场景中已有的其他几何描述一起传递给观察函数。除非指定其他属性值，OpenGL 图元按默认的大小和颜色来显示。默认的图元颜色是白色，而默认的点大小等于一个屏幕像素大小。

使用下面的 OpenGL 函数可指定一个点位置的坐标值：

```
glVertex* ( );
```

这里的星号（*）表示该函数要有后缀码。这些后缀码用来指明空间尺寸、用做坐标值的数据类型和可能的向量形式坐标描述。在 glBegin 函数和 glEnd 函数之间必须有一个 glVertex 函数。glBegin 函数的变量用来指定要显示的输出图元的类型，而 glEnd 函数没有变量。对于点的绘制，glBegin 函数的变量是符号常量 GL_POINTS。因此，一个点位置的 OpenGL 描述形式是

```
glBegin (GL_POINTS);
   glVertex* ( );
glEnd ( );
```

尽管术语顶点（vertex）严格地代表一个多边形的“角”点、一个角两边的交点、椭圆和其主轴的交点或几何结构中其他类似的坐标位置，OpenGL 中的 glVertex 函数可用于描述任意一点的位置。这样，使用一个简单的函数来描述点、线段和多边形，而更多地使用多边形面片来描述场景对象。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

OpenGL 中的坐标位置可以有二维、三维或四维。glVertex 的后缀为 2、3 或 4 表示其坐标位置的维数。四维描述意味着齐次坐标（homogeneous-coordinate）表示，其中的齐次参数 h（第四维坐标）是笛卡儿坐标值的比例因子。齐次坐标表示对利用矩阵形式表达变换操作很有用，第 5 章将对其进行详细讨论。由于 OpenGL 将二维作为三维的特殊情况来处理，任意(x, y)坐标描述等同于 $h = 1$ 时的$(x, y, 0)$。

我们需要指出在坐标的数值描述中使用什么数据结构。这由 glVertex 函数的第二个后缀来完成。用于指定数值数据类型的后缀是：i（整数）、s（短整数）、f（浮点数）和 d（双精度浮点数）。最后，glVertex 中可以使用显式的坐标值或引入矩阵形式坐标位置的单个变量。如果使用矩阵形式坐标位置，则需要第三个后缀码：v（“向量”）。

在下面的例子中，在斜率为 2 的直线上绘出了三个等距离的点（参见图 3.3）。坐标用整数对给出。

```
glBegin (GL_POINTS);
  glVertex2i (50, 100);
  glVertex2i (75, 150);
  glVertex2i (100, 200);
glEnd ( );
```

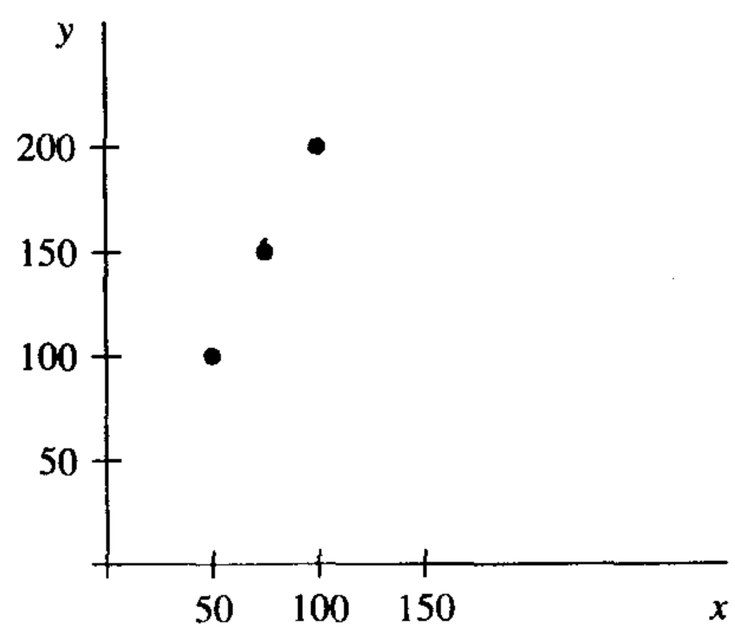


图 3.3 利用 glBegin (GL_POINTS)生成的三个点的显示

换一种方法，我们可以将前面这些点的坐标值以矩阵形式描述：

```
int point1 [ ]= {50, 100};
int point2 [ ]= {75, 150};
int point3 [ ]= {100, 200};
```

且调用 OpenGL 函数来绘出这三个点：

```
glBegin (GL_POINTS);
  glVertex2iv (point1);
  glVertex2iv (point2);
  glVertex2iv (point3);
glEnd ( );
```

下面是在三维世界坐标系中描述两个点位置的例子。这里按显式浮点数方式给出坐标。

```
glBegin (GL_POINTS);
  glVertex3f (-78.05, 909.72, 14.60);
  glVertex3f (261.91, -5200.67, 188.33);
glEnd ( );
```

我们还可以为各种维数中描述的点位置定义 C++ 类或结构（struct）。例如

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
class wcPt2D {
public:
   GLfloat x, y;
};
```

有了这一类定义，我们可以使用下列语句描述一个二维世界的坐标点位置：

```
wcPt2D pointPos;

pointPos.x = 120.75;
pointPos.y = 45.30;
glBegin (GL_POINTS);
   glVertex2f (pointPos.x, pointPos.y);
glEnd ( );
```

然后我们可以在C++过程中使用OpenGL画点函数来实现setPixel命令。

3.4 OpenGL画线函数

图形软件包一般都提供一个描述一条或多条直线段的函数，其中每一直线段由两个端点坐标位置定义。在OpenGL中，和选择一个点位置一样，我们使用glVertex函数选择单个端点的坐标位置。我们使用一对glBegin/glEnd来引入一串端点位置。有三个OpenGL符号常量可用于指定如何把这一串端点位置连接成一组直线段。默认情况下，每一符号常量显示白色实线。

使用图元线常量GL_LINES可连接每一对相邻端点而得到一组直线段。通常，这会导致一组未连接的线段，除非重复某些坐标位置。如果只描述了一个端点则什么也不会显示，如果列出的端点数为奇数则最后一个端点不被处理。例如，如果我们有5个坐标位置，标成p1到p5，每一个用二维数组表示，则下列程序能生成图3.4(a)。

```
glBegin (GL_LINES);
   glVertex2iv (p1);
   glVertex2iv (p2);
   glVertex2iv (p3);
   glVertex2iv (p4);
   glVertex2iv (p5);
glEnd ( );
```

这样，我们在第一和第二坐标位置之间得到一条直线段并在第三和第四位置间得到另一条直线段。此时，指定的端点数为奇数，因此最后一个坐标位置被忽略。

使用OpenGL的图元常量GL_LINE_STRIP，我们可获得折线（polyline）。此时，显示从第一个端点到最后一个端点之间一组首尾相连的线段。第一条线段在第一端点和第二端点之间显示；第二条线段在第二端点和第三端点之间显示；依次进行，直到最后一个端点。如果我们不列出至少两个坐标位置则什么也不显示。使用上例中的5个坐标位置，我们用下列程序显示图3.4(b)：

```
glBegin (GL_LINE_STRIP);
   glVertex2iv (p1);
   glVertex2iv (p2);
   glVertex2iv (p3);
   glVertex2iv (p4);
   glVertex2iv (p5);
glEnd ( );
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第三个OpenGL图元常量是生成封闭折线的GL_LINE_LOOP。它在前面的折线基础上再增加了一条直线段，因此折线序列中最后一个端点与第一个端点相连接。图3.4(c)给出了使用这一线选项的显示。

```
glBegin (GL_LINE_LOOP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd ( );
```

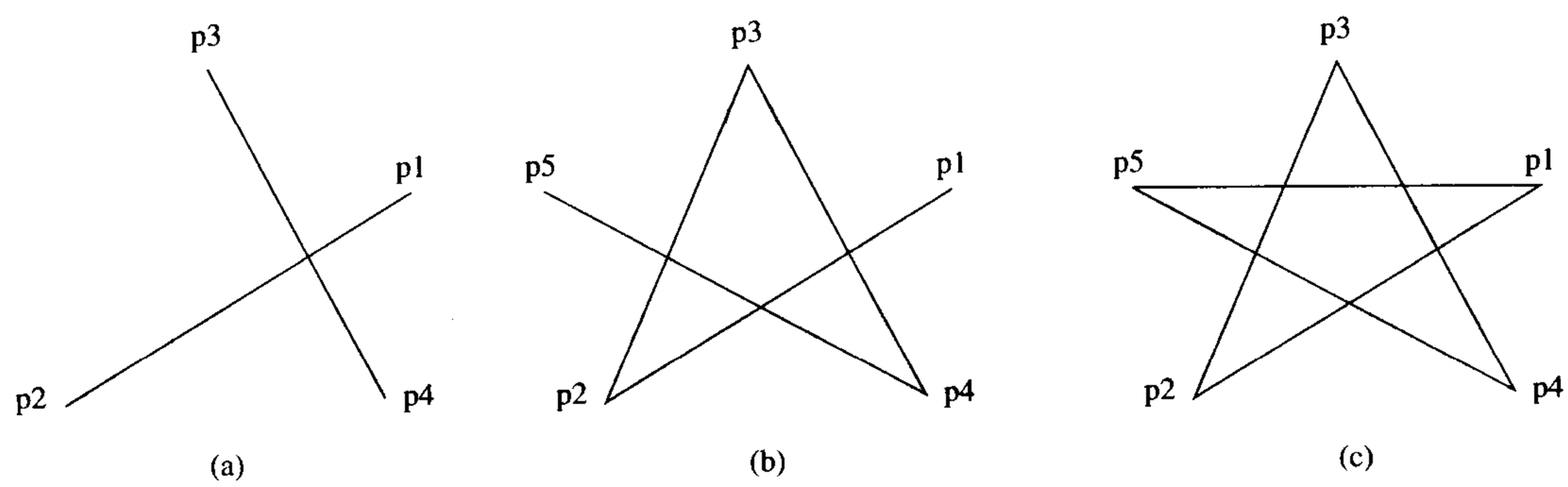


图3.4 OpenGL中使用5个端点坐标可以显示的线段：(a)使用图元常量GL_LINES生成一组未连接的线段；(b)使用GL_LINE_STRIP生成一折线；(c)使用GL_LINE_LOOP生成封闭折线

如前所述，世界坐标系中描述的图形部分最终要映射到输出设备的坐标系统中。然后图中的几何信息被扫描转换到像素位置。下一节，我们将看到实现OpenGL画线函数的扫描转换算法。

3.5 画线算法

场景中的直线段由其两端点的坐标位置来定义。要在光栅监视器上显示一线段，图形系统必须先将两端点投影到整数屏幕坐标，并确定离两端点间的直线路径最近的像素位置。接下来将颜色值装入帧缓存相应的像素坐标处。视频控制器从帧缓存读出写入的颜色值并绘制屏幕像素。这一过程将一线段数字化为一组离散的整数位置。一般而言，这些位置是实际线路径的近似。例如，计算出的线位置(10.48, 20.51)转换为像素位置(10, 21)。坐标值舍入到整数，引起除水平和垂直以外所有线段的阶梯效应（“锯齿形”），如图3.5所示。光栅线段特有的阶梯现象在低分辨率系统中特别容易看出来，而在高分辨率系统中可以得到改善。平滑光栅线段的更有效技术是基于调整线路径上的像素强度（参见4.17节）。

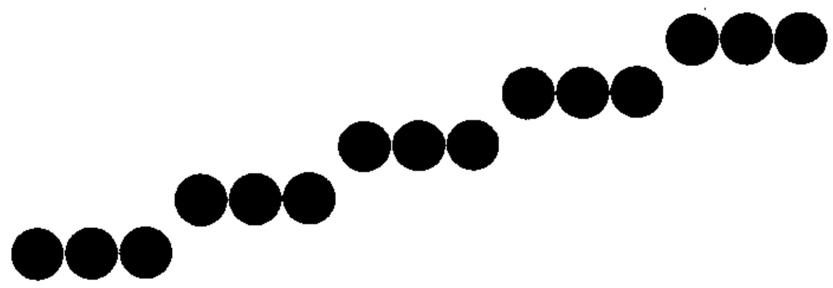


图3.5 当直线作为一系列像素位置生成时所产生的阶梯效果（锯齿形）

3.5.1 直线方程

根据直线的几何特征可确定直线路径的像素位置。直线的笛卡儿斜率截距方程为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$y = m \cdot x + b \tag{3.1}$$

其中，m 表示直线的斜率，b 是 y 轴截距。给定图 3.6 所示线段的两个端点(x_0, y_0)和(x_{end}, y_{end})，可以计算斜率 m 和 y 轴截距 b：

$$m = \frac{y_{end} - y_0}{x_{end} - x_0} \tag{3.2}$$

$$b = y_0 - m \cdot x_0 \tag{3.3}$$

显示直线的算法则以直线方程（3.1）以及等式（3.2）和等式（3.3）给出的计算方法为基础。

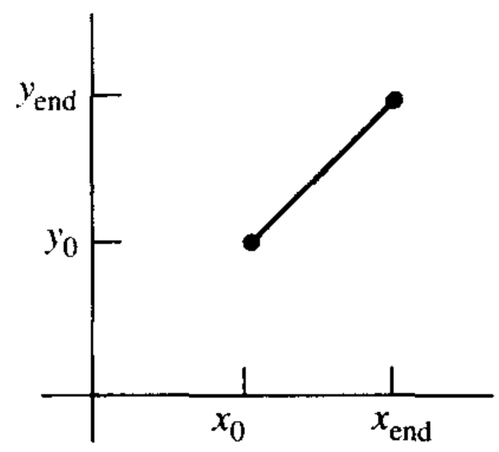


图 3.6　在端点(x_0, y_0)和(x_{end}, y_{end})之间的线路径

对于任何沿直线给定的 x 增量 δx，可以从等式（3.2）中计算出对应的 y 增量 δy：

$$\delta y = m \cdot \delta x \tag{3.4}$$

同样，可以得出对应于指定的 δy 的 x 增量 δx：

$$\delta x = \frac{\delta y}{m} \tag{3.5}$$

这些方程形成了模拟设备（如向量扫描系统）中确定偏转电压的基础，其中有可能造成微小的偏转电压变化。对于具有斜率绝对值 $|m| < 1$ 的直线，可以设置一个较小的水平偏转电压 δx，对应的垂直偏转电压则可以使用等式（3.4）计算出来的 δy 来设定；而对于斜率值 $|m| > 1$ 的直线，则设置一个较小的垂直偏转电压 δy，对应的水平偏转电压则由等式（3.5）计算出来的 δx 来设定；对于斜率 $m = 1$ 的直线，$\delta x = \delta y$，因此水平偏转和垂直偏转电压相等。在每一种情况下，都可以在指定的端点间生成一条斜率为 m 的平滑直线。

在光栅系统中，通过像素绘制直线，水平和垂直方向的步长受到像素的间距的限制。也就是必须在离散位置上对直线取样，并且在每个取样位置上确定距直线最近的像素。图 3.7 给出了直线的扫描转换过程及沿 x 轴具有离散取样点位置。

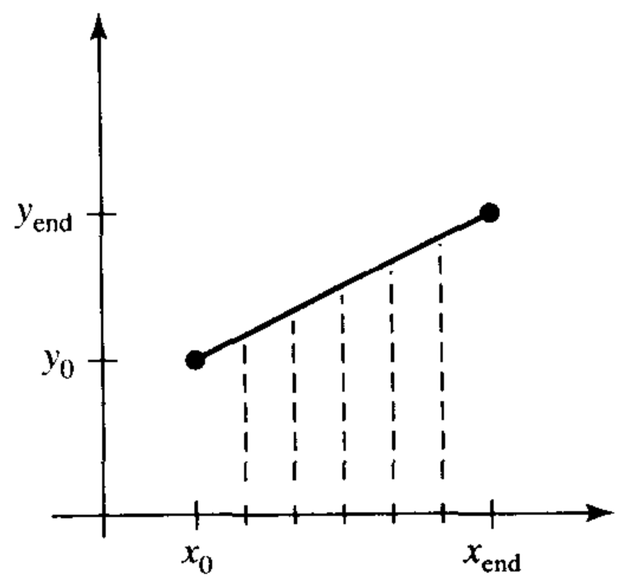


图 3.7　沿 x 轴的 x_0 和 x_{end} 之间具有 5 个取样点的线段

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3.5.2 DDA 算法

数字微分分析仪（digital differential analyzer，DDA）方法是一种线段扫描转换算法，基于使用等式（3.4）或等式（3.5）计算的 δx 或 δy。在一个坐标轴上以单位间隔对线段取样，从而确定另一个坐标轴上最靠近线路径的对应整数值。

首先考虑如图 3.6 所示的具有正斜率的线段。例如，如果斜率小于等于 1，则以单位 x 间隔（$\delta x = 1$）取样，并逐个计算每一个 y 值：

$$y_{k+1} = y_k + m \tag{3.6}$$

下标 k 取整数值，从第一个点 1 开始递增直至最后端点。由于 m 可以是 0 与 1 之间的任意实数，所以计算出的 y 值必须取整。

对于具有大于 1 的正斜率的线段，则交换 x 和 y 的位置。也就是以单位 y 间隔（$\delta y = 1$）取样，并计算每个连续的 x 值：

$$x_{k+1} = x_k + \frac{1}{m} \tag{3.7}$$

此时，每一个计算出的 x 值要沿 y 扫描线舍入到最近的像素位置。

等式（3.6）和等式（3.7）基于从左端点到右端点处理线段的假设（参见图 3.6）。假如这个过程中的处理方向相反，即起始端点在右侧，那么 $\delta x = -1$，并且

$$y_{k+1} = y_k - m \tag{3.8}$$

或者（当斜率大于 1 时）是 $\delta y = -1$，并且

$$x_{k+1} = x_k - \frac{1}{m} \tag{3.9}$$

等式（3.6）和等式（3.9）也可以用来计算具有负斜率的线段的像素位置。假如斜率的绝对值小于 1，并且起始端点在左侧，可设置 $\delta x = 1$ 并用等式（3.6）计算 y 值。当起始端点在右侧（具有相同斜率）时，我们可设置 $\delta x = -1$ 并且由等式（3.8）得到 y 的位置。同样，负斜率的绝对值大于 1 时，可以使用 $\delta y = -1$ 和等式（3.9）或者 $\delta y = 1$ 和等式（3.7）进行计算。

这个算法可以概括为下面的过程：输入线段两个端点的像素位置，端点位置间的水平和垂直差值赋给参数 dx 和 dy。绝对值大的参数确定参数 steps 的值。从像素位置(x0, y0)开始，确定沿线段生成下一个像素位置的每一步所需的偏移量，并循环上述过程 steps 次。假如 dx 的绝对值大于 dy 的绝对值，且 x0 小于 xEnd，那么 x 和 y 方向的增量值分别为 1 和 m。假如 x 方向的变化较大，但 x0 大于 xEnd，那么就采用减量 −1 和 $-m$ 来生成线段上的每个点。在其他情况下，y 方向使用单位增量（或减量），x 方向使用 $1/m$ 的增量（或减量）。

```
#include <stdlib.h>
#include <math.h>

inline int round (const float a)  { return int (a + 0.5); }

void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
   int dx = xEnd - x0,  dy = yEnd - y0,  steps,  k;
   float xIncrement, yIncrement, x = x0, y = y0;

   if (fabs (dx) > fabs (dy))
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    steps = fabs (dx);
else
    steps = fabs (dy);
xIncrement = float (dx) / float (steps);
yIncrement = float (dy) / float (steps);

setPixel (round (x), round (y));
for (k = 0; k < steps; k++) {
    x += xIncrement;
    y += yIncrement;
    setPixel (round (x), round (y));
  }
}
```

DDA方法计算像素位置要比直接使用直线方程（3.1）计算的速度更快。它利用光栅特性消除了直线方程（3.1）中的乘法，而在x或y方向使用合适的增量，从而沿线路径逐步得到各像素的位置。但在浮点增量的连续迭加中，取整误差的积累使得对于较长线段所计算的像素位置偏离实际线段。而且该过程中的取整操作和浮点运算仍然十分耗时。我们可以通过将增量m和$1/m$分离成整数和小数部分，从而使所有的计算都简化为整数操作来改善DDA算法的性能。在4.10节将讨论用整数步长计算$1/m$增量的方法。在下一节中，我们考虑既能用于直线又能用于曲线的更通用的扫描线程序。

3.5.3 Bresenham画线算法

Bresenham画线算法是由Bresenham提出的一种精确而有效的光栅线生成算法，该算法仅仅使用增量整数计算。另外，Bresenham算法还可用于显示圆和其他曲线。图3.8和图3.9给出了绘制线段的屏幕局部。垂直轴表示扫描线位置，水平轴标识像素列。在这个例子中，我们以单位x间隔取样，并且需要确定每次取样时两个可能的像素位置中的哪一个更接近于线路径。从图3.8中的左端点开始，需要确定下一个取样像素位置是(11, 11)还是(11, 12)。类似地，图3.9则给出了像素位置(50, 50)为左端点的具有负斜率的线段。此时，需要确定下一个像素位置是(51, 50)还是(51, 49)。为了解决这些问题，Bresenham算法将对整型参数的符号进行检测，整型参数的值正比于两像素与实际线段之间的偏移。

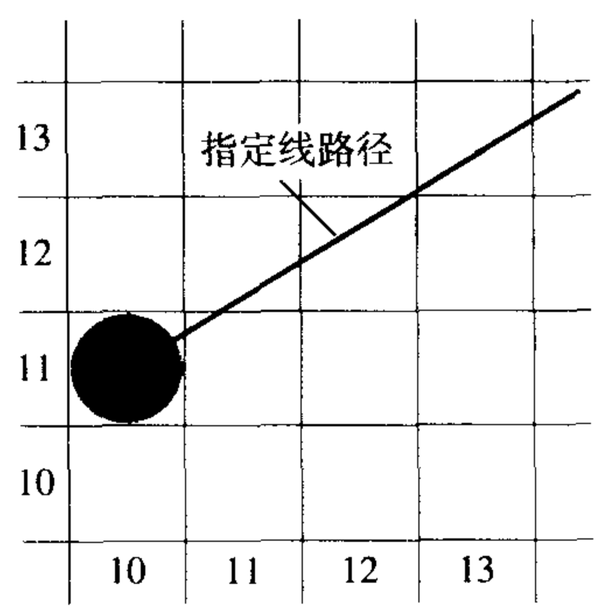


图3.8 从(10, 11)像素开始绘制线段的屏幕局部

为了说明Bresenham方法，首先考虑斜率小于1的直线的扫描转换过程，沿线路径的像素位置由以单位x间隔的取样来确定。从给定线段的左端点(x_0, y_0)开始，逐步处理每个后继列（x位置），并在其扫描线y值最接近线段的像素上绘出一点。图3.10显示了这个过程的第k步。假如已经决定要显示的像素在(x_k, y_k)，那么下一步需要确定在列$x_{k+1}=x_k+1$上绘制哪个像素，是在位置(x_k+1, y_k)，还是位置(x_k+1, y_k+1)。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

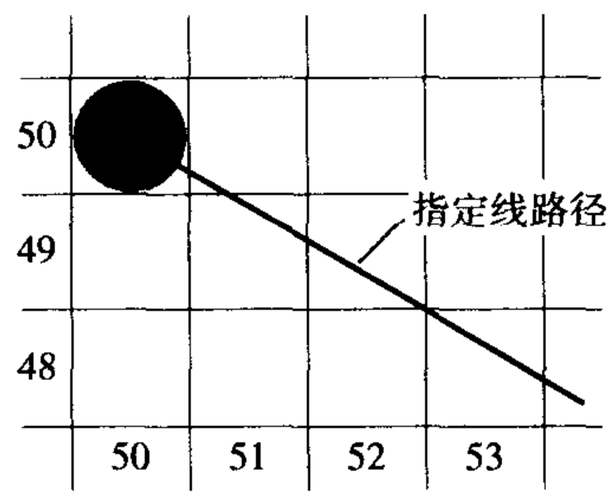


图 3.9 从(50, 50)像素开始的负斜率直线段的屏幕局部

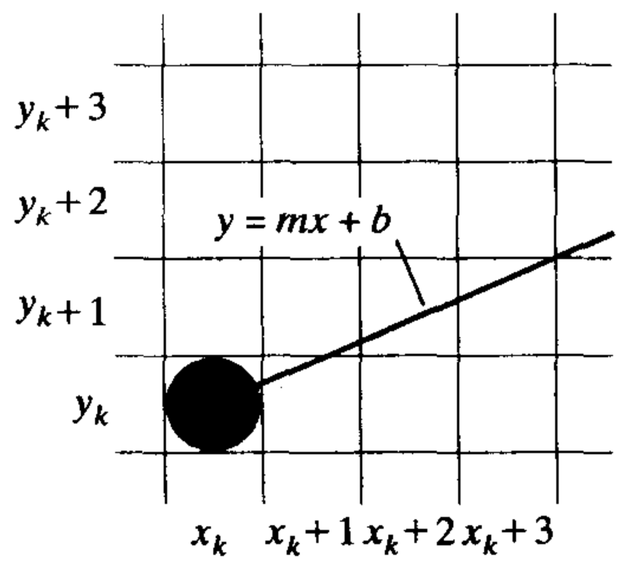


图 3.10 从$(x_k + y_k)$像素开始，绘制斜率$0 < m < 1$的直线的屏幕局部

在取样位置x_k+1，我们使用d_{lower}和d_{upper}来标识两个像素与数学上线路径的垂直偏移（参见图 3.11），在像素列位置x_k+1处的直线上的y坐标可计算为

$$y = m(x_k + 1) + b \tag{3.10}$$

那么

$$\begin{aligned} d_{\text{lower}} &= y - y_k \\ &= m(x_k + 1) + b - y_k \end{aligned} \tag{3.11}$$

和

$$\begin{aligned} d_{\text{upper}} &= (y_k + 1) - y \\ &= y_k + 1 - m(x_k + 1) - b \end{aligned} \tag{3.12}$$

要确定两像素中哪一个更接近线路径，需测试这两个像素偏移的差：

$$d_{\text{lower}} - d_{\text{upper}} = 2m(x_k + 1) - 2y_k + 2b - 1 \tag{3.13}$$

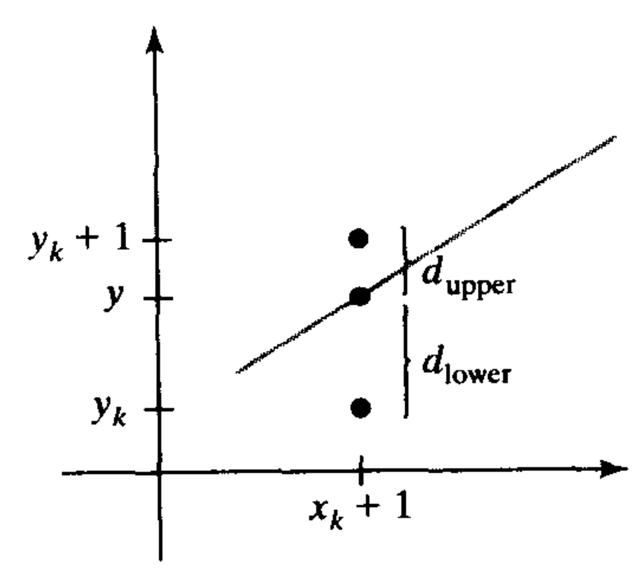


图 3.11 在取样位置x_k+1处，像素位置到直线上y坐标之间的垂直距离

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

通过重新安排等式（3.13）可获得画线算法第 k 步的决策参数 p_k，从而仅使用整数进行计算。设 Δy 和 Δx 分别为两端点的垂直和水平偏移量，令 $m = \Delta y / \Delta x$，将决策参数定义为

$$\begin{aligned} p_k &= \Delta x(d_{\text{lower}} - d_{\text{upper}}) \\ &= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c \end{aligned} \tag{3.14}$$

p_k 的符号与 $d_{\text{lower}} - d_{\text{upper}}$ 的符号相同（因为例子中 $\Delta x > 0$），参数 c 是一常量，其值为 $2\Delta y + \Delta x(2b - 1)$，它与像素位置无关，且会在循环计算 p_k 时被消除。假如 y_k 处的像素比 y_k+1 的像素更接近于线段（即 $d_{\text{lower}} < d_{\text{upper}}$），那么参数 p_k 是负的。此时，绘制下面的像素；反之，绘制上面的像素。

直线上的坐标会沿 x 或 y 方向的单位步长而变化。因此，可以利用递增整数运算得到后继的决策参数值。在 $k+1$ 步，决策参数可以从等式（3.14）计算得出：

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

将上述方程减去等式（3.14），可以得到

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

但是 $x_{k+1} = x_k + 1$，因而得到

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) \tag{3.15}$$

其中，$y_{k+1} - y_k$ 取 0 或 1，取决于参数 p_k 的符号。

决策参数的递归运算在线段的坐标端点开始的每个整数 x 位置进行，起始像素位置 (x_0, y_0) 的第一个参数 p_0 通过等式（3.14）及 $m = \Delta y/\Delta x$ 计算得出：

$$p_0 = 2\Delta y - \Delta x \tag{3.16}$$

我们可以将正斜率小于 1 的线段的 Bresenham 画线算法概括为以下步骤。常量 $2\Delta y$ 和 $2\Delta y - 2\Delta x$ 对每条进行扫描转换的直线只计算一次，因此该系统仅进行这两个常量之间的整数加减法。

$|m|<1$ 时的 Bresenham 画线算法：

1. 输入线段的两个端点，并将左端点存储在 (x_0, y_0) 中；
2. 将 (x_0, y_0) 装入帧缓存，画出第一个点；
3. 计算常量 Δx、Δy、$2\Delta y$ 和 $2\Delta y - 2\Delta x$，并得到决策参数的第一个值：

$$p_0 = 2\Delta y - \Delta x$$

4. 从 $k = 0$ 开始，在沿线路径的每个 x_k 处，进行下列检测：

   如果 $p_k < 0$，下一个要绘制的点是 $(x_k + 1, y_k)$，并且

$$p_{k+1} = p_k + 2\Delta y$$

   否则，下一个要绘制的点是 $(x_k + 1, y_k + 1)$，并且

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. 重复步骤 4，共 $\Delta x - 1$ 次。

例 3.1　Bresenham 画线算法

为了演示上述算法，我们绘制这样一条线段：端点为 (20, 10) 和 (30, 18)。该线段的斜率为 0.8 且

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\Delta x = 10, \qquad \Delta y = 8$$

那么初始决策参数的值为

$$\begin{aligned} p_0 &= 2\Delta y - \Delta x \\ &= 6 \end{aligned}$$

计算后继决策参数的增量为

$$2\Delta y = 16, \qquad 2\Delta y - 2\Delta x = -4$$

绘制初始点$(x_0, y_0) = (20, 10)$，并从决策参数中确定沿线路径的后继像素位置为

k	p_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)
1	2	(22, 12)
2	−2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)

k	p_k	(x_{k+1}, y_{k+1})
5	6	(26, 15)
6	2	(27, 16)
7	−2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

图 3.12 中给出了沿这条线路径生成的像素点。

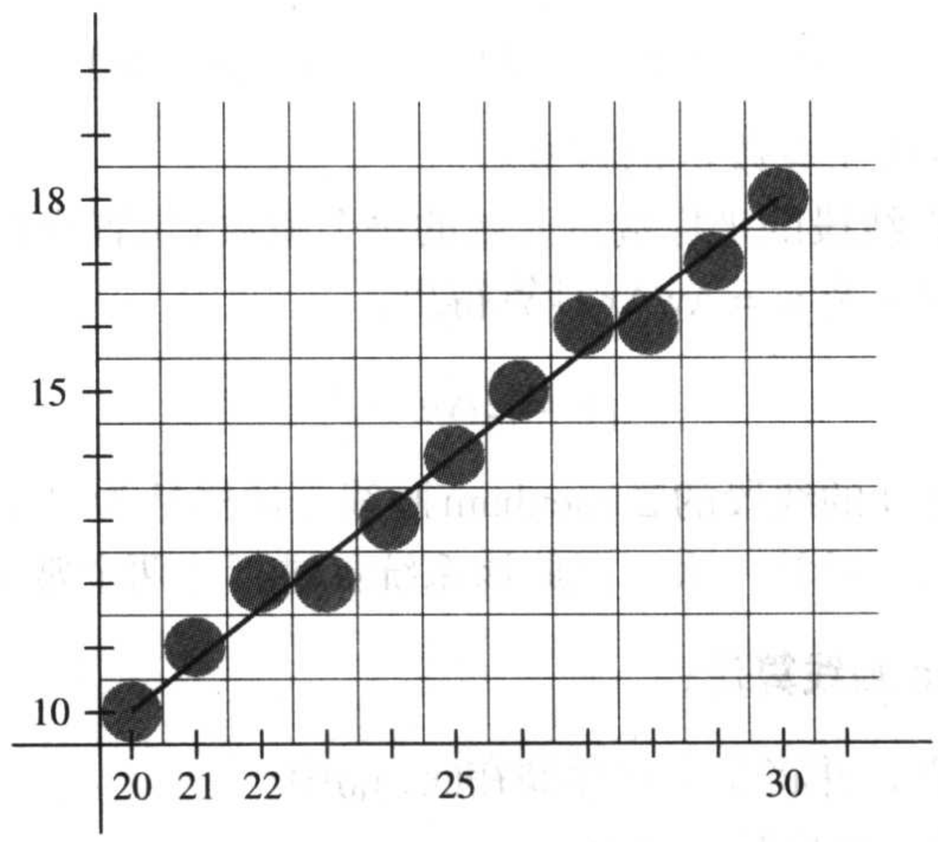


图 3.12　沿端点(20, 10)和(30, 18)间的线路径和用 Bresenham 画线算法绘制的像素位置

下列程序中给出了斜率为 $0 < m < 1.0$ 的 Bresenham 画线算法的实现。首先将线段的端点像素位置输入程序，然后从左端点到右端点绘制像素。

```
#include <stdlib.h>
#include <math.h>

/*  Bresenham line-drawing procedure for |m| < 1.0.  */
void lineBres (int x0, int y0, int xEnd, int yEnd)
{
   int dx = fabs (xEnd - x0),  dy = fabs(yEnd - y0);
   int p = 2 * dy - dx;
   int twoDy = 2 * dy,  twoDyMinusDx = 2 * (dy - dx);
   int x, y;

   /* Determine which endpoint to use as start position.  */
   if (x0 > xEnd) {
      x = xEnd;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        y = yEnd;
        xEnd = x0;
    }
    else {
        x = x0;
        y = y0;
    }
    setPixel (x, y);

    while (x < xEnd) {
        x++;
        if (p < 0)
            p += twoDy;
        else {
            y++;
            p += twoDyMinusDx;
        }
        setPixel (x, y);
    }
}
```

通过考虑xy平面各种八分和四分区域间的对称性，Bresenham算法对任意斜率的线段具有通用性。对于斜率为正值且大于1.0的线段，只要交换x和y方向的规则，即沿y方向以单位步长移动并计算最接近线路径的连续x值。当然，也可以改变程序，使之能从任何端点开始绘制像素，假如正斜率线段的初始位置是右端点，那么在从右至左的步进中，x和y都将递减。为了确定无论从任何端点开始都能绘制相同的像素，当候选像素相对于线段的两个垂直偏移相等时（$d_{lower} = d_{upper}$），我们总是选择其中较高（或较低）的像素。对于绘制负斜率的线段，除非一个坐标递减而另一个递增，否则程序是类似的。最后，可以分别处理下列特殊情况：水平线（$\Delta y = 0$）、垂直线（$\Delta x = 0$）和对角线（$|\Delta x| = |\Delta y|$），它们都可直接装入帧缓存而无需进行画线算法处理。

3.5.4 显示折线

折线函数通过$n-1$次调用画线函数并显示连接n个端点的$n-1$条线段来实现。每次调用传递所需绘出的下一条线段的两个端点坐标，其中第一端点是上一条线段的后一端点。一旦将第一条线段像素位置的颜色写进帧缓存，下一条线段的处理从其第一端点之后的像素位置开始。这样，我们可以避免对某些端点赋两次颜色。在3.13节，我们将详细讨论避免重复显示对象的方法。

3.6 并行画线算法

上面讨论过的线段生成算法顺序地确定像素位置。而利用并行计算机，则可通过将计算分割到可用的多个处理器中来得到线段的像素位置。分割问题的一种解决方法是将现有的顺序算法放到多个处理器上。我们也可以寻找其他处理办法，从而使像素位置能以并行方式有效地计算。在设计并行算法中，重要的是要考虑平衡可用处理器间的处理负载。

给定n_p个处理器，我们可以通过把线段分割成n_p个子段，并在每个子段中同时生成线段而建立起并行的Bresenham画线算法。对于斜率为$0 < m < 1.0$且左端点坐标位置为(x_0, y_0)的线段，我们沿正x方向对线段进行分割。相邻分段的起始x位置间的距离可计算为

$$\Delta x_p = \frac{\Delta x + n_p - 1}{n_p} \tag{3.17}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

其中，Δx 是线段的水平宽度，分段水平宽度 Δx_p 的值利用整数除法来计算。将分段和处理器从 0、1、2 直到 n_p-1 编号，可以计算出第 k 分段的起始的 x 坐标为

$$x_k = x_0 + k\Delta x_p \tag{3.18}$$

例如，假设 $\Delta x = 15$，并且具有 $n_p = 4$ 个处理器，那么分段的水平宽度是 4，各分段的初始 x 值为 x_0、x_0+4、x_0+8、x_0+12。对于这种分段策略，有些情况下最后（最右边）的子段会比其他段小。此外，假如线段的端点不是整数，舍入误差将导致沿线的长度产生宽度不同的分段。

为了将 Bresenham 算法用于各分段，需要有每个分段的 y 坐标初始值和决策参数的初始值。分段 y 方向的变化 Δy_p，可从线段斜率 m 和分段宽度 Δx_p 计算得出：

$$\Delta y_p = m\Delta x_p \tag{3.19}$$

那么，第 k 分段的起始 y 坐标为

$$y_k = y_0 + \mathrm{round}(k\Delta y_p) \tag{3.20}$$

第 k 分段起始处 Bresenham 算法的初始决策参数可从等式（3.14）中得到

$$p_k = (k\Delta x_p)(2\Delta y) - \mathrm{round}(k\Delta y_p)(2\Delta x) + 2\Delta y - \Delta x \tag{3.21}$$

然后，各处理器利用该分段的初始决策参数值和起始坐标(x_k, y_k)，计算指定的分段上的像素位置。我们也可以通过替换 $m = \Delta y/\Delta x$ 和重新安排有关项，将 y_k 和 p_k 起始值计算中的浮点运算简化为整数运算。在 y 方向对线段进行分段并计算分段的起始 x 值，可以将并行 Bresenham 算法拓展到斜率大于 1.0 的线段。对于负斜率，则可以在一个方向递增坐标值，而在另一方向上递减。

建立光栅系统并行算法的另一种方法是，为每个处理器分配一组屏幕像素。只要具有足够数量的处理器，就可以将每个处理器分配给某个屏幕区域内的一个像素。这种方法可以通过为一个处理器分配线段坐标范围之内的一个像素并计算像素距线段的距离而移植成线段显示的方法。在线段的包围盒中的像素数目为 $\Delta x \cdot \Delta y$（参见图 3.13）。在图 3.13 中，从线段到坐标(x, y)处像素的垂直距离 d，可以利用下列算式得到：

$$d = Ax + By + C \tag{3.22}$$

其中

$$A = \frac{-\Delta y}{\text{线长}}$$

$$B = \frac{\Delta x}{\text{线长}}$$

$$C = \frac{x_0\Delta y - y_0\Delta x}{\text{线长}}$$

并且

$$\text{线长} = \sqrt{\Delta x^2 + \Delta y^2}$$

一旦估算出线段的常量 A、B 和 C，那么每个处理器只需完成两次乘法和两次加法来计算像素距离 d。如果 d 小于指定的线段粗细参数，那么就绘制一个像素。

除了把屏幕分割成单个像素，我们也可以按线段的斜率为每个处理器分配一条扫描线或一列像素。然后，每个处理器计算线段与分配给该处理器的水平行或垂直列的交点。对于斜率 $|m| < 1.0$ 的

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

线段，每个处理器将简单地按给定的x值从直线方程中求解y。对于斜率值大于1.0的线段，处理器则根据给定的扫描线y值，从直线方程中求解x。尽管这种直接方法在顺序算法机器中的计算速度很慢，但通过使用多处理器能十分有效地完成这一算法。

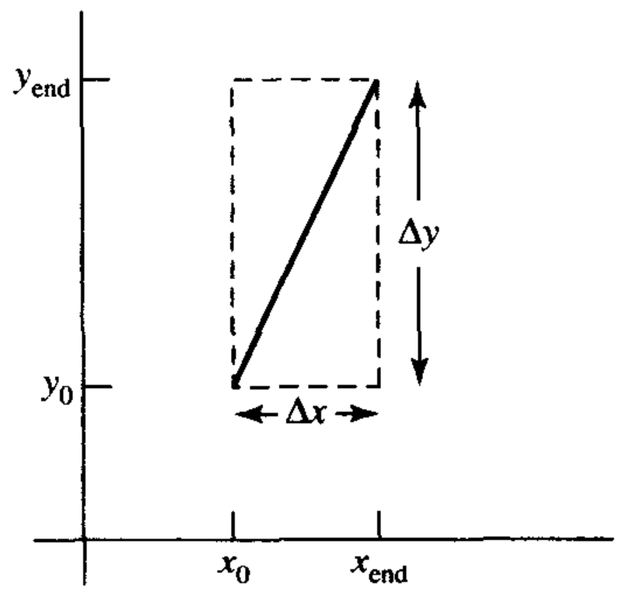


图 3.13　坐标范围为 Δx 和 Δy 的线段边界框

3.7　帧缓存值的装载

实现线段和其他对象显示函数的最后一步工作是设定帧缓存的颜色值。由于扫描转换算法以连续的单位间隔生成像素位置，因此扫描转换算法可使用增量方法在每一步高效地存取帧缓存。

作为一个特殊的例子，假设帧缓存矩阵是以行为主要顺序进行编址，并且像素位置从屏幕左上方(0, 0)变化到屏幕右上方(x_{max}, y_{max})（参见图 3.14）。对于二级系统（每个像素 1 位），像素位置(x, y)的帧缓存位地址可以这样计算：

$$\text{addr}(x, y) = \text{addr}(0, 0) + y(x_{max} + 1) + x \tag{3.23}$$

沿扫描线移动，像素$(x+1, y)$处的帧缓存地址可以根据位置(x, y)的地址偏移进行计算：

$$\text{addr}(x + 1, y) = \text{addr}(x, y) + 1 \tag{3.24}$$

从(x, y)按对角线方向转移到下一条扫描线，那么$(x+1, y+1)$的帧缓存地址的算式为

$$\text{addr}(x + 1, y + 1) = \text{addr}(x, y) + x_{max} + 2 \tag{3.25}$$

其中，常数$x_{max}+2$对于所有线段只需计算一次。同样，从等式（3.23）中可以得到屏幕x和y负方向单位步长的增量计算。这种地址计算仅包含一个整数加法。

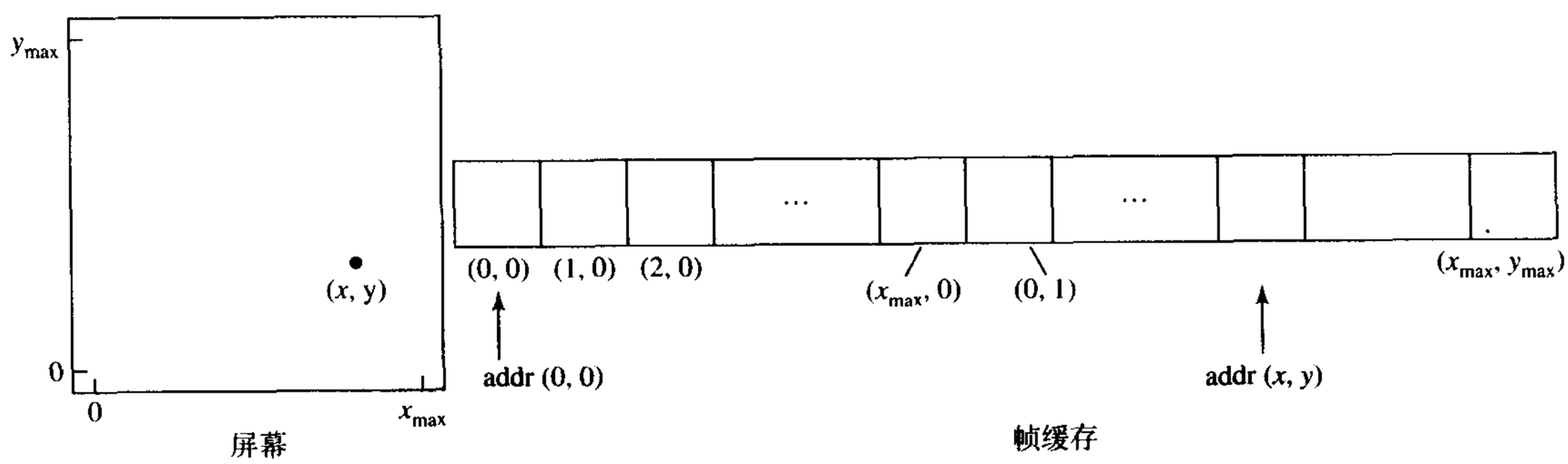


图 3.14　帧缓存内以行为主序而线性存储的像素屏幕位置

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

实现存储像素亮度值的setPixel程序的方法，取决于特定系统的能力以及软件包的设计需求。对于能为每个像素显示一定范围亮度值的系统，帧缓存地址的计算包括像素宽度（位数）以及像素屏幕位置。

3.8 OpenGL 曲线函数

生成圆和椭圆等基本曲线的函数并未作为图元功能包含在OpenGL核心库中。但该库包含了显示Bezier样条的功能，该曲线是由一组离散点定义的多项式。OpenGL实用库（GLU）中包含有球面和柱面等三维曲面函数以及生成B样条的函数，它是包含简化Bézier曲线的样条曲线的总集。我们可以使用有理B样条显示圆、椭圆和其他二维曲线。此外，OpenGL实用工具包（GLUT）中还有可以用来显示某些三维曲面（如球面、锥面和其他形体）的函数。然而，所有这些函数比本章中介绍的基本图元应用得更多，因此我们将在第8章进一步讨论这一组函数。

我们还可以使用折线来近似地显示简单曲线。仅需确定一组曲线上的点并将它们连接成一组直线段。折线中的线段越多，曲线越平滑。图3.15给出了用于表示弧的几种折线显示。

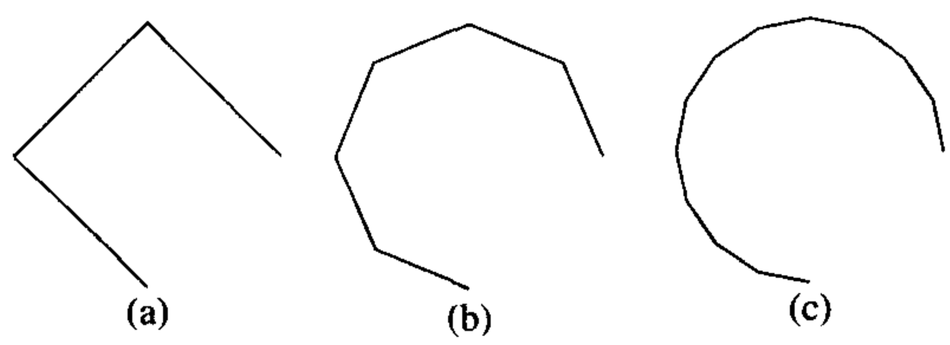


图3.15 近似表示一段弧，使用(a)三条线段、(b)六条线段、(c)十二条线段

第三种可选方法是按下面将给出的算法写出自己的曲线生成函数。首先讨论生成圆和椭圆的高速算法，然后再看一个生成其他二次曲线、多项式和样条曲线的函数。

3.9 圆生成算法

由于圆是图形中经常使用的元素，因此在大多数图形软件中都包含生成圆和圆弧的函数。这些软件有时也会提供一个能显示包括圆和椭圆在内的多种曲线的通用函数。

3.9.1 圆的特性

我们将圆定义为所有距中心位置(x_c, y_c)为给定值r的点集（参见图3.16）。对于任意的圆点(x, y)，这个距离关系可用笛卡儿坐标系中的勾股（Pythagorean）定理定义为

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \tag{3.26}$$

利用这个方程，我们可以沿x轴从x_c-r到x_c+r以单位步长计算对应的y值，从而得到圆周上每点的位置：

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2} \tag{3.27}$$

但这并非是生成圆的最好方法。这个方法的一个问题是每一步包含很大的计算量。而且，如图3.17所示，所画像素位置间的间距是不一致的。我们可以在圆斜率的绝对值大于1后，交换x和y（即步进y值并计算x值）来调整间距。但是，这种方法增加了算法所需的计算量和处理过程。

另一种消除图3.17中不等间距的方法是使用极坐标r和θ（参见图3.16）来计算沿圆周的点。以参数极坐标形式表示圆方程，可以得到方程组：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$x = x_c + r\cos\theta$$
$$y = y_c + r\sin\theta \tag{3.28}$$

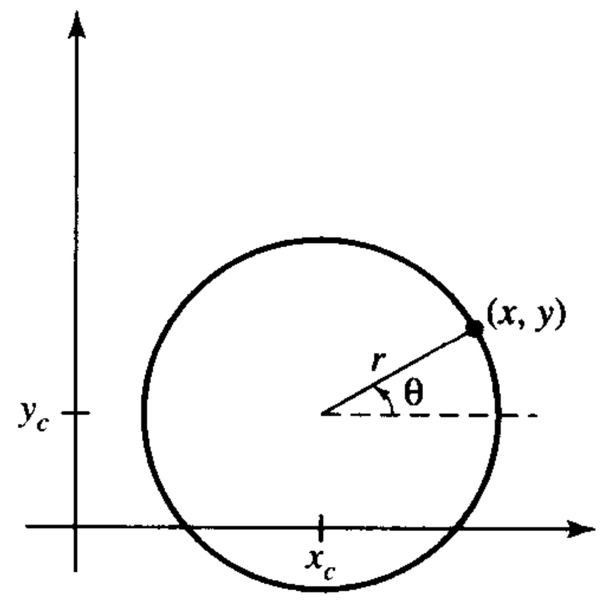


图 3.16　圆心为(x_c, y_c)、半径为 r 的圆

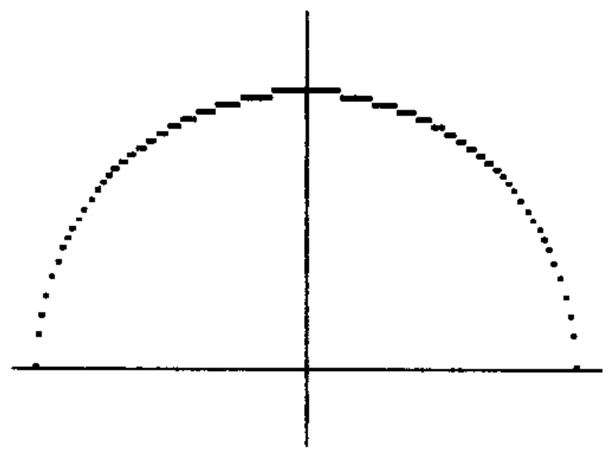


图 3.17　圆心$(x_c, y_c) = (0, 0)$，用方程（3.27）绘制的正半圆

使用上述方法以固定角度为步长生成显示结果时，就可以利用沿圆周的等距点来绘制出圆。为了减少计算量，我们可以在相邻点间使用较大的角度间隔并用线段连接相邻点来逼近圆的路径。在光栅显示中设定角度间隔为 $1/r$ 可获得较连续的边界。这样绘出的像素位置大约间隔一个单位。尽管极坐标系统提供了等距点，但三角函数计算是十分耗时的。

对于上述任何一种圆生成算法，考虑圆的对称性可以减少计算量。圆的形状在每个象限中是相似的。因此，如果我们确定了在第一象限中圆的位置，则可以生成该圆在 xy 平面中第二象限的部分，这是因为两个圆弧段对于 y 轴是对称的。考虑对于 x 轴的对称性，根据第一和第二象限中的圆弧可以得到第三和第四象限的圆弧。在八分圆之间也有对称性，因此可以进一步细化，一个象限内的相邻八分圆的圆弧对于分割两个部分的 45° 直线是对称的。这种对称情况可参见图 3.18，图中八分之一圆上的一点(x, y)将映射到 xy 平面的其他七个八分圆的点。这种方法利用了圆的对称性，仅需计算从 $x = 0$ 到 $x = y$ 分段内的点就可得到整个圆的所有像素位置。在这个八分圆中，圆弧斜率绝对值小于或等于 1.0。$x = 0$ 时圆的斜率是 0，而当 $x = y$ 时圆的斜率是 −1.0。

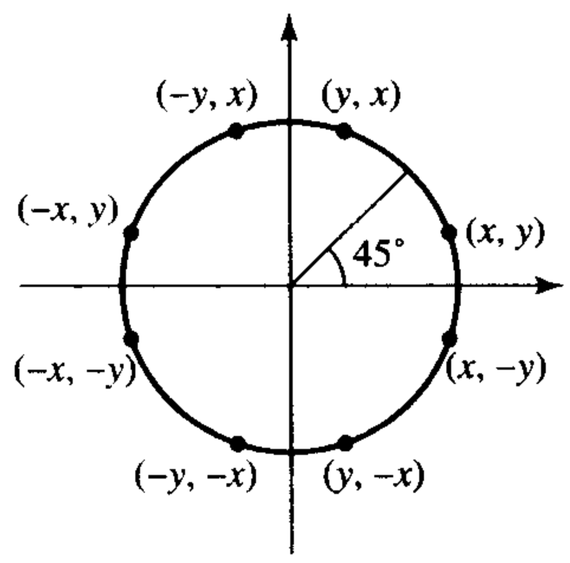


图 3.18　圆的对称性；计算一个八分圆上的点(x, y)，可以映射为其余七个八分圆的点

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

使用对称性及方程（3.26）或方程（3.28）来确定圆周上的像素位置，仍然需要大量的计算时间。笛卡儿方程（3.26）包括乘法和平方根运算，而参数方程中包含乘法和三角运算。更有效的画圆算法是如同 Bresenham 画线算法一样以决策参数的增量计算为基础，这样仅仅包括简单的整数处理。

通过设定在每一取样步骤中寻找最接近圆周像素的决策参数，可以将光栅系统的Bresenham画线算法移植为画圆算法。然而，圆方程（3.26）是非线性的，计算像素与圆的距离必须进行平方根运算。Bresenham 画圆算法则通过比较像素与圆的距离的平方而避免了平方根运算。

然而，不做平方运算而直接比较距离是可能的。该方法的基本思想是检验两像素间的中间位置以确定该中点是在圆边界之内还是之外。这种方法更易应用于其他圆锥曲线，并且对于整数圆半径，中点方法生成与Bresenham画圆算法相同的像素位置。而且使用中点检验时，沿任何圆锥截面曲线所确定的像素位置，其误差限制在像素间隔的 1/2 以内。

3.9.2 中点画圆算法

如同光栅画线算法，我们在每个步中以单位间隔取样并确定离指定圆最近的像素位置。对于给定半径 r 和屏幕中心(x_c, y_c)，可以先使用算法计算圆心在坐标原点(0, 0)的圆的像素位置，然后通过将 x_c 加到 x 且 y_c 加到 y，从而把计算出的每个位置(x, y)移动到其适当的屏幕位置。在第一象限中，圆弧段从 $x=0$ 到 $x=y$，曲线的斜率从 0 变化到 -1.0。因此，可以在该八分圆上的正 x 方向取单位步长，并使用决策参数来确定每一步两个可能的 y 位置中，哪一个更接近于圆的位置。然后，其他七个八分圆中的位置可由对称性得到。

为了应用中点画圆算法，我们定义一个圆函数：

$$f_{\text{circ}}(x, y) = x^2 + y^2 - r^2 \tag{3.29}$$

半径为 r 的圆边界上任何一点(x, y)满足方程$f_{\text{circ}}(x, y) = 0$。如果点在圆的内部，则圆函数为负值；如果点在圆的外部，则圆函数为正值。总之，任何点(x, y)的相对位置可由圆函数的符号来决定：

$$f_{\text{circ}}(x, y) \begin{cases} < 0, & \text{如果}(x, y)\text{位于圆边界内} \\ = 0, & \text{如果}(x, y)\text{位于圆边界上} \\ > 0, & \text{如果}(x, y)\text{位于圆边界外} \end{cases} \tag{3.30}$$

（3.30）式中的圆函数在每个取样步上对接近圆周的两个像素的中点进行测试。因此，在中点算法中圆函数是决策参数，并且可以类似画线算法为这个函数设置增量运算。

图3.19给出了取样位置x_k+1上两个候选像素间的中点。假设刚在(x_k, y_k)绘制了一个像素，下一步需要确定是像素位置(x_k+1, y_k)还是(x_k+1, y_k-1)更接近于圆。我们的决策参数是圆函数方程（3.29）在这两个像素的中点求值而得到的：

$$\begin{aligned} p_k &= f_{\text{circ}}\left(x_k + 1, y_k - \frac{1}{2}\right) \\ &= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \end{aligned} \tag{3.31}$$

假如 $p_k<0$，那么这个中点位于圆内，扫描线 y_k 上的像素接近于圆周边界。否则，中点位于圆外或在圆周边界上，我们选择扫描线 y_k-1 的像素。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

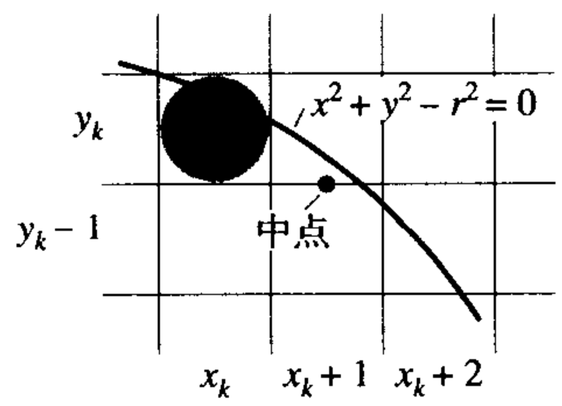


图 3.19 沿圆路径取样位置 x_k+1 上候选像素间的中点

后续的决策参数可以使用增量运算得到。我们对取样位置 $x_{k+1}+1=x_k+2$ 处的圆函数求值，可以得到下一个决策参数的循环表达式：

$$p_{k+1}=f_{\text{circ}}\left(x_{k+1}+1,y_{k+1}-\frac{1}{2}\right)$$

$$=[(x_k+1)+1]^2+\left(y_{k+1}-\frac{1}{2}\right)^2-r^2$$

或

$$p_{k+1}=p_k+2(x_k+1)+\left(y_{k+1}^2-y_k^2\right)-(y_{k+1}-y_k)+1 \tag{3.32}$$

其中，y_{k+1} 是 y_k 或是 y_k-1，取决于 p_k 的符号。

为了得到 p_{k+1}，增量可能是 $2x_{k+1}+1$（如果 p_k 为负）或是 $2x_{k+1}+1-2y_{k+1}$。$2x_{k+1}$ 和 $2y_{k+1}$ 的求值也可以通过增量的方式进行，即

$$2x_{k+1}=2x_k+2$$
$$2y_{k+1}=2y_k-2$$

在起始位置(0, r)处，这两个项的值分别为0和2r。$2x_{k+1}$ 项的每个后续值可以通过对前一值加2或是对 $2y_{k+1}$ 的前一值减2而得到。

对圆函数在起始位置$(x_0, y_0)=(0, r)$处求值，就可以得到初始决策参数：

$$p_0=f_{\text{circ}}\left(1,r-\frac{1}{2}\right)$$

$$=1+\left(r-\frac{1}{2}\right)^2-r^2$$

或

$$p_0=\frac{5}{4}-r \tag{3.33}$$

假如将半径 r 指定为整数，就可以对 p_0 进行简单的取整：

$$p_0=1-r\ （r\text{是整数}）$$

因为所有的增量都是整数。

假设以整数屏幕坐标指定圆参数，那么如同Bresenham画线算法中一样，中点方法使用整数加减来计算沿圆周的像素位置。我们可将中点画圆算法的步骤概括如下：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

中点画圆算法的步骤

1. 输入圆半径 r 和圆心(x_c, y_c)，并得到圆周（圆心在原点）上的第一个点：

$$(x_0, y_0) = (0, r)$$

2. 计算决策参数的初始值：

$$p_0 = \frac{5}{4} - r$$

3. 在每个 x_k 位置，从 $k=0$ 开始，完成下列测试：假如 $p_k < 0$，圆心在(0, 0)的圆的下一个点为(x_{k+1}, y_k)，并且

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

否则，圆的下一个点是(x_k+1, y_k-1)，并且

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

其中 $2x_{k+1} = 2x_k + 2$ 且 $2y_{k+1} = 2y_k - 2$。

4. 确定在其他七个八分圆中的对称点。
5. 将每个计算出的像素位置(x, y)移动到圆心在(x_c, y_c)的圆路径上，并画坐标值：

$$x = x + x_c, \qquad y = y + y_c$$

6. 重复步骤 3 到步骤 5，直至 $x \geq y$。

例 3.2　使用中点画圆算法画图

给定圆半径 $r = 10$，我们将演示中点画圆算法，确定在第一象限从 $x=0$ 到 $x=y$ 沿八分圆的像素位置。决策参数的初始值为

$$p_0 = 1 - r = -9$$

对于中心在坐标原点的圆，初始点$(x_0, y_0) = (0, 10)$，计算决策参数的初始增量项：

$$2x_0 = 0, \qquad 2y_0 = 20$$

使用中点画圆算法计算的后继决策参数值和沿圆路径的位置为

k	p_k	(x_{k+1}, y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0	−9	(1, 10)	2	20
1	−6	(2, 10)	4	20
2	−1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	−3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

第一象限中生成的像素位置请参见图 3.20。

下列程序段给出了用于实现中点画圆算法的过程。半径值和圆心坐标传递给过程 circleMidpoint。然后计算第一个八分圆上的一个像素位置并传递给过程 circlePlotPoints。该过程通过反复调用 setPixel 函数在帧缓存中该像素及与其对称的所有像素位置来设定圆的颜色，setPixel 由 OpenGL 画点函数实现。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

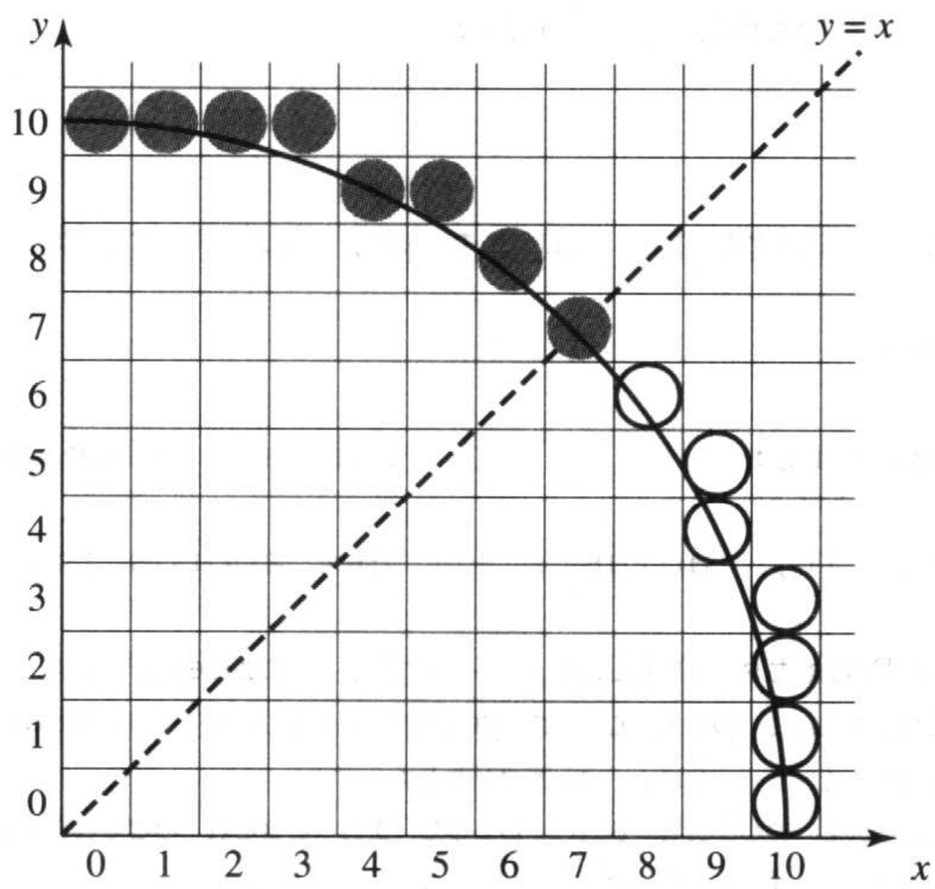


图 3.20 沿半径 $r = 10$、圆心在原点的圆路径，使用中点画圆算法选择的像素位置（实心圆）。空心小圆点显示在第一象限的对称位置

```
#include <GL/glut.h>

class screenPt
{
    private:
       GLint x, y;

    public:
       /*  Default Constructor: initializes coordinate position to (0, 0).  */
       screenPt ( )  {
           x = y = 0;
       }
       void setCoords (GLint xCoordValue, GLint yCoordValue)  {
           x = xCoordValue;
           y = yCoordValue;
       }

       GLint getx ( ) const  {
           return x;
       }

       GLint gety ( ) const  {
           return y;
       }
       void incrementx ( )  {
           x++;
       }
       void decrementy ( )  {
           y--;
       }
};

void setPixel (GLint xCoord, GLint yCoord)
{
    glBegin (GL_POINTS);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        glVertex2i (xCoord, yCoord);
    glEnd ( );
}

void circleMidpoint (GLint xc, GLint yc, GLint radius)
{
    screenPt circPt;

    GLint p = 1 - radius;          //  Initial value for midpoint parameter.

    circPt.setCoords (0, radius); //  Set coords for top point of circle.

    void circlePlotPoints (GLint, GLint, screenPt);
    /*  Plot the initial point in each circle quadrant.  */
    circlePlotPoints (xc, yc, circPt);
    /*  Calculate next point and plot in each octant.  */
    while (circPt.getx ( ) < circPt.gety ( )) {
        circPt.incrementx ( );
        if (p < 0)
            p += 2 * circPt.getx ( ) + 1;
        else {
            circPt.decrementy ( );
            p += 2 * (circPt.getx ( ) - circPt.gety ( )) + 1;
        }
        circlePlotPoints (xc, yc, circPt);
    }
}

void circlePlotPoints (GLint xc, GLint yc, screenPt circPt)
{
    setPixel (xc + circPt.getx ( ), yc + circPt.gety ( ));
    setPixel (xc - circPt.getx ( ), yc + circPt.gety ( ));
    setPixel (xc + circPt.getx ( ), yc - circPt.gety ( ));
    setPixel (xc - circPt.getx ( ), yc - circPt.gety ( ));
    setPixel (xc + circPt.gety ( ), yc + circPt.getx ( ));
    setPixel (xc - circPt.gety ( ), yc + circPt.getx ( ));
    setPixel (xc + circPt.gety ( ), yc - circPt.getx ( ));
    setPixel (xc - circPt.gety ( ), yc - circPt.getx ( ));
}
```

3.10 椭圆生成算法

非严格地说，椭圆是拉长了的圆。还可以说椭圆是经过修改的圆，它的半径从一个方向的最大值变到其正交方向的最小值。椭圆内部这两个正交方向的直线段称为椭圆的长轴和短轴。

3.10.1 椭圆的特征

通过椭圆上任一点到称为椭圆焦点的两个定点的距离可给出椭圆的精确定义：椭圆上任一点到这两点的距离之和都等于一个常数（参见图3.21）。如果椭圆上的任一点 $\mathbf{P}=(x, y)$ 到两个焦点的距离为 d_1 和 d_2，那么椭圆的通用方程可以表示为

$$d_1 + d_2 = \text{常数} \tag{3.34}$$

用焦点坐标 $\mathbf{F}_1=(x_1, y_1)$ 和 $\mathbf{F}_2=(x_2, y_2)$ 来表示距离 d_1 和 d_2，可以得到

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\sqrt{(x-x_1)^2+(y-y_1)^2}+\sqrt{(x-x_2)^2+(y-y_2)^2}=\text{常数} \tag{3.35}$$

对方程求平方，去除剩余的根，再平方，可以按照下列形式重写通用椭圆方程：

$$Ax^2+By^2+Cxy+Dx+Ey+F=0 \tag{3.36}$$

其中，系数 A、B、C、D、E、F 依据焦点坐标和长轴及短轴的尺寸而求得。长轴是通过焦点从椭圆的一侧到另一侧的线段。短轴是横跨椭圆的较短尺寸，它在两个焦点之间的中点位置（椭圆中心）将长轴二等分。

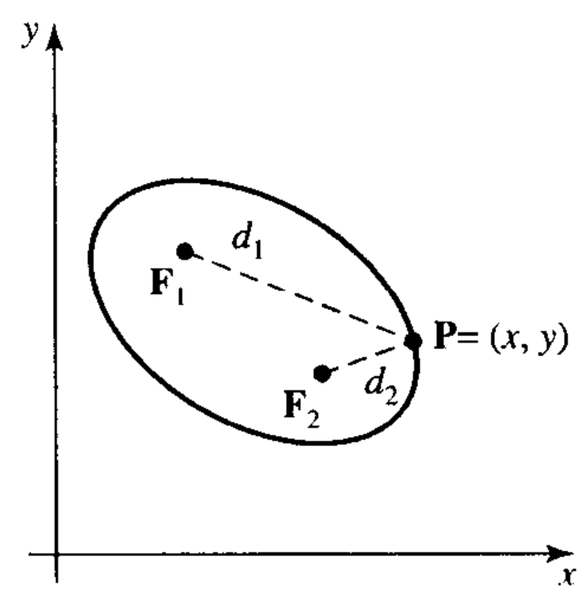


图 3.21　以 $\mathbf{F}_1$、$\mathbf{F}_2$ 为焦点的椭圆

在任意方向指定一个椭圆的交互方法是输入两个焦点和椭圆边界上一个点。利用这三个坐标位置，就可以求出方程（3.35）中的常数。然后就可以求出方程（3.36）中的系数，并利用其生成沿椭圆路径的像素。

如果短轴和长轴与坐标轴方向平行，那么椭圆方程就可以大大简化。图 3.22 给出了一个“标准位置”的椭圆——长轴和短轴平行于 x 和 y 轴。此例中参数 r_x 标识长轴，参数 r_y 标识短轴。对于图 3.22 中的椭圆方程，可以借助于椭圆中心坐标和参数 r_x 和 r_y 而改写为

$$\left(\frac{x-x_c}{r_x}\right)^2+\left(\frac{y-y_c}{r_y}\right)^2=1 \tag{3.37}$$

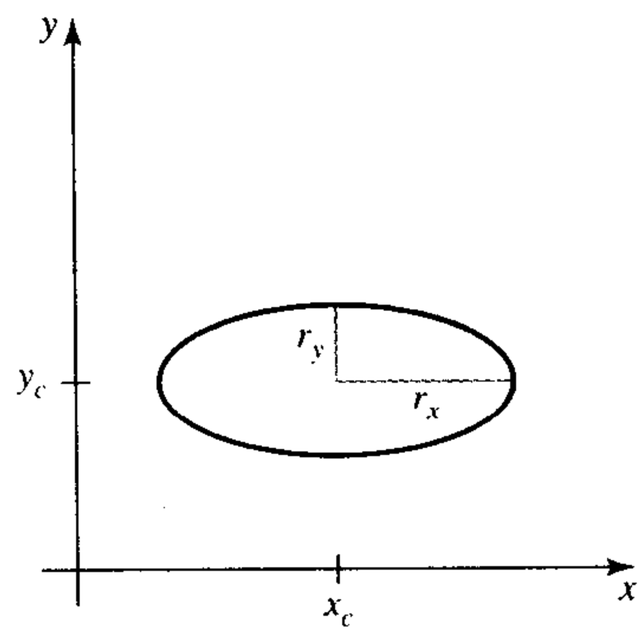


图 3.22　中心为(x_c, y_c)、长轴为 r_x、短轴为 r_y 的椭圆

利用极坐标 r 和 θ，也可以按照参数方程的形式来描述标准位置的椭圆：

$$\begin{aligned} x &= x_c + r_x\cos\theta \\ y &= y_c + r_y\sin\theta \end{aligned} \tag{3.38}$$

称为椭圆离心角（eccentric angle）的 θ 沿其包围圆的圆周进行计量。如果 $r_x > r_y$，包围圆的半径是 $r=r_x$（参见图 3.23），否则，包围圆的半径是 $r=r_y$。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

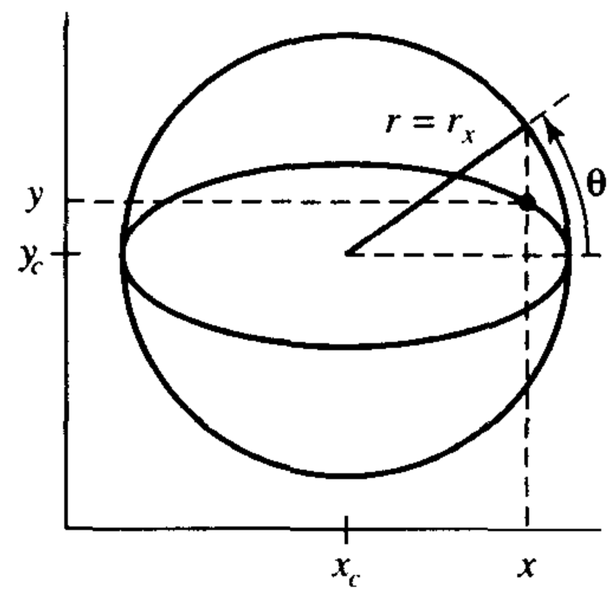


图 3.23　$r_x > r_y$ 时椭圆的包围圆和离心角 θ

和圆的算法一样，考虑椭圆的对称性可以进一步减少计算量。标准位置的椭圆在四分象限中是对称的，但与圆不同，它在八分象限中不是对称的。因此，我们必须计算一个象限中椭圆曲线的像素位置，再由对称性得到其他三个象限中的像素位置（参见图 3.24）。

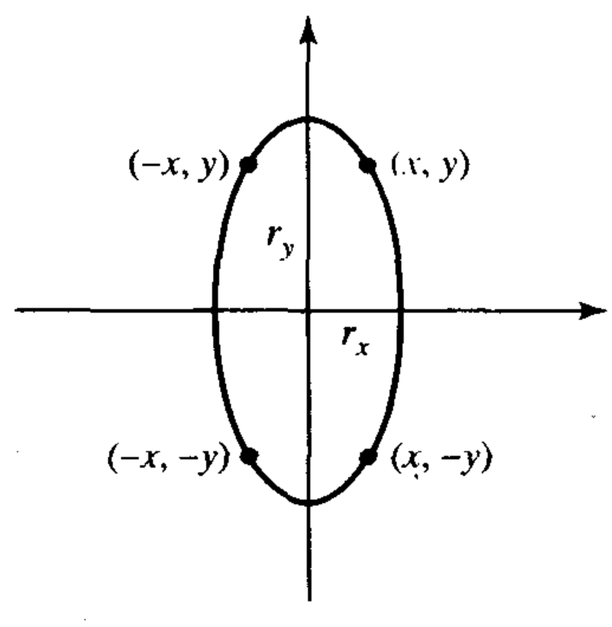


图 3.24　椭圆的对称性——计算在一个四分象限中的点(x, y)，可以得到其他三个象限中的椭圆点

3.10.2　中点椭圆算法

这里的方法类似于显示光栅圆。给定参数 r_x、r_y 和(x_c, y_c)，首先确定以原点为中心的标准位置椭圆上的点(x, y)，然后将这些点平移到以 (x_c, y_c) 为中心的椭圆上。如果希望显示不在标准位置的椭圆，那么就绕中心坐标旋转并对长轴和短轴重新定向。但目前仅考虑显示标准位置的椭圆，第 5 章将讨论变换对象方向和位置的通用方法。

中点椭圆算法将分成两部分应用于第一象限。图3.25给出了依据 $r_x < r_y$ 的椭圆斜率对第一象限进行的划分。并通过在斜率绝对值小于 1 的区域内在 x 方向取单位步长，以及在斜率绝对值大于 1 的区域内在 y 方向取单位步长来处理这个象限。

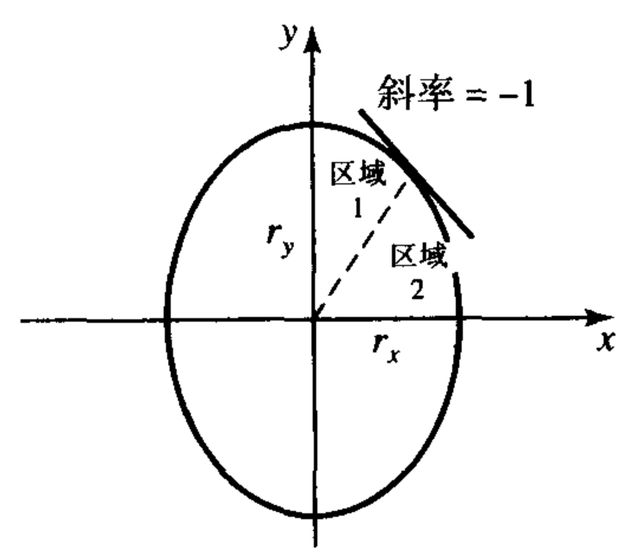


图 3.25　椭圆的处理区域：在区域 1，椭圆斜率小于 −1；在区域 2，椭圆斜率大于 −1

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

区域 1 和区域 2（参见图 3.25）可以通过多种方法进行处理。可以从位置$(0, r_y)$开始，在第一象限内沿椭圆路径顺时针步进；当斜率变为小于 -1 时，将 x 方向的单位步长转化为 y 方向的单位步长。反过来，也可以从$(r_x, 0)$开始，以逆时针方式选取点，并当斜率大于 -1 时，将 y 方向的单位步长改为 x 方向的单位步长。利用并行处理器，则可同时计算两个区域内的像素位置。作为中点椭圆算法串行实现的例子，则以$(0, r_y)$为起点，在第一象限内顺时针沿椭圆路径步进。

在方程（3.37）中，取$(x_c, y_c) = (0, 0)$，定义椭圆函数为

$$f_{\text{ellipse}}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 \tag{3.39}$$

该函数具有下列特性：

$$f_{\text{ellipse}}(x, y) \begin{cases} < 0, & \text{如果}(x, y)\text{位于椭圆边界内} \\ = 0, & \text{如果}(x, y)\text{位于椭圆边界上} \\ > 0, & \text{如果}(x, y)\text{位于椭圆边界外} \end{cases} \tag{3.40}$$

因此，椭圆函数$f_{\text{ellipse}}(x, y)$作为中点算法的决策参数。在每个取样位置，根据椭圆函数沿椭圆轨迹对两个候选像素间中点求值的符号来选择下一个像素。

从$(0, r_y)$开始，在 x 方向取单位步长直到区域 1 和区域 2 之间的分界处（参见图 3.25）；然后转换成 y 方向的单位步长，再覆盖第一象限中剩余的曲线段。在每一步中，需要检测曲线的斜率值。椭圆的斜率可从方程（3.39）中计算得出：

$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y} \tag{3.41}$$

在区域 1 和区域 2 的交界区，$dy/dx = -1$，且

$$2r_y^2 x = 2r_x^2 y$$

因此，移出区域 1 的条件是

$$2r_y^2 x \geqslant 2r_x^2 y \tag{3.42}$$

图 3.26 给出了第一象限内取样位置 x_k+1 处两个候选像素间的中点。假如在前一步中选择了位置(x_k, y_k)，便可通过该中点对决策参数（即椭圆函数（3.39）式）求值来确定沿椭圆轨迹的下一个位置：

$$\begin{aligned} p1_k &= f_{\text{ellipse}}\left(x_k + 1, y_k - \frac{1}{2}\right) \\ &= r_y^2 (x_k + 1)^2 + r_x^2 \left(y_k - \frac{1}{2}\right)^2 - r_x^2 r_y^2 \end{aligned} \tag{3.43}$$

如果 $p1_k < 0$，那么中点位于椭圆内，扫描线 y_k 上的像素更接近于椭圆边界。否则，中点在椭圆之外，或在椭圆边界上，所选的像素应在扫描线 y_k-1 上。

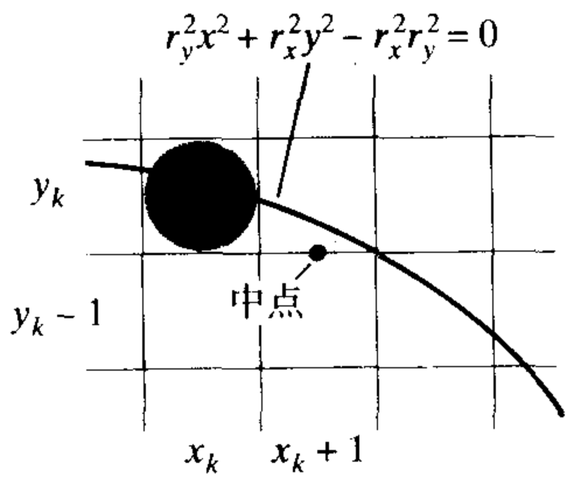


图 3.26　椭圆轨迹上取样位置 x_k+1 处候选像素间的中点

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在下一个取样位置($x_{k+1}+1 = x_k+2$)，区域 1 的决策参数可求值为

$$p1_{k+1} = f_{\text{ellipse}}\left(x_{k+1}+1, y_{k+1}-\frac{1}{2}\right)$$
$$= r_y^2[(x_k+1)+1]^2 + r_x^2\left(y_{k+1}-\frac{1}{2}\right)^2 - r_x^2 r_y^2$$

或

$$p1_{k+1} = p1_k + 2r_y^2(x_k+1) + r_y^2 + r_x^2\left[\left(y_{k+1}-\frac{1}{2}\right)^2 - \left(y_k-\frac{1}{2}\right)^2\right] \tag{3.44}$$

其中，y_{k+1} 根据 $p1_k$ 的符号取值为 y_k 或 y_k-1。

决策参数以下列增量递增：

$$\text{增量} = \begin{cases} 2r_y^2 x_{k+1} + r_y^2, & \text{如果 } p1_k < 0 \\ 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}, & \text{如果 } p1_k \geqslant 0 \end{cases}$$

如同画圆算法，决策参数的增量计算仅需使用加减运算，因为$2r_y^2 x$和$2r_x^2 y$的值也可以通过递增而得到。在起始位置$(0, r_y)$，这两项可以计算为

$$2r_y^2 x = 0 \tag{3.45}$$
$$2r_x^2 y = 2r_x^2 r_y \tag{3.46}$$

当 x 和 y 递增时，通过将等式（3.45）加上$2r_y^2$以及将等式（3.46）减去 $2r_x^2$就得到更新值。对每一步的更新值进行比较，当满足条件（3.42）式时，就从区域 1 移向区域 2。

在区域 1 中，决策参数的初始值可以通过椭圆函数在起始点$(x_0, y_0) = (0, r_y)$求得：

$$p1_0 = f_{\text{ellipse}}\left(1, r_y-\frac{1}{2}\right)$$
$$= r_y^2 + r_x^2\left(r_y-\frac{1}{2}\right)^2 - r_x^2 r_y^2$$

或

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4}r_x^2 \tag{3.47}$$

在区域 2 中，在负 y 方向以单位步长取样。每一步都将计算并得到水平像素间的中点（参见图 3.27）。对于该区域，决策参数将求值为

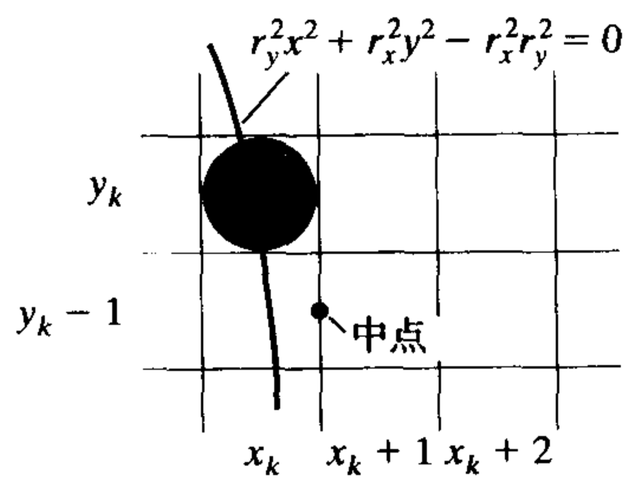


图 3.27　椭圆轨迹在取样位置 y_k-1 处候选像素间的中点

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$p2_k = f_{\text{ellipse}}\left(x_k + \frac{1}{2}, y_k - 1\right)$$
$$= r_y^2\left(x_k + \frac{1}{2}\right)^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2 \tag{3.48}$$

如果$p2_k > 0$，中点位于椭圆边界之外，则选择x_k处的像素；如果$p2_k \leqslant 0$，中点位于椭圆边界之内或之上，我们选择x_{k+1}处的像素。

为了确定区域2中连续的决策参数间的关系，我们在下一个取样步骤$y_{k+1}-1 = y_k-2$对椭圆函数求值：

$$p2_{k+1} = f_{\text{ellipse}}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right)$$
$$= r_y^2\left(x_{k+1} + \frac{1}{2}\right)^2 + r_x^2[(y_k - 1) - 1]^2 - r_x^2 r_y^2 \tag{3.49}$$

或

$$p2_{k+1} = p2_k - 2r_x^2(y_k - 1) + r_x^2 + r_y^2\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] \tag{3.50}$$

其中，x_{k+1}的设置根据$p2_k$的符号可取值为x_k或x_k+1。

当进入区域2时，其初始点(x_0, y_0)就是在区域1中选择的最后位置，那么区域2的初始决策参数为

$$p2_0 = f_{\text{ellipse}}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$
$$= r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2 \tag{3.51}$$

为了简化$p2_0$的计算，以顺时针方向从$(r_x, 0)$开始选择像素位置。然后以正y方向取单位步长，直到区域1中选择的最后位置。

使用方程（3.36）并计算整个椭圆轨迹的像素位置，就能使中点算法适用于生成非标准位置的椭圆。另外，我们可以使用第5章讨论的变换方法将椭圆轴重新定向到标准位置，应用中点算法来确定曲线位置，然后将计算出的像素位置转换成沿原始椭圆轨迹的位置。

假设已经在整数屏幕坐标中给定r_x、r_y和椭圆中心，在中点椭圆算法中我们仅需增量的整数运算来确定决策参数的值。增量r_x^2、r_y^2、$2r_x^2$和$2r_y^2$仅需在程序的开始求值一次。中点椭圆算法可以概括为下列步骤：

中点椭圆算法的步骤

1. 输入r_x、r_y和椭圆中心(x_c, y_c)，并得到椭圆（中心在原点）上的第一个点：

$$(x_0, y_0) = (0, r_y)$$

2. 计算区域1中决策参数的初始值：

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4}r_x^2$$

3. 在区域1中的每个x_k位置，从$k=0$开始，完成下列测试：假如$p1_k < 0$，沿中心在(0, 0)的椭圆的下一个点为(x_{k+1}, y_k)，并且

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

否则，沿椭圆的下一个点为(x_k+1, y_k-1)，并且

$$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

其中

$$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2, \qquad 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$

并且直到$2r_y^2 x \geqslant 2r_x^2 y$。

4. 使用区域1中计算的最后点(x_0, y_0)来计算区域2中参数的初始值：

$$p2_0 = r_y^2\left(x_0 + \frac{1}{2}\right)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2$$

5. 在区域2的每个y_k位置处，从$k=0$开始，完成下列测试：假如$p2_k > 0$，沿中心为(0, 0)的椭圆的下一个点为(x_k, y_k-1)，并且

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

否则，沿椭圆的下一个点为(x_k+1, y_k-1)，并且

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

使用与区域1中相同的x和y增量进行计算，直到$y=0$。

6. 确定其他三个象限中的对称点。
7. 将计算出的每个像素位置(x, y)移到中心在(x_c, y_c)的椭圆轨迹上，并按坐标值绘制点：

$$x = x + x_c, \qquad y = y + y_c$$

例 3.3　中点椭圆算法画图

给定输入椭圆参数$r_x = 8$和$r_y = 6$，我们将给出中点椭圆算法的步骤，从而确定第一象限内椭圆轨迹上的光栅像素位置。决策参数的初始值和增量则计算为

$$2r_y^2 x = 0 \qquad (\text{增量}\ 2r_y^2 = 72)$$
$$2r_x^2 y = 2r_x^2 r_y \qquad (\text{增量}\ -2r_x^2 = -128)$$

对于区域1，圆心在原点的椭圆的初始点为$(x_0, y_0) = (0, 6)$，决策参数的初始值为

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2 = -332$$

下表列出了使用中点算法计算的后续决策参数值和椭圆轨迹的位置。

k	$p1_k$	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	−332	(1, 6)	72	768
1	−224	(2, 6)	144	768
2	−44	(3, 6)	216	768
3	208	(4, 5)	288	640
4	−108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

由于$2r_y^2 x > 2r_x^2 y$，因此椭圆轨迹已经移出区域1。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于区域2，初始点为$(x_0, y_0) = (7, 3)$，初始决策参数为

$$p2_0 = f_{\text{ellipse}}\left(7 + \frac{1}{2}, 2\right) = -151$$

第一象限中椭圆轨迹的其余位置计算为

k	$p1_k$	(x_{k+1}, y_{k+1})	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$
0	−151	(8, 2)	576	256
1	233	(8, 1)	576	128
2	745	(8, 0)	—	—

图3.28给出了第一象限内沿椭圆边界计算出的位置。

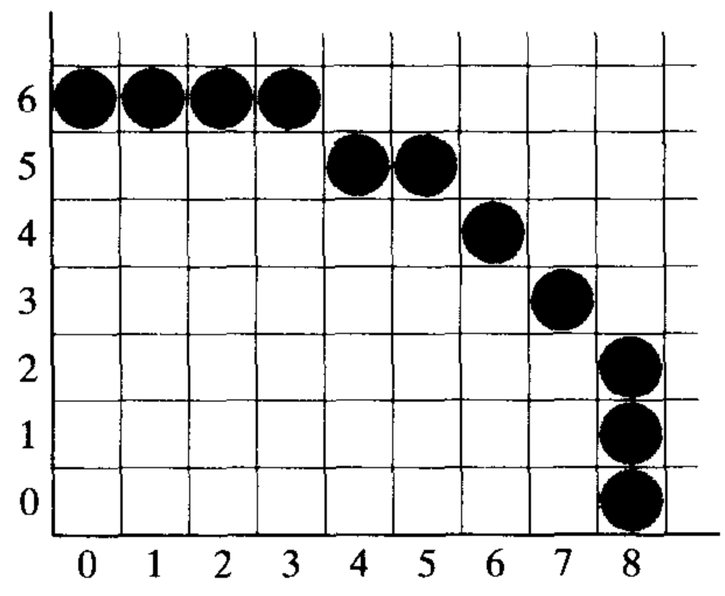


图3.28 中心在原点，$r_x = 8$和$r_y = 6$，使用中点算法计算第一象限内椭圆的像素位置

下面的程序给出了采用中点算法来显示椭圆的例子。椭圆参数Rx、Ry、xCenter和yCenter的值输入过程ellipseMidpoint。第一象限曲线上的位置经计算并传递给过程ellipsePlotPoints。其他三个象限的椭圆位置利用对称性来获得，与这些位置对应的帧缓存中的椭圆颜色由子程序setPixel设定。

```
inline int round (const float a)  { return int (a + 0.5); }

  /*  The following procedure accepts values for an ellipse
   *  center position and its semimajor and semiminor axes, then
   *  calculates ellipse positions using the midpoint algorithm.
   */
  void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry)
  {
     int Rx2 = Rx * Rx;
     int Ry2 = Ry * Ry;
     int twoRx2 = 2 * Rx2;
     int twoRy2 = 2 * Ry2;
     int p;
     int x = 0;
     int y = Ry;
     int px = 0;
     int py = twoRx2 * y;
     void ellipsePlotPoints (int, int, int, int);

     /* Plot the initial point in each quadrant. */
     ellipsePlotPoints (xCenter, yCenter, x, y);

     /* Region 1 */
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    p = round (Ry2 - (Rx2 * Ry) + (0.25 * Rx2));
    while (px < py) {
        x++;
        px += twoRy2;
        if (p < 0)
            p += Ry2 + px;
        else {
            y--;
            py -= twoRx2;
            p += Ry2 + px - py;
        }
        ellipsePlotPoints (xCenter, yCenter, x, y);
    }

    /* Region 2 */
    p = round (Ry2 * (x+0.5) * (x+0.5) + Rx2 * (y-1) * (y-1) - Rx2 * Ry2);
    while (y > 0) {
        y--;
        py -= twoRx2;
        if (p > 0)
            p += Rx2 - py;
        else {
            x++;
            px += twoRy2;
            p += Rx2 - py + px;
        }
        ellipsePlotPoints (xCenter, yCenter, x, y);
    }
}

void ellipsePlotPoints (int xCenter, int yCenter, int x, int y);
{
    setPixel (xCenter + x, yCenter + y);
    setPixel (xCenter - x, yCenter + y);
    setPixel (xCenter + x, yCenter - y);
    setPixel (xCenter - x, yCenter - y);
}
```

3.11 其他曲线

许多曲线函数在对象建模、动画轨迹的描述、数据和函数的图形化以及其他图形应用中是十分有用的。常见的曲线包括圆锥曲线、三角和指数函数、概率分布、通用多项式和样条函数。这些曲线的显示可采用类似于前面讨论的圆和椭圆函数来生成。沿曲线轨迹的位置可直接从表达式$y=f(x)$或参数方程中得到。此外，还可以使用增量中点算法绘制用隐式函数$f(x, y)=0$描述的曲线。

显示一指定的曲线函数的简单方法是使用直线段来逼近曲线。这时，对于要得到沿曲线轨迹的等距线段的端点位置，则可以使用参数表达式。也可以按曲线的斜率选择独立变量，而从显式表达式中生成等距位置。假如$y=f(x)$斜率的绝对值小于1，就选择x作为自变量并对相等的x增量计算y值；当斜率绝对值大于1时，要使用反函数$x=f^{-1}(y)$并在相同的y步长中计算x的值。

使用直线或曲线逼近法可以图示离散坐标点的数据集，我们可以使用直线段来将离散点连结在一起，或采用线性回归（最小二乘法），从而通过单个直线来拟合数据集。非线性最小二乘法用来显示具有某些拟合函数（通常是多项式）的数据组。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

像圆和椭圆一样，许多函数具有对称性，从而可以减少曲线轨迹上坐标位置的计算量。例如，正态分布函数关于中心位置（均值）是对称的，沿正弦曲线一个循环的所有点可以从90°区间内的点生成。

3.11.1　圆锥曲线

通常，我们可以使用二次方程来描述**圆锥曲线**（conic section）：

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0 \tag{3.52}$$

其中，参数 A、B、C、D、E 和 F 的值决定所要显示的曲线类型。给定这组系数，就可以通过对判别式 $B^2 - 4AC$ 求值来确定所要生成的特定圆锥曲线：

$$B^2 - 4AC \begin{cases} < 0, \text{生成椭圆或圆} \\ = 0, \text{生成抛物线} \\ > 0, \text{生成双曲线} \end{cases} \tag{3.53}$$

例如，当 $A = B = 1$、$C = 0$、$D = -2x_c$、$E = -2y_c$ 和 $F = x^2_c + y^2_c - r^2$ 时，就得到圆方程（3.26）。方程（3.52）也能描述退化的圆锥曲线：点和直线。

在有些应用中，圆弧和椭圆弧可以方便地用起始角和终止角表示，如图 3.29 所示。这些弧有时也用它们的端点坐标位置来定义。两种情况下我们都可以使用修改的中点方法来生成弧，或显示一组逼近直线段。

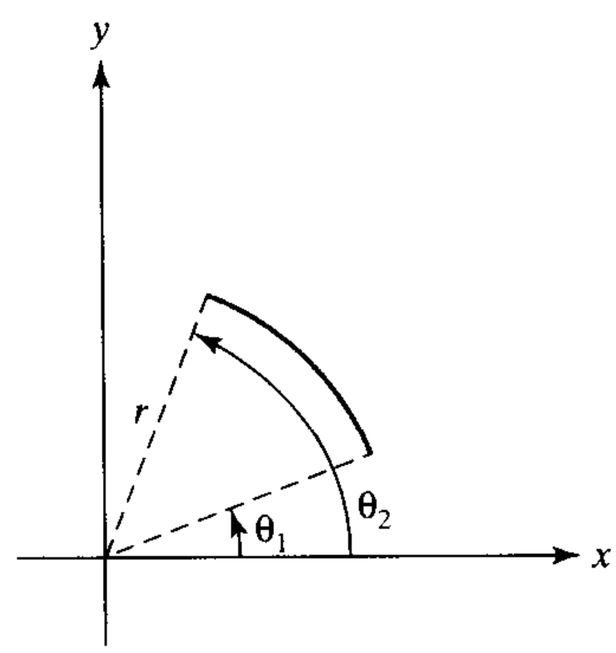


图 3.29　用起始角 θ_1、终止角 θ_2 和半径 r 定义的中心在原点的圆弧

椭圆、双曲线和抛物线在某些动画应用中有独特的用处。这些曲线可以描述受到地球引力、电磁场和原子力作用的物体的运行轨道和其他运动。例如：太阳系的平面轨迹是椭圆，进入均匀地球引力场的物体沿抛物线轨迹运动。对于负 y 方向作用的引力场，图3.30给出了标准位置上的抛物线轨迹。显示物体的抛物线轨迹方程可以写为

$$y = y_0 + a(x - x_0)^2 + b(x - x_0) \tag{3.54}$$

常数 a 和 b 由物体的初始速度 v_0 和均匀引力引起的加速度 g 决定。我们也可利用以秒计量的时间参数 t，根据初始发射点的参数方程来描述这种抛物线轨迹：

$$\begin{aligned} x &= x_0 + v_{x0}t \\ y &= y_0 + v_{y0}t - \frac{1}{2}gt^2 \end{aligned} \tag{3.55}$$

其中，v_{x0} 和 v_{y0} 是初始速度分量，g 的值在地球表面约为 980 cm/s²。沿抛物线轨迹的物体位置就可以按照选定的时间步长计算出来。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

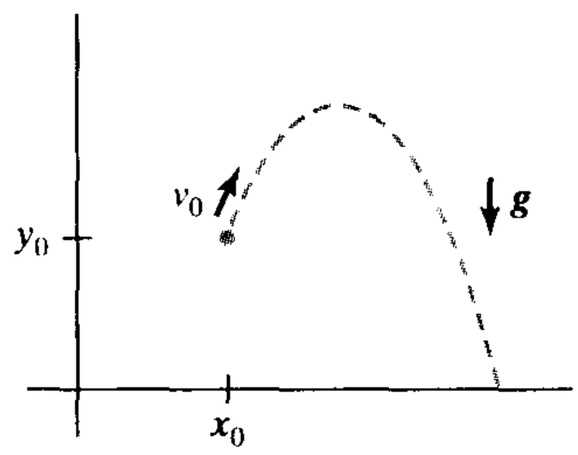


图 3.30 初始点(x_0, y_0)处物体（抛入向下的引力场）的抛物线轨迹

双曲线运动（参见图3.31）发生在有关带电粒子碰撞的问题以及某些引力问题中。例如，彗星或陨星绕太阳的运动是沿双曲线轨迹，并且向外层空间逃逸而从不返回。描述物体运动的特定分支（图 3.31 中的左边或右边曲线）取决于问题中涉及的力。我们可以将图 3.31 中双曲线（中心位于原点）的标准方程写为

$$\left(\frac{x}{r_x}\right)^2 - \left(\frac{y}{r_y}\right)^2 = 1 \tag{3.56}$$

对于左分支，$x \leqslant -r_x$；对于右分支，$x \geqslant r_x$。由于这个方程与标准椭圆方程（3.39）之间的不同仅在于 x^2 和 y^2 项的符号，因此只需对椭圆算法进行细小的改动就可以产生双曲线轨迹上的点。

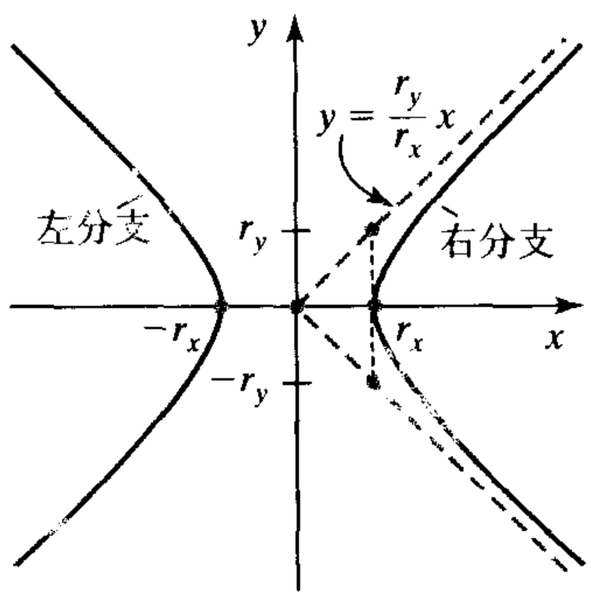


图 3.31 沿 x 轴对称的标准位置上双曲线的左分支和右分支

抛物线和双曲线具有对称轴。例如，由方程（3.55）描述的抛物线关于下列轴是对称的：

$$x = x_0 + v_{x0}v_{y0}/g$$

中点椭圆算法中的方法可直接用于在下面的两个区域内获得抛物线和双曲线轨迹上对称轴一侧的点：(1)曲线斜率的绝对值小于 1；(2)曲线斜率的绝对值大于 1。为此，首先选择方程（3.52）的合适形式，然后利用所选的函数来建立两个区域内决策参数的表达式。

3.11.2 多项式和样条曲线

x 的 n 次多边形函数可以定义为

$$\begin{aligned} y &= \sum_{k=0}^{n} a_k x^k \\ &= a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + a_n x^n \end{aligned} \tag{3.57}$$

其中，n 为非负整数，a_k 是常数且 $a_n \neq 0$，当 $n = 2$ 时得到二次曲线，$n = 3$ 时为三次多项式，$n = 4$ 时为四次曲线，等等；当 $n = 1$ 时得到直线。多项式用于包括对象形状设计、动画轨迹的确定以及在离散数据点集合中数据趋向的图形化等许多图形应用中。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对象形状或运动轨迹的设计中一般先通过指定少量的点来定义一个大概的曲线轮廓，然后利用多项式来拟合选定的点。曲线拟合的一种方法是在每对指定点之间构造三次多项式曲线段，每个曲线段可以通过参数形式描述：

$$\begin{aligned} x &= a_{x0} + a_{x1}u + a_{x2}u^2 + a_{x3}u^3 \\ y &= a_{y0} + a_{y1}u + a_{y2}u^2 + a_{y3}u^3 \end{aligned} \tag{3.58}$$

其中，参数 u 在 0 和 1.0 之间变化。参数方程中 u 的系数值根据曲线段的边界条件确定。边界条件之一是两个相邻曲线段具有公共端点，另一个条件是在边界上匹配两条曲线的斜率，以便得到连续的平滑曲线（参见图 3.32）。利用多项式曲线段形成的这种连续曲线称为**样条曲线**（spline curve），简称为**样条**。还有许多其他的建立样条曲线的方法，我们将在第 8 章研究各种样条的生成方法。

图 3.32　在指定坐标位置用一个三次多项式曲线形成的样条

3.12　并行曲线算法

在曲线生成中使用的并行方法类似于显示线段中使用的方法。我们既可采用顺序算法按曲线分段分配处理器，也可以提出其他方法将处理器分配给屏幕的不同区域。

显示圆的并行中点方法是，将 45° 到 90° 的圆弧分成等长子圆弧，并给每段子圆弧配置一个处理器。然后类似于并行 Bresenham 画线算法，需要对每个处理器建立初始 y 值并确定参数 p_k 的值。接着计算整个子圆弧的像素位置，并通过对称性得到其他八分圆中的位置。同样，并行椭圆中点算法将第一象限内的椭圆弧分割成等长弧，并将这些弧分配给各个处理器，在其他象限中的像素位置可由对称性得到。圆和椭圆的屏幕分段方案是，将每条与曲线相交的扫描线分配给相应的处理器。此时，每个处理器根据圆或椭圆方程计算曲线与扫描线的相交坐标。

为了显示椭圆弧和其他曲线，我们可以简单地使用扫描线分段方法。每个处理器应用曲线方程，从而确定与为其设置的扫描线的交点位置。为每个处理器分配一组像素后，处理器计算每一个像素到曲线的距离（或距离的平方）。如果计算出的距离小于预先指定的值，就绘制出该像素。

3.13　像素编址和对象的几何要素

在显示图元的光栅算法讨论中，已经假定帧缓存坐标使用屏幕像素的中心。现在我们考虑不同编址方法的效果和 OpenGL 等一些图形软件包使用的像素编址方法。

对象的描述即图形程序的输入，由精确的世界坐标位置来给出，其中的每一个位置都是数学上一个无限小的点。然而，当对象经扫描转换进帧缓存后，输入描述变换为对应有限屏幕区域的像素坐标，且显示的光栅图像可能并不严格符合相关的尺寸。如果必须保证世界坐标系中对象的几何描述，我们可以在输入的数学点到有限的像素区域映射中进行补偿。一种方法是简单地按照物体边界与像素区域的覆盖量来调整物体显示的尺寸。例如，如果一个矩形的宽度为 40 cm，那么我们可以调整屏幕显示，这样矩形的宽度为 40 个像素，其中每一个像素表示 1 cm。另一种方法是将世界坐标映射到像素间的屏幕位置，以使物体边界与像素边界对齐，而不是与像素中心对齐。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3.13.1 屏幕网格坐标

图3.33给出了配有划分像素边界的网格线的屏幕区域。其中，一个屏幕位置由一对标识两像素间网格线位置的整数值来给出。任一像素以其左下角进行编址，如图3.34所示。一条直线路径现在被想像成一组网格线交点。例如，端点坐标为(0, 0)、(5, 2)和(1, 4)的折线的数学路径如图3.35所示。

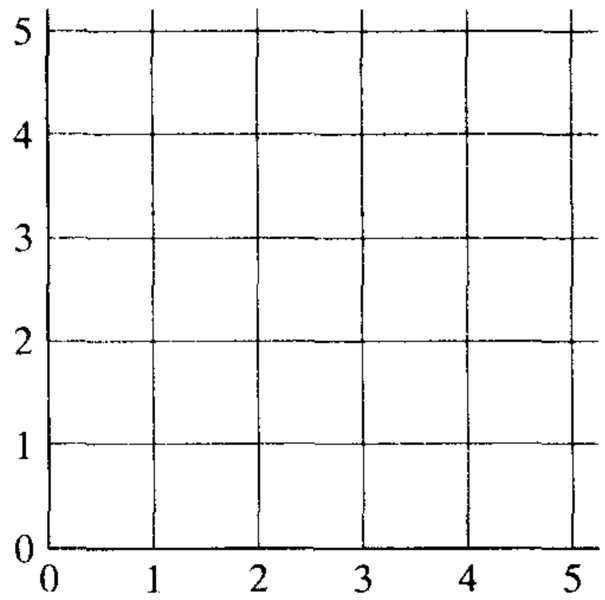


图3.33 利用网格线相交表示坐标位置的屏幕区域的左下部

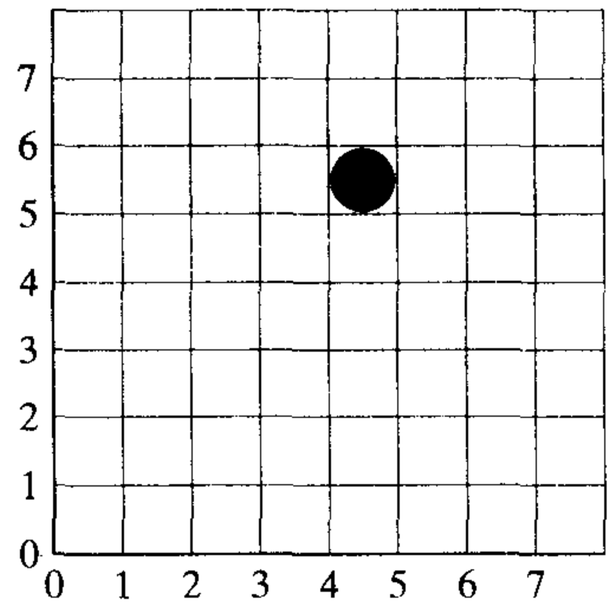


图3.34 在光栅位置(4, 5)处的像素

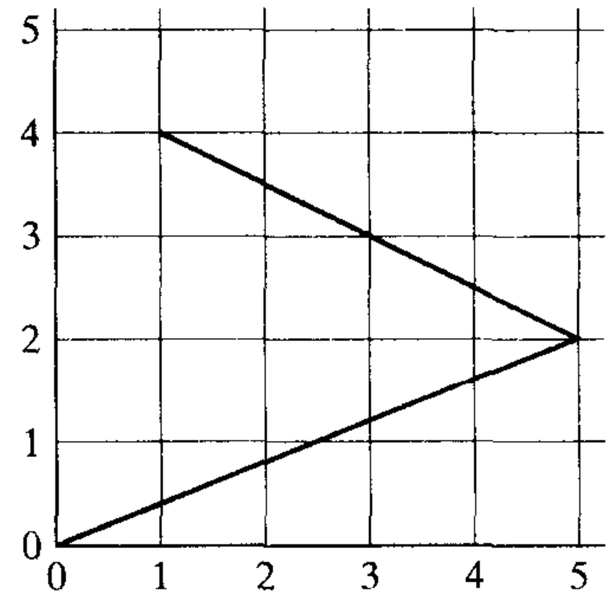


图3.35 屏幕网格坐标位置间两条相连线段的路径

使用屏幕网格坐标，我们让屏幕坐标位置(x, y)上的像素占据对角位置在(x, y)和$(x+1, y+1)$处的单位正方形。这种像素编址方案有很多优点：它避免了半整数像素边界，实现了精确的对象表示，并简化了包含在许多扫描转换算法和其他光栅程序中的处理。

在前面几节中讨论的画线及曲线生成算法仍可用于以屏幕网格坐标表示的输入位置。这些算法中的决策参数将简化为对屏幕网格的间距差的度量，而不再是像素中心之间的间距差。

3.13.2 保持显示对象的几何特性

当将对象的几何描述转换为像素表示时，就将把数学上的点和线转换为有限的屏幕区域。假如要保留由对象的输入坐标指定的原始几何度量，那么在将对象的定义转换到屏幕显示时，就需要考虑像素的有限尺寸。

图3.36给出了在3.5节中由Bresenham画线算法绘出的线段例子。如果将线段端点(20, 10)和(30, 18)解释为精确的网格交点位置，那么可以看到线段不应该延伸到超过屏幕网格位置(30, 18)。如果要像3.5节给定例子那样绘制屏幕坐标(30, 18)的像素，将会显示跨越水平方向11个单位、垂直方向9个单位的一条直线。而对于数学上的一条直线，$\Delta x = 10$、$\Delta y = 8$。如果使用像素的中心位置编址，那么可以通过去除端点中的一个来调整线段显示的长度。如果把屏幕坐标差作为编址的像素边界（如图3.36所示），那么仅用线路径“内部”的一些像素，即通过线段两端点间的像素来绘制直线。对于上述例子，绘制从最左边的像素(20, 10)到最右边的像素(29, 17)之间的像素，这样显示的直线就同数学上从(20, 10)到(30, 18)的直线具有相同的几何大小。

对于封闭区域，通过仅显示在对象边界内的像素来保持输入的几何特性。例如，图3.37(a)中用屏幕坐标顶点定义的矩形，在按照包括用指定顶点连成的边界像素线在内的像素填充显示时会变大。定义的矩形区域是12个单位，而在图3.37(b)中显示的却是20个单位的区域。在图3.37(c)中，仅显示内部像素就保持了原始的矩形度量。输入的矩形的右边界为$x = 4$，为了在显示中保持这条边界，将最右边的像素网格坐标设置在$x = 3$处。这样垂直列中的像素跨越的间隔为$x = 3$到$x = 4$。同样，矩形的上部边界（数学上）在$y = 3$处，因而我们将显示矩形的上部像素设置在$y = 2$处。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

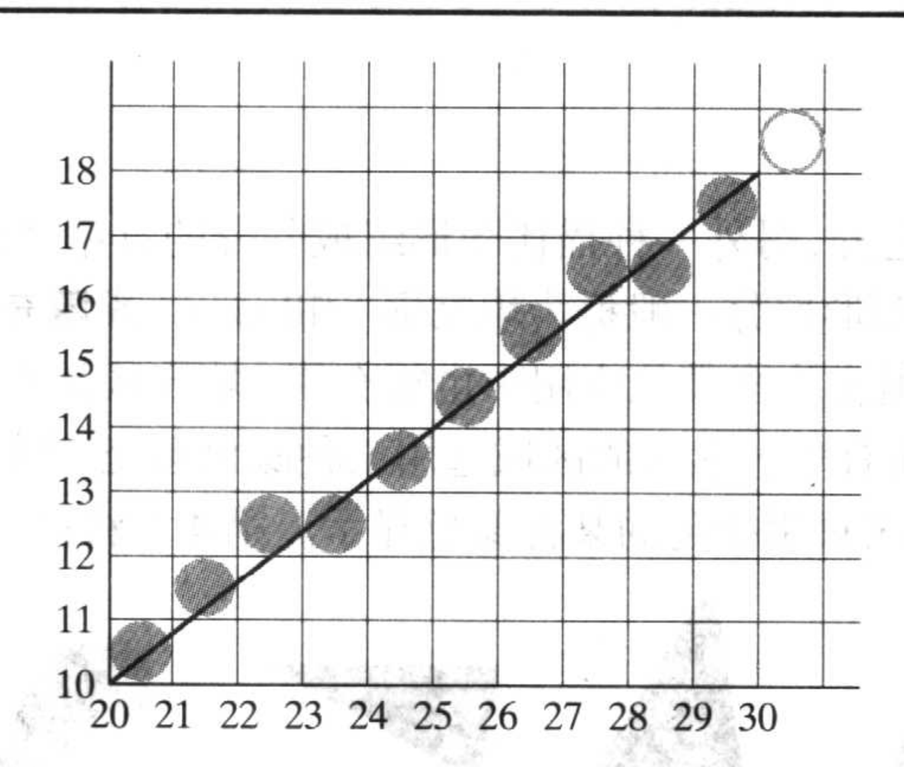


图 3.36　网格端点坐标为(20, 10)和(30, 18)的线路径及相应的像素显示

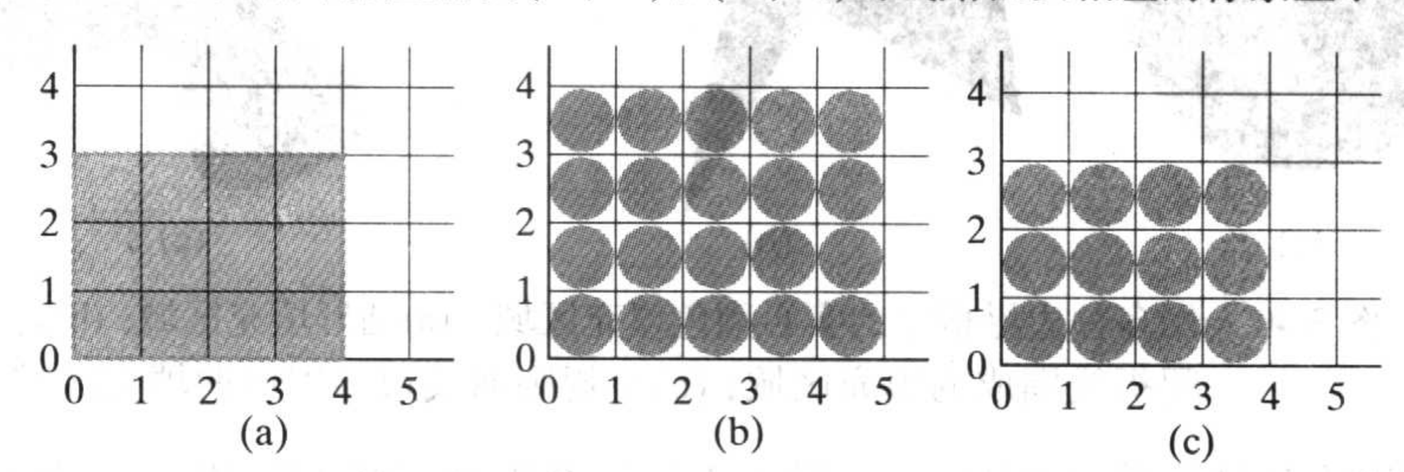


图 3.37　矩形的转换：(a)屏幕坐标顶点在(0, 0)、(4, 0)、(4, 3)和(0, 3)处；(b)包含右边和顶端边界；(c)保持几何大小

这种沿对象边界的有限像素宽度的补偿能应用于其他多边形和曲线图形，从而使光栅显示保持输入对象的要求。例如，半径为 5、中心位置为(10,10)的圆，利用中点算法并以像素中心作为屏幕坐标位置的显示结果如图 3.38 所示。但绘制出的圆的直径为 11。要绘制直径为 10 的圆，就需要修改画圆算法，如同图 3.39 那样缩短每条像素扫描线和每个像素列。一种方法是从屏幕坐标(10,5)开始，在第三象限中沿圆弧顺时针生成点。对于每个生成的点，其他七个圆对称点通过沿扫描线将 x 坐标值减 1 和沿像素列将 y 坐标值减 1 而生成。类似的方法可以用于椭圆算法中，以保持椭圆显示的指定特性。

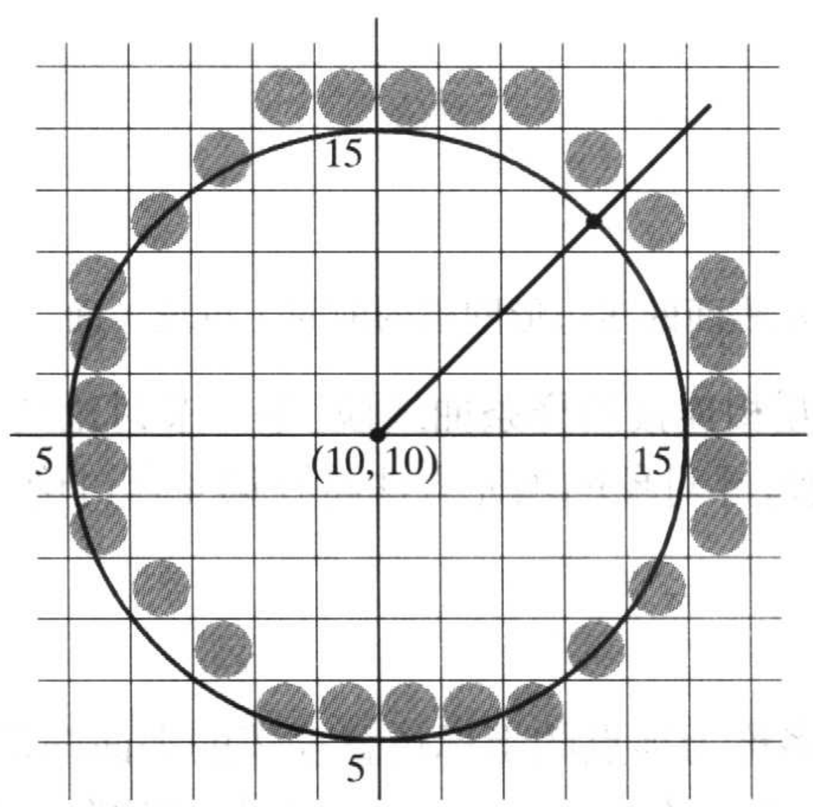


图 3.38　以屏幕坐标圆路径和中点画圆算法绘制的半径为 5 的圆

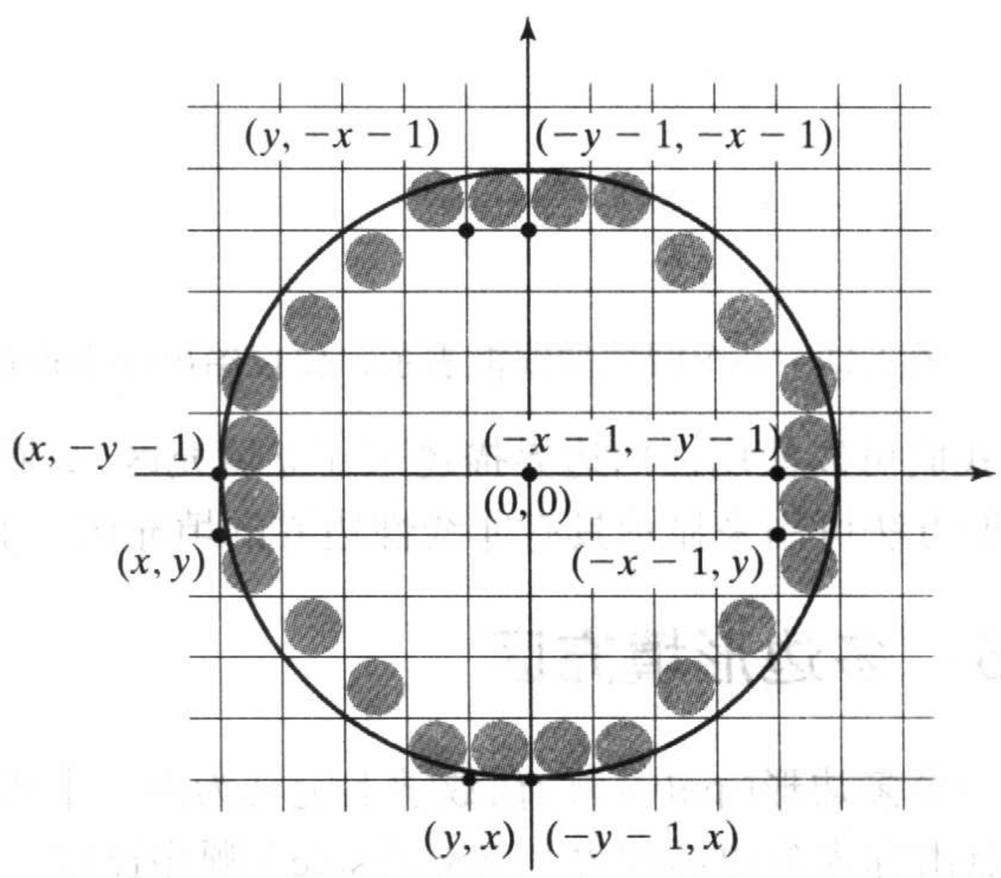


图 3.39　为保持指定的圆直径 10 而对图 3.38 绘制的圆进行修改

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3.14 填充区图元

除了点、直线段和曲线之外，另外一种描述图形组成部分的有用结构是使用某种颜色或图案进行填充的区域。这种类型的图形部分一般称为**填充区**（fill area）或**填充的区域**（filled area）。通常，填充区用于描述实体表面，但在许多其他应用中也很有用。填充区常常是一个平表面，主要是多边形。但一般而言，图形中可能有多种形状的区域选用某种颜色填充。图3.40给出了几种可能的填充区形状。目前我们假定填充区只用指定的某种颜色显示。第4章将讨论其他的填充选项。

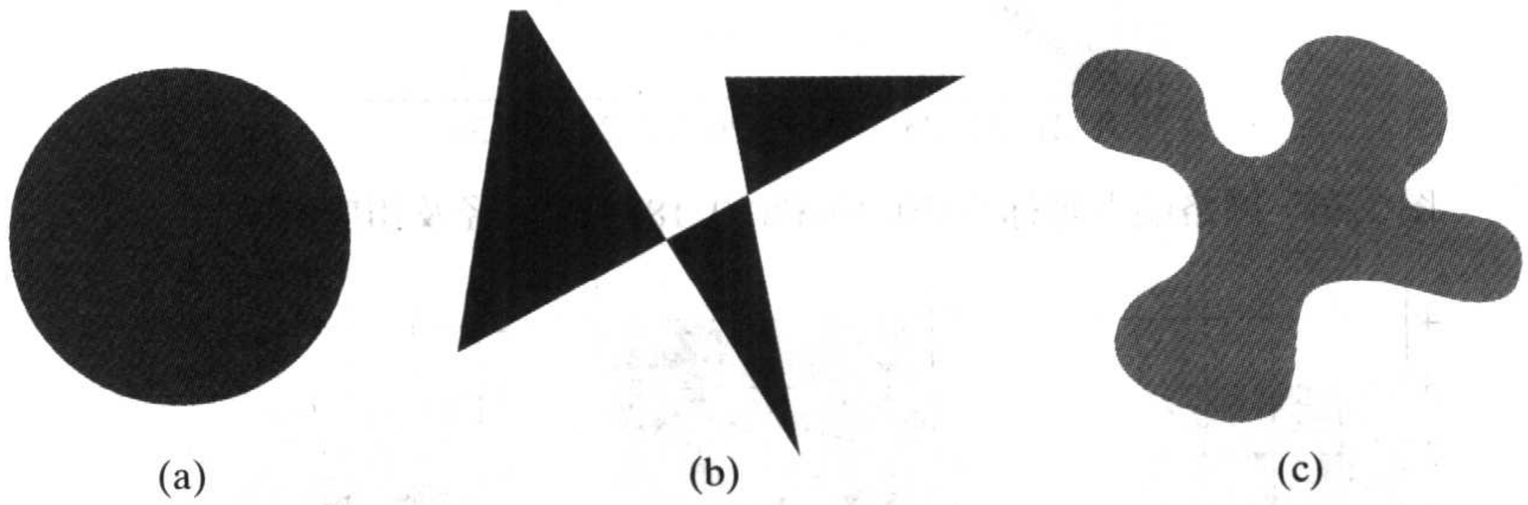


图 3.40 用多种边界指定的实心颜色填充区域：(a)圆形填充区域；(b)封闭折线围成的填充区域；(c)不规则曲线边界围成的填充区域

尽管有可能使用各种形状，但图形库一般不支持任意填充形状的描述。多数库函数要求填充区指定为多边形。由于多边形有线性边界，因而比其他填充形状更容易处理。另外，多数曲面可用一组适当的多边形面片来逼近，就如曲线可用一组直线段逼近一样。在使用光照效果和表面处理时，逼近曲面可以显示得相当逼真。利用多边形面片对一曲面进行的逼近有时称为表面细分（surface tessellation），或者可以使用多边形网格（polygon mesh）来拟合曲面。图3.41给出了一个用轮廓线形式的多边形网格逼近的金属圆柱体。作为线框图（wire-frame），因其仅给出一般标识表面结构的多边形的边，所以这类图能快速地进行显示。线框模型经绘制处理生成具有自然材料表面的显示。使用一组多边形面片描述的对象称为**标准图形对象**（standard graphics object）或称为**图形对象**（graphics object）。

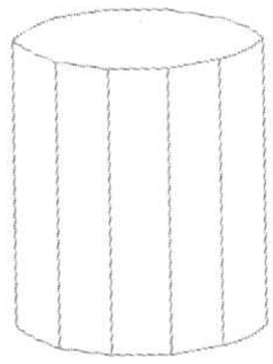


图 3.41 一个圆柱的线框表示，仅仅给出用来近似表示表面的多边形网格的前向（可见）面

我们可以使用任何边界描述来建立填充区，如圆或互相连接的样条曲线段。下一节讨论的一些多边形方法可用来显示具有非线性边界的填充区。曲线边界对象的填充方法将在第4章讨论。

3.15 多边形填充区

一个**多边形**（polygon）在数学上定义为由三个或更多称为顶点的坐标位置描述的平面图形，这些顶点由称为多边形的边（edge或side）顺序连接。进一步来看，几何上要求多边形的边除了端点之外没有其他的公共点。因此，根据定义，一个多边形在其单一平面上必须有其所有的顶点且边之间无交叉。多边形的例子有三角形、矩形、八边形和十六边形等。有时，任一有封闭折线边界的平

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

面图形暗指一个多边形，而若其没有交叉边则称为标准多边形（standard polygon）或简单多边形（simple polygon）。为了避免对象引用的混淆，我们把术语“多边形”限定为那些有封闭折线边界且无交叉边的平面图形。

在计算机图形学的应用中，用于指定多边形的一组顶点并不严格地在一个平面上。这可能由数值计算的舍入误差、对坐标位置的选择错误或更一般地由于使用一组多边形面片逼近曲面而引起。纠正该问题的一种方法是简单地将指定曲面网分割成三角形。但有时必须保留网络面片的原始形状，以便开发用平面图形逼近非平面多边形的方法。我们将在平面方程的有关内容中讨论如何计算这些逼近平面。

3.15.1 多边形分类

多边形的一个**内角**（interior angle）是由两条相邻边形成的多边形边界之内的角。如果一个多边形的所有内角均小于180°，则该多边形为**凸**（convex）多边形。凸多边形的一个等价定义是它的内部完全在它的任一边及其延长线的一侧。同样，如果任意两点位于凸多边形的内部，其连线也位于内部。不是凸多边形的多边形称为**凹**（concave）多边形。图3.42给出了凸多边形和凹多边形的例子。

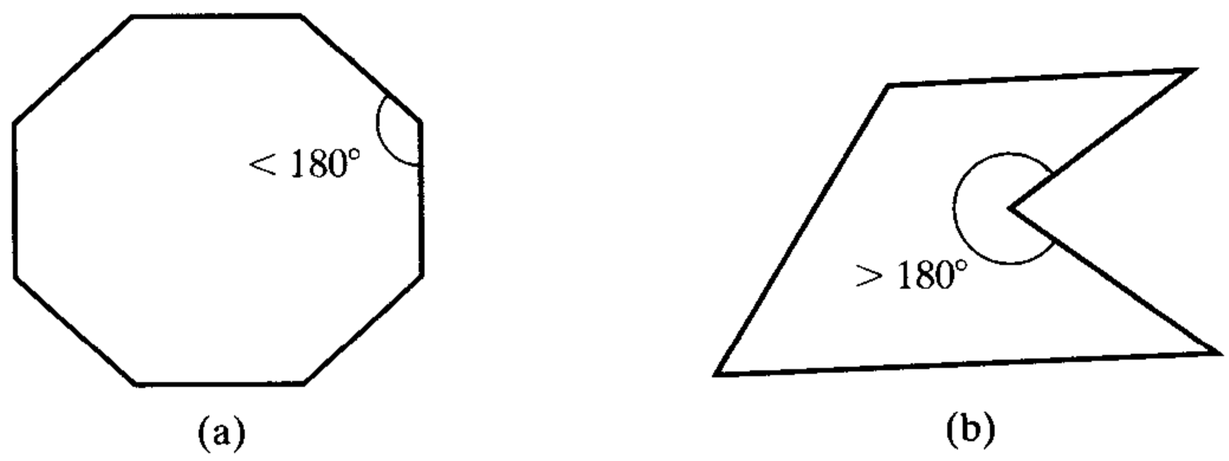


图3.42 (a)一个凸多边形和(b)一个凹多边形

术语**退化多边形**（degenerate polygon）常用来描述共线或重叠坐标位置的顶点集。共线顶点生成一线段。重叠顶点位置可以生成有多余线段、重叠边或长度为0的边的多边形。有时术语退化多边形也用于少于三个坐标位置的顶点队列。

为了软件的鲁棒性，图形软件包可以拒绝退化或非平面的顶点集。但这要求额外的识别该问题的处理，因此图形系统常把这种考虑留给程序员。

凹多边形也会有相关的一些问题。对凹多边形的填充算法和其他子程序的实现比较复杂，因此在处理前常将凹多边形分割成一组凸多边形以便提高效率。和其他的多边形预处理算法一样，凹多边形的分割一般也不包括在图形函数库中。OpenGL等图形软件包要求所有的填充多边形为凸多边形。有些系统仅接受三角形填充区，这将大大简化许多显示函数。

3.15.2 识别凹多边形

凹多边形中至少有一个内角大于180°。凹多边形某些边的延长线会与其他边相交且有时一对内点连线会与多边形边界相交。因此，我们可以将凹多边形的这些特征中的任意一个作为基础设计识别算法。

如果为每一边建立一个向量，则可使用相邻边的叉积来测试凸凹性。凸多边形的所有向量叉积均同号。因此，如果某些叉积取正值而另一些为负值，可确定其为凹多边形。图3.43给出了识别凹多边形的边向量、叉积方法。

识别凹多边形的另一个方法是观察多边形顶点位置与每条边延长线的关系。如果有些顶点在某一边延长线的一侧而其他一些顶点在另一侧，则该多边形为凹多边形。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

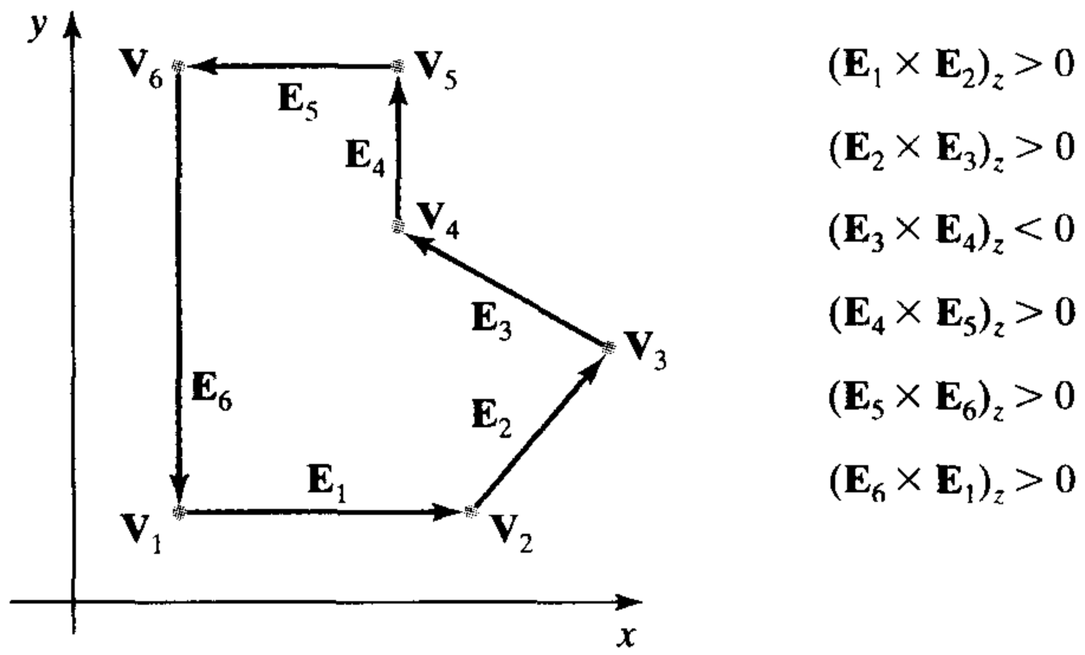


图 3.43 通过计算连续两边向量的叉积来识别凹多边形

3.15.3 分割凹多边形

一旦识别出凹多边形，我们可以将它切割成一组凸多边形。这可使用边向量和边叉积来完成。我们可以利用顶点和边延长线的关系来确定哪些顶点在其一侧、哪些顶点在另一侧。在下面的算法中，我们假定所有多边形均在 xy 平面上。当然，在世界坐标系中描述的多边形的初始位置可能不在 xy 平面上，但我们可以使用第 5 章讨论的变换方法将它们移到 xy 平面上。

对于分割凹多边形的**向量方法**（vector method），我们首先要形成边向量。给定相继的向量位置 $\mathbf{V}_k$ 和 $\mathbf{V}_{k+1}$，定义边向量：

$$\mathbf{E}_k = \mathbf{V}_{k+1} - \mathbf{V}_k$$

接着按多边形边界顺序计算连续的边向量的叉积。如果有些叉积的 z 分量为正而另一些为负，则多边形为凹多边形；否则，多边形为凸多边形。这意味着不存在三个连续的顶点共线，即不存在连续两个边向量其叉积为0。如果所有顶点共线，则得到一个退化多边形（一条线段）。我们可以通过逆时针方向处理边向量来应用向量方法。如果有一个叉积的 z 分量为负值（如图 3.43 所示），那么多边形为凹且可沿叉积中第一边向量的直线进行切割。下面的例子给出了分割凹多边形的这一方法。

例 3.4 分割凹多边形的向量方法

图 3.44 给出了一个有 6 个顶点的凹多边形。该多边形的边向量表示为

$$\mathbf{E}_1 = (1, 0, 0) \qquad \mathbf{E}_2 = (1, 1, 0)$$
$$\mathbf{E}_3 = (1, -1, 0) \qquad \mathbf{E}_4 = (0, 2, 0)$$
$$\mathbf{E}_5 = (-3, 0, 0) \qquad \mathbf{E}_6 = (0, -2, 0)$$

这里的 z 分量均为 0，因为所有边均在 xy 平面上。两个连续的边向量的叉积 $\mathbf{E}_j \times \mathbf{E}_k$ 是垂直于 xy 平面的向量，其分量等于 $E_{jx}E_{ky} - E_{kx}E_{jy}$：

$$\mathbf{E}_1 \times \mathbf{E}_2 = (0, 0, 1) \qquad \mathbf{E}_2 \times \mathbf{E}_3 = (0, 0, -2)$$
$$\mathbf{E}_3 \times \mathbf{E}_4 = (0, 0, 2) \qquad \mathbf{E}_4 \times \mathbf{E}_5 = (0, 0, 6)$$
$$\mathbf{E}_5 \times \mathbf{E}_6 = (0, 0, 6) \qquad \mathbf{E}_6 \times \mathbf{E}_1 = (0, 0, 2)$$

因为叉积 $\mathbf{E}_2 \times \mathbf{E}_3$ 的 z 分量为负，我们沿向量 $\mathbf{E}_2$ 的直线分割多边形。该边的直线方程中的斜率为 1 而 y 轴截距为 -1。然后我们可以确定这条直线和其他边的交点来将多边形分割成两片。其他边叉积不为负，所以得到的两个多边形均为凸多边形。

我们还可以使用**旋转法**（rotational method）来分割凹多边形。沿多边形的边的逆时针方向，逐一将顶点 $\mathbf{V}_k$ 移到坐标系原点。然后顺时针旋转多边形，使下一顶点 $\mathbf{V}_{k+1}$ 落在 x 轴上。如果再下一个

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

顶点 $\mathbf{V}_{k+2}$ 位于 x 轴下面，则多边形为凹。然后我们利用 x 轴将多边形分割成两个新多边形，并对这两个新多边形重复使用凹测试。上述步骤一直重复到多边形中所有顶点均经过测试。对象位置旋转和平移的方法将在第 5 章讨论。图 3.45 给出了分割凹多边形的旋转法。

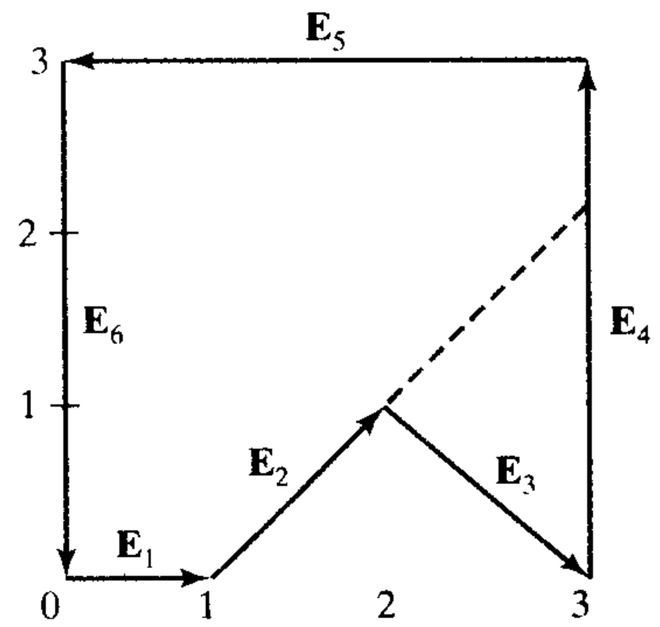


图 3.44　使用向量方法分割凹多边形

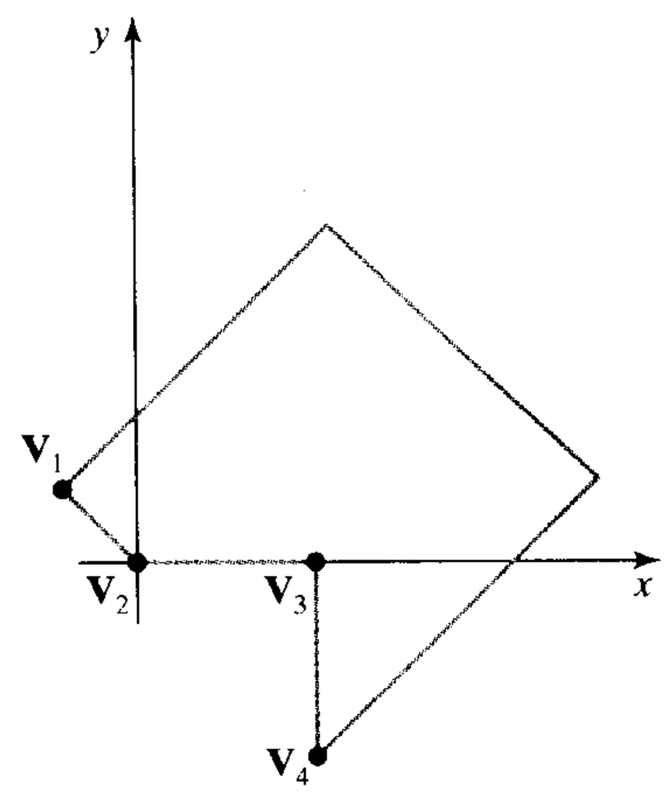


图 3.45　使用旋转方法分割一个凹多边形。在将 $\mathbf{V}_2$ 移到坐标系原点且将 $\mathbf{V}_3$ 旋转到 x 轴后，发现 $\mathbf{V}_4$ 在 x 轴下方。故可沿 $\overline{\mathbf{V}_2\mathbf{V}_3}$（$\mathbf{V}_2\mathbf{V}_3$ 上加一横）即 x 轴分割该多边形

3.15.4　将凸多边形分割成三角形集

一旦有了一个凸多边形的顶点集，我们可以将其变成一组三角形。这通过将任意顺序的三个连续顶点定义为一个新多边形（三角形）来实现。然后将三角形的中间顶点从多边形原顶点队列中删除。接着使用相同的过程处理修改后的顶点队列来分出另一个三角形。这种分割一直进行到原多边形仅留下三个顶点，它们定义三角形集中的最后一个。凹多边形也可以使用这种方法分割为三角形集，但要求每次三顶点形成的内角小于 180°（一个“凸”角）。

3.15.5　内 – 外测试

各种图形处理经常需要鉴别对象的内部区域。识别简单对象如凸多边形、圆或椭圆的内部通常是一件很容易的事情。但有时我们必须处理较复杂的对象。例如，我们可能描述一个图 3.46 所示的有相交边的复杂填充区。在该形状中，xy 平面上哪一部分为对象边界的“内部”、哪一部分为“外部”并不总是一目了然的。奇偶规则和非零环绕规则是识别平面图形内部区域的两种常用方法。

奇偶规则（odd-even rule）也称奇偶性规则（odd-parity rule）或偶奇规则（even-odd rule），该规则从任意位置 **P** 到对象坐标范围以外的远点画一条概念上的直线（射线），并统计沿该射线与各

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

边的交点数目。假如与这条射线相交的多边形边数为奇数，则**P**是内部（interior）点，否则**P**是外部（exterior）点。为了得到精确的相交边数，必须确认所画的直线不与任何多边形顶点相交。图3.46(a)示例了根据奇偶规则得到的存在自相交的一组边的内部和外部区域。我们可以使用该过程对两个同心圆或两个同心多边形的内部填上指定颜色。

另一种定义内部区域的方法是**非零环绕数**（nonzero winding-number）规则。该方法统计多边形边以逆时针方向环绕某一特定点的次数，这个数称为**环绕数**。将二维对象的内部点定义为具有非零值的环绕数。在对多边形应用非零环绕数规则时，将环绕数初始化为零。设想从任意位置**P**到对象坐标范围外的远处一点画一条射线。所选择的射线不能与多边形的任何顶点相交。当从**P**点沿射线方向移动时，统计穿过该射线的边的方向。每当多边形从右到左穿过射线时，边数加1；从左到右时，边数减1。在所有穿过的边都已计数后，环绕数的最后值决定了**P**的相对位置。假如环绕数为非零，则**P**将定义为内部点，否则**P**是外部点。图3.46(b)给出了使用非零环绕数规则的自相交多边形的内部和外部区域。对于多边形和圆等简单对象，非零环绕数规则和奇偶规则给出了相同的结果；但对于比较复杂的形状，两种方法可能会产生如图3.46所示的不同的内部和外部区域。

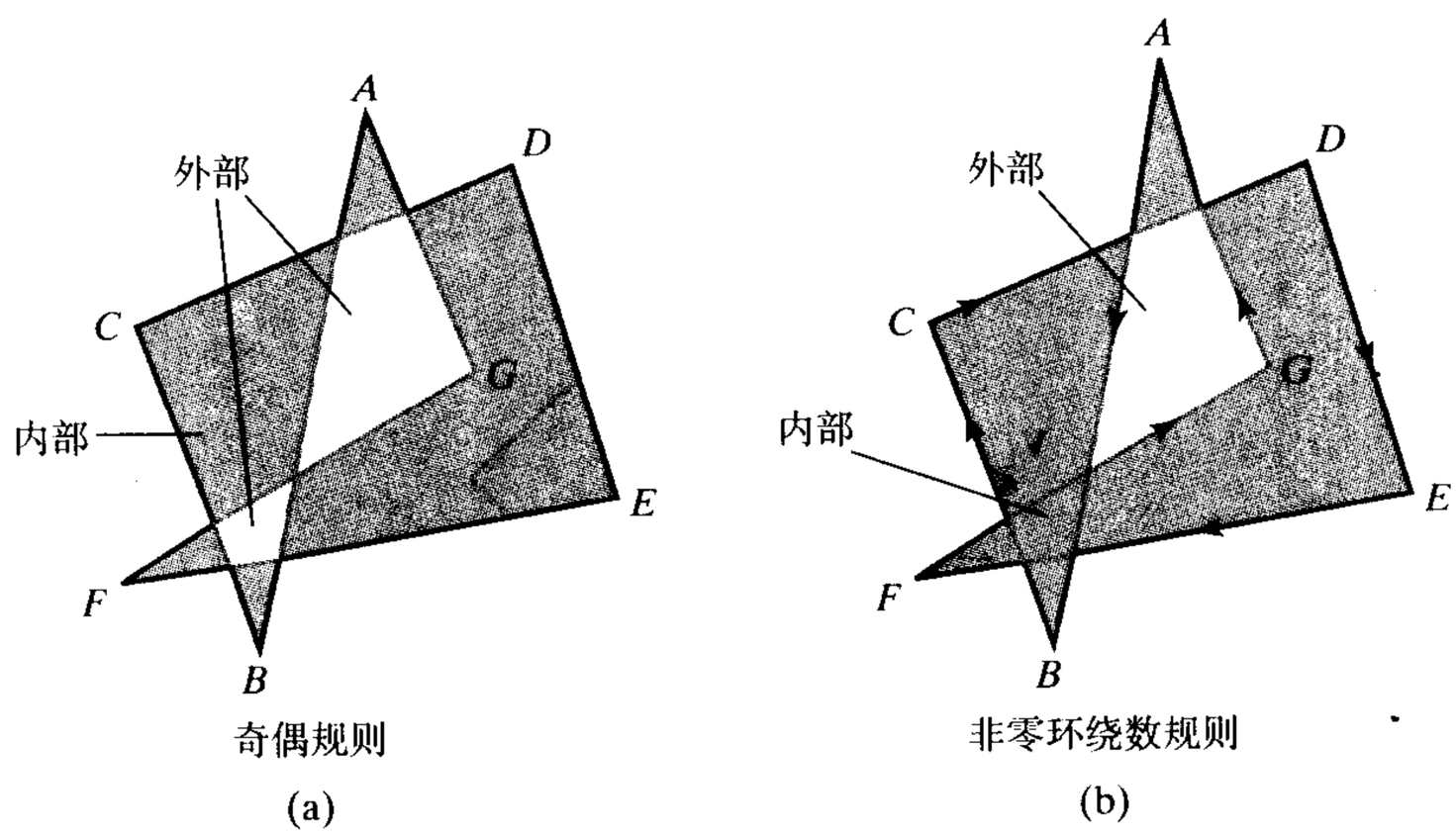


图3.46 自相交多边形的内部和外部区域识别

一种确定有向边界穿越的方法是沿对象边建立向量（或边界线），将从**P**点出发的射线向量**u**与穿过射线的每条边的边向量**E**进行叉积运算。假定在xy平面上有一个二维对象，每一叉积的方向或者在$+z$方向、或者在$-z$方向。如果对于某一特定的边，叉积$\mathbf{u} \times \mathbf{E}$的$z$分量为正，那么边从右到左穿过射线，环绕数加1。否则，边从左到右穿越射线，环绕数减1。边向量可以使用边的终止端点位置减去边的起始顶点位置进行计算。

计算有向边界穿越的更简单的方法是使用点积代替叉积。为此，建立与向量**u**正交且当站在**P**点沿**u**方向看时从右到左方向的一个向量。如果**u**的分量表示为(u_x, u_y)，则这个垂直于**u**的向量的分量为$(-u_y, u_x)$（参见附录A）。现在，如果该正交向量与边界线向量的点积为正，表示从右向左穿越，让环绕数加1。否则，边界从左向右穿过参考线，环绕数减1。

非零环绕数规则将有些区域定为内部而奇偶规则将其定为外部，这在有些应用中可以是有益的。一般情况下，平面图形可定义为多个不相连的组成部分，为每一不相连的边界集指定的方向可用于指定内部和外部。这种例子有字符（如阿拉伯数字和标点符号）、拼接的多边形及同心圆或椭圆。对于曲线，奇偶规则通过计算与曲线路径的交点来应用。类似地，使用非零环绕数规则，我们需要在曲线从**P**点出发的射线相交点处计算切向量。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

非零环绕数规则的变形可用于以另一种方法定义的内部区域。例如，我们可以在环绕数为正或为负时定义一个点为内点。我们也可以使用任何其他的规则来生成各种填充区域。有时，使用布尔操作指定填充区域为两区域的混合。布尔操作的一种实现方法是使用非零环绕数规则的一个变形。在这种方法下，先为每一区域定义简单的无相交的边界。然后如果考虑每一边界的方向为逆时针，那么两区域的并包含那些使环绕数为正的点（参见图3.47）。类似地，逆时针边界的两区域的交包含那些使环绕数大于1的点，如图3.48所示。要建立两区域差的填充区，如A－B，我们可以引入逆时针边界的A和顺时针边界的B。其差区域（参见图3.49）即为那些使环绕数为正的点。

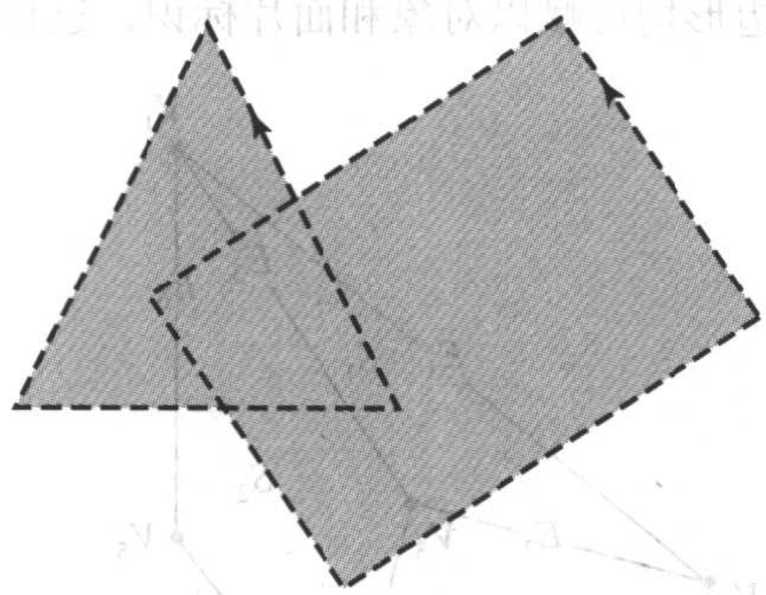


图3.47 环绕数为正定义的填充区。该填充区为两个逆时针边界区域的并

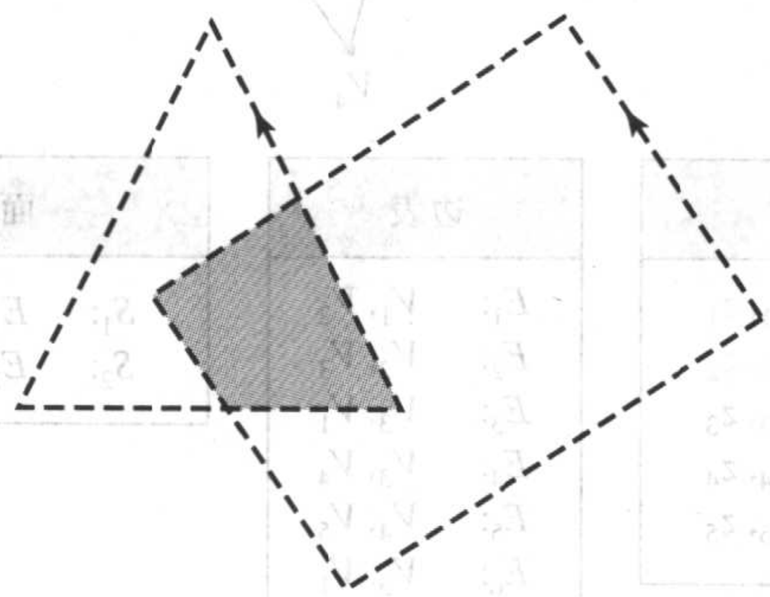


图3.48 环绕数大于1定义的填充区。它是两个逆时针边界区域的交

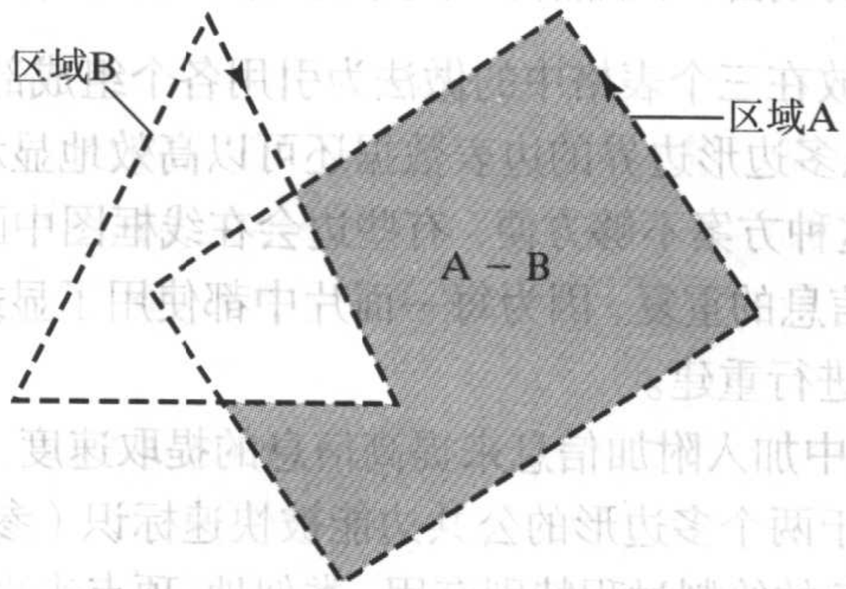


图3.49 环绕数为正定义的填充区。它是两区域的差A－B，其中A使用正边界方向（逆时针），B使用负边界方向（顺时针）

3.15.6 多边形表

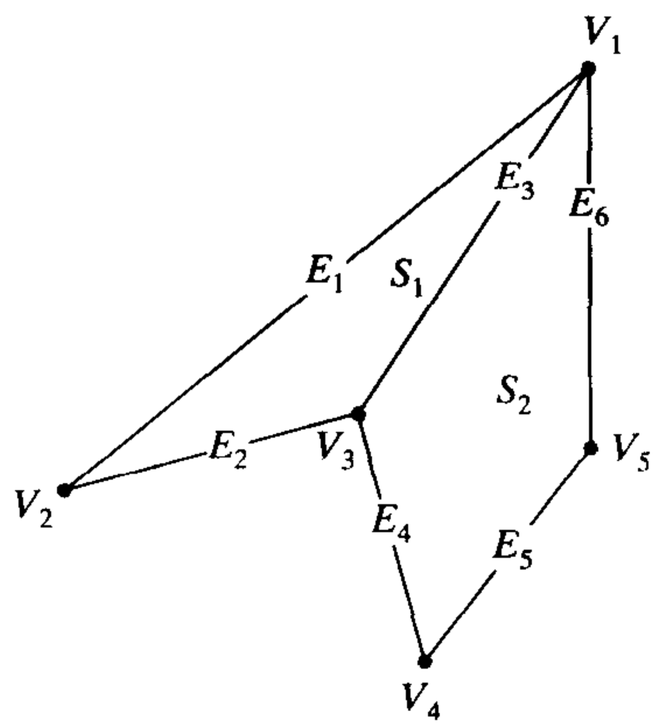
场景中的对象一般用一组多边形面片来描述。实际上，图形软件包经常提供以多边形网格形式描述表面形状的函数。对每一个对象的描述包括指定多边形面片的几何信息和其他表面参数（如颜

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

色、透明性及光反射特性）。在输入每个多边形的信息时，数据放进一些表格中等待后续处理、显示和场景的对象管理。这些多边形数据表分成两组来组织：几何数据表和属性数据表。几何数据表包含顶点坐标和标识多边形面片空间方向的参数。对象的属性信息包含指定对象的透明程度及其表面的反射性能和纹理特征。

场景中对象的几何数据简单地组织为三张表：顶点表、边表和面片表（简称为面表）。对象的每一顶点的坐标存储在顶点表中。边表包含指向顶点表的指针以确定每一多边形的边的端点。而面片表包含指向边表的指针以确定每个多边形的边。图3.50给出了对象表面两个相邻多边形面片的相关表。另外，对象及其组成多边形均可赋以对象和面片标识，这样可以比较容易地引用它们。



顶点表	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

边表	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

面片表	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

图 3.50　一对象中分别由 6 个顶点及 5 个顶点形成的两个相邻多边形的几何数据表

如图3.50中将几何数据放在三个表格中的做法为引用各个组成部分（每个对象的顶点、边、和面片）提供了方便。使用标识多边形边界的边表数据还可以高效地显示对象。另一种安排是只用两张表：顶点表和面片表。但这种方案不够方便，有些边会在线框图中画两次。另一种可能是仅使用一张面片表，这会引起坐标信息的重复，因为每一面片中都使用了显式的坐标值。边和面的关系也必须从面片表的顶点清单中进行重建。

可以在图 3.50 的数据表中加入附加信息来提高信息的提取速度。例如，扩充边表使其包含指向面片表的指针，从而使属于两个多边形的公共边能被快速标识（参见图 3.51）。这对需要从一个多边形到下一个平滑过渡着色的绘制过程特别有用。类似地，顶点表也可以扩充指向相应边的指针，以便快速提取信息。

另外一些常常存储在数据表中的几何信息包括每一条边的斜率和多边形边、多边形面片及场景中每一对象的坐标范围。输入顶点时，我们可以计算边的斜率并通过扫描坐标值来确认单条线段及多边形的最小和最大的 x、y、z 值。由于几何数据表可以包含场景中大量的复杂对象的顶点和边，检查其一致性和完整性是十分重要的。特别是在交互应用中有可能在描述顶点、边和多边形的过程中，产生一些输入错误使对象的显示变形。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

数据表中包含的信息越多，错误的检查越容易。因此，当使用三个数据表（顶点、边和面片）时错误的检查比较容易，因为这个方案提供了最多的信息。可由图形软件包来完成的测试有：(1)每一顶点至少有两条边以其作为端点，(2)每条边至少是一个多边形的组成部分，(3)每一个多边形都是封闭的，(4)每个多边形至少有一条共享边，(5)如果边表包含指向多边形的指针，那么由多边形指针引用的每条边都有一个反向指针指回该多边形。

E_1:	V_1, V_2, S_1
E_2:	V_2, V_3, S_1
E_3:	V_3, V_1, S_1, S_2
E_4:	V_3, V_4, S_2
E_5:	V_4, V_5, S_2
E_6:	V_5, V_1, S_2

图 3.51　图 3.50 中面的边表扩充了指向面片表的指针

3.15.7　平面方程

要完成一个三维场景的显示，图形系统要对输入数据进行若干步处理。这些处理包括在观察流水线中的模型坐标和世界坐标描述的变换、可见面判定及对各面片的绘制。其中有些处理需要对象表面的空间方向信息。该信息可从顶点坐标值和描述多边形表面的方程中获得。

场景中的每一个多边形包含在一个无限平面中。平面的一般方程为

$$Ax + By + Cz + D = 0 \tag{3.59}$$

其中(x, y, z)是平面中的任一点，系数 A、B、C、D（称为平面参数，plane parameter）是描述平面空间特征的常数。用三个平面上非共线点的坐标代入，获得三个平面方程，求解后便得出 A、B、C、D 的值。为此，可选择逆时针凸多边形的三个连续顶点(x_1, y_1, z_1)、(x_2, y_2, z_2)和(x_3, y_3, z_3)并解下列联立方程组来求 A/D、B/D 和 C/D：

$$(A/D)x_k + (B/D)y_k + (C/D)z_k = -1, \qquad k = 1, 2, 3 \tag{3.60}$$

这组方程的解可使用 Cramer 规则以行列式形式求出：

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} \tag{3.61}$$

展开行列式，可得计算平面系数的表达式：

$$\begin{aligned} A &= y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2) \\ B &= z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2) \\ C &= x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \\ D &= -x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1) \end{aligned} \tag{3.62}$$

这种计算对任意三个坐标位置均有效，包括 $D = 0$ 的情况。将顶点坐标和其他信息装入多边形的数据结构中后，计算每个多边形 A、B、C、D 的值并与其他的多边形信息一起保存。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

定义一个多边形面片的坐标有可能不在一个平面中。这样的问题可通过将面片分割成若干个三角形来解决。也可以为该顶点集找到一个逼近平面来解决。获得逼近平面的一种方法是将顶点集分成若干子集，每个子集内有三个顶点，计算每一子集的平面参数 A、B、C、D。将每一组平面参数求平均值就得到逼近平面。另一种方法是将顶点集投影到一个坐标平面上。然后按多边形面积与其在 yz 平面上投影面积之比求得 A，按与其在 xz 平面上投影之比求得 B，按与其在 xy 平面上投影之比求得 C。投影方法较多地用于光线跟踪应用中。

3.15.8 前向面与后向面

由于我们通常处理包围对象内部的多边形表面，因此需要区分每个面的两侧。向着对象内部的一侧称为**后向面**（back face），可见或朝外的一侧称为**前向面**（front face）。判定一个点相对于多边形前向面和后向面的空间位置是许多图形算法的基本任务，例如在判定对象可见性中。每一多边形包含在将空间分为两区域的一个无限平面中。任何一个不在平面上且可看见对象前向面的点称为在平面的前方（或外部），因此该点在对象的外部。任何可看见多边形后向面的点均称为在平面后方（或内部）。位于所有多边形所在平面后方（内部）的点是对象的内点。必须注意，这种内/外分类是与包含多边形的平面联系在一起的，而前面的使用环绕数或奇偶规则的内/外测试则是针对某些二维边界的内部。

平面方程可用于判定空间一点与对象的多边形面片的相对位置关系。如果任意点(x, y, z)不在参数为 A、B、C、D 的平面上，则

$$Ax + By + Cz + D \neq 0$$

因此我们可以按 $Ax + By + Cz + D$ 的符号来判定一个点是否在该面中多边形的后方或前方：

如果$Ax + By + Cz + D < 0$，则点(x, y, z)在平面后方

如果$Ax + By + Cz + D > 0$，则点(x, y, z)在平面前方

这些不等式测试在右手笛卡儿系统中有效，其中参数 A、B、C、D 使用从前往后观察平面时严格按逆时针顺序排列的坐标位置中选出的坐标值计算而得。例如，图3.52中任意一个在着色多边形外部（前方）的点满足不等式 $x - 1 > 0$，而任意内部（后方）的点的 x 坐标小于 1。

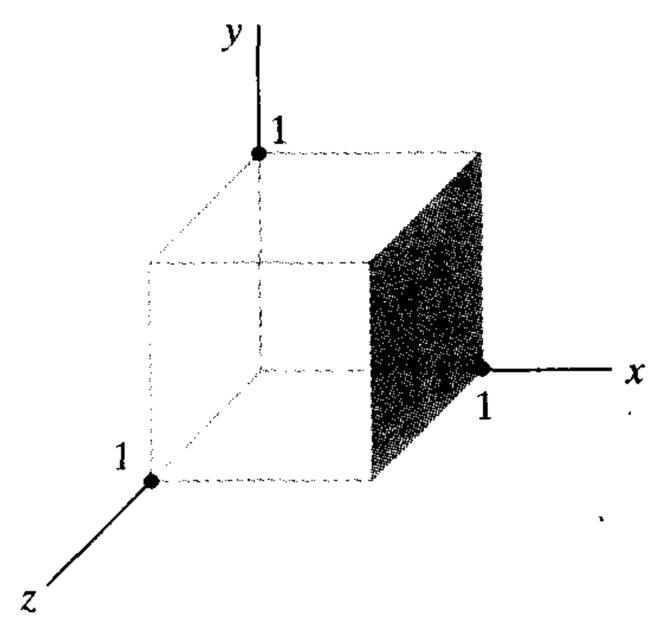


图 3.52 单位立方体的着色多边形的平面方程为 $x-1=0$

多边形表面的空间方向可用其所在平面的**法向量**（normal vector）来描述，如图 3.53 所示。该表面法向量与平面垂直且以（A, B, C）为其笛卡儿坐标分量，其中 A、B 和 C 是用方程（3.62）计算而得的平面系数。法向量从平面的内部指向外部，即从多边形的后方指向前方。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

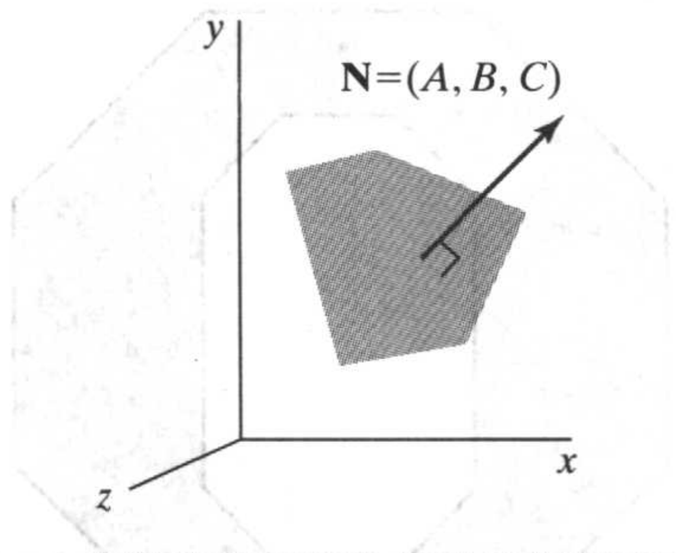


图 3.53 方程 $Ax+By+Cz+D=0$ 所描述平面的法向量 **N** 与该平面垂直并有笛卡儿分量(A, B, C)

作为计算多边形法向量即平面参数的例子，我们选择图3.52中单位立方体着色面的三个顶点。这些顶点按从外向其中心方向观察立方体时的逆时针方向排序。按此顺序选择的顶点坐标用于方程（3.62）获得平面系数：$A=1$、$B=0$、$C=0$、$D=-1$。因此，该平面的法向量是 $\mathbf{N}=(1, 0, 0)$，即 x 轴的正方向。这就是说，法向量从立方体内部指向外部且与平面 $x=1$ 垂直。

法向量的分量也可通过向量叉积计算获得。假定我们有一个凸多边形面片和一个右手坐标系，我们再选择任意三个顶点：$\mathbf{V}_1$、$\mathbf{V}_2$ 和 $\mathbf{V}_3$，满足从对象外部向内观察时的逆时针排序。形成两个向量，一个从 $\mathbf{V}_1$ 到 $\mathbf{V}_2$ 而第二个从 $\mathbf{V}_1$ 到 $\mathbf{V}_3$，按向量叉积计算 **N**：

$$\mathbf{N}=(\mathbf{V}_2-\mathbf{V}_1)\times(\mathbf{V}_3-\mathbf{V}_1) \tag{3.63}$$

这样生成了平面参数 A、B 和 C，接下来将这些值和一个多边形顶点坐标代入平面方程（3.59）可解出 D。使用法向量 **N** 和平面上任一点 **P** 可给出向量形式的平面方程：

$$\mathbf{N}\cdot\mathbf{P}=-D \tag{3.64}$$

对于凸多边形来说，我们也可以使用两个连续的边向量的叉积来获得平面参数。对于凹多边形，我们可以选择这样的三个顶点，使用于叉积计算的两条边的夹角小于180°。否则，我们取叉积的反向量来获得正确的多边形法向量方向。

3.16 OpenGL 多边形填充区函数

描述填充多边形的OpenGL过程与描述点和折线类似，但有一个例外。函数glVertex用来输入多边形的一个顶点坐标，而完整的多边形用从glBegin到glEnd之间的一组顶点来描述。但有另外一个函数可以用来显示具有完全不同格式的矩形。

默认时多边形内部显示为单色，由当前颜色设定确定其颜色。作为选项（下面的内容中叙述），可以用图案填充多边形且显示多边形的边作为内部填充的边界。函数glBegin中指定多边形填充区的变量可使用6个不同的符号常量。这6个基本常量可用来显示单一填充多边形、一组不相连的填充多边形或一组相连的填充多边形。

OpenGL中的填充区必须指定为凸多边形。因此，一个填充多边形的顶点集中至少包含三个顶点，其中无相交边，多边形所有内角均小于180°。单个多边形填充区只用一个顶点集描述，其内部不能包含图3.54所示的洞。那样的图形可用两个重叠的多边形来描述。

我们描述的每一个多边形有两个面：后向面和前向面。在OpenGL中，可以为每个面分别设定填充颜色和其他属性，且在二维和三维观察子程序中要求有后向/前向标志。因此，多边形按从“外部”观察它时的逆时针方向描述。这标识了该多边形的前向面。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

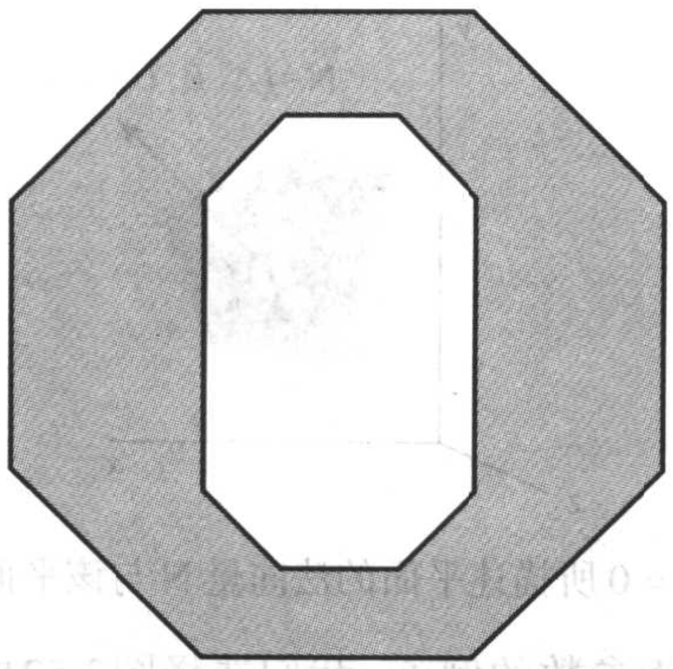


图 3.54 有复杂内部结构的多边形，不能用单个顶点集描述

因为图形显示中经常包含有矩形的填充区，OpenGL 提供了一个特殊的矩形函数，直接在 *xy* 平面中描述顶点。在有些 OpenGL 的实现中，下面的函数比用 glVertex 描述的填充区有更高的效率：

```
glRect* (x1, y1, x2, y2);
```

该矩形的一个角位于坐标位置($x1, y1$)处，而与其相对的一角位于坐标位置($x2, y2$)处。glRect 的后缀码指出坐标数据类型及是否用数组元素来表示坐标。这些编码是 i（整数）、s（短整数）、f（浮点数）、d（双倍长浮点数）和 v（向量）。矩形的边平行于 *xy* 坐标轴。作为一个例子，下面的语句定义了图 3.55 中给出的正方形：

```
glRecti (200, 100, 50, 250);
```

如果将坐标值放在数组中，可用下列语句生成同样的正方形：

```
int vertex1 [ ] = {200, 100};
int vertex2 [ ] = {50, 250};
glRectv (vertex1, vertex2);
```

利用函数 glRect 生成矩形时，多边形的边按顶点序列($x1, y1$)、($x2, y1$)、($x2, y2$)、($x1, y2$)来形成，然后返回到第一顶点($x1, y1$)。在该例子中，我们生成了顺时针次序的顶点集。在许多二维应用中，前向面和后向面的确定是不重要的。但如果确实要将不同的特性赋给矩形的前向面和后向面，那就应该将本例中的两个顶点次序倒过来从而得到逆时针的顶点次序。第4章将讨论另一种可以颠倒前向面和后向面描述的方法。

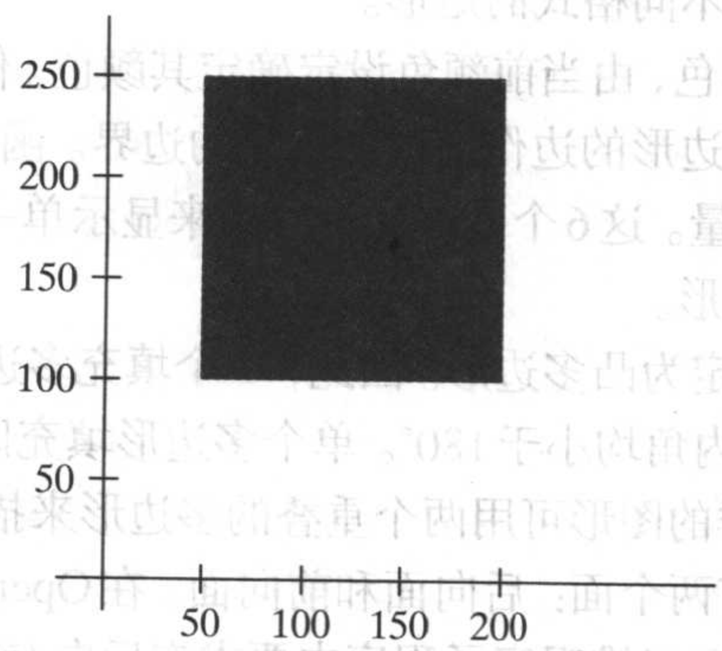


图 3.55 使用 glRect 显示一个矩形填充区

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

另外六个 OpenGL 多边形填充图元中的每一个都用 glBegin 函数中的符号常量及一组 glVertex 命令描述。使用 OpenGL 图元常量 GL_POLYGON 可以显示图 3.56(a)那样的单个多边形。在该例子中，我们假定有六个顶点，标号 p1 到 p6，描述一个逆时针次序的两维多边形顶点位置。每一点用一个数组(*x*, *y*)坐标值表示。

```
glBegin (GL_POLYGON);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
    glVertex2iv (p6);
glEnd ( );
```

一个多边形的顶点集至少包含三个顶点，否则什么也不显示。

如果改变前一例子中顶点集的次序并将图元常量改变成GL_TRIANGLES，就获得图3.56(b)所示的两个分开的三角形填充区。

```
glBegin (GL_TRIANGLES);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p6);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
glEnd ( );
```

此时，前面三个坐标点定义一个三角形的顶点，后面三点定义下一个三角形，依次类推。对于每一个三角形填充区，我们指定逆时针次序的顶点位置。除非重复使用某些顶点，否则该图元常量仅显示不相连的三角形。如果顶点数小于 3，则什么也不显示。而如果指定的顶点数不是 3 的倍数，则最后一个或两个顶点没有用。

再次改变顶点集次序并将图元常量改为 GL_TRIANGLE_STRIP，可显示图 3.56(c)所示的一组连接的三角形。

```
glBegin (GL_TRIANGLE_STRIP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p6);
    glVertex2iv (p3);
    glVertex2iv (p5);
    glVertex2iv (p4);
glEnd ( );
```

假定在 N 个顶点的集合中没有重复的坐标位置，可获得 $N-2$ 个三角形的带。很清楚，必须有 $N \geqslant 3$，否则什么也不显示。在该例子中，$N=6$，得到 4 个三角形。每一后继三角形共享前面定义的三角形的一条边，因此顶点次序的设定必须保证显示的一致性。一个三角形在前面两个顶点的基础上再加一个顶点来定义。因此，最前面三个顶点必须按从前面（外部）观察三角形表面时的逆时针次序列出。随后，顶点表中用于另外三个三角形的三个顶点安排成顺时针次序。这通过按次序 $n=1, n=2,\ldots, n=N-2$ 处理顶点集中的位置 n，并按 n 是否为奇数或偶数确定相应的三顶点集次序来实现。如果 n 是奇数，三角形顶点的多边形列表次序是 n、$n+1$、$n+2$。如果 n 是偶数，则三角形顶点次序为 $n+1$、n、$n+2$。在前面的例子中，第一个三角形（$n=1$）的顶点次序为(p1, p2, p6)。第

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

二个三角形（$n=2$）的顶点次序为(p6, p2, p3)。第三个三角形（$n=3$）为(p6, p3, p5)。多边形表中第四个三角形（$n=4$）的顶点次序为(p5, p3, p4)。

生成一组相连三角形的另一种方法是使用图3.56(d)所示的“扇形”方法，其中有一个顶点被所有的三角形共享。使用图元常量GL_TRIANGLE_FAN和六个顶点的原来次序可以获得这样的设置:

```
glBegin (GL_TRIANGLE_FAN);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
    glVertex2iv (p6);
glEnd ( );
```

N个顶点可获得N–2个三角形，不需要重复使用任何顶点，但必须至少列出三个顶点。另外，必须合适地描述顶点次序，以便正确定义每一个三角形的前向面和后向面。第一个列出的顶点（此时为p1）是扇形中每一三角形共享的顶点。如果我们再按$n=1, n=2,..., n=N-2$来计算多边形和坐标位置，则多边形表中第n个三角形的顶点次序为1、n+1、n+2。因此，三角形1由顶点集(p1, p2, p3)定义；三角形2的顶点次序为(p1, p3, p4)；三角形3的顶点次序为(p1, p4, p5)；而三角形4的顶点集为(p1, p5, p6)。

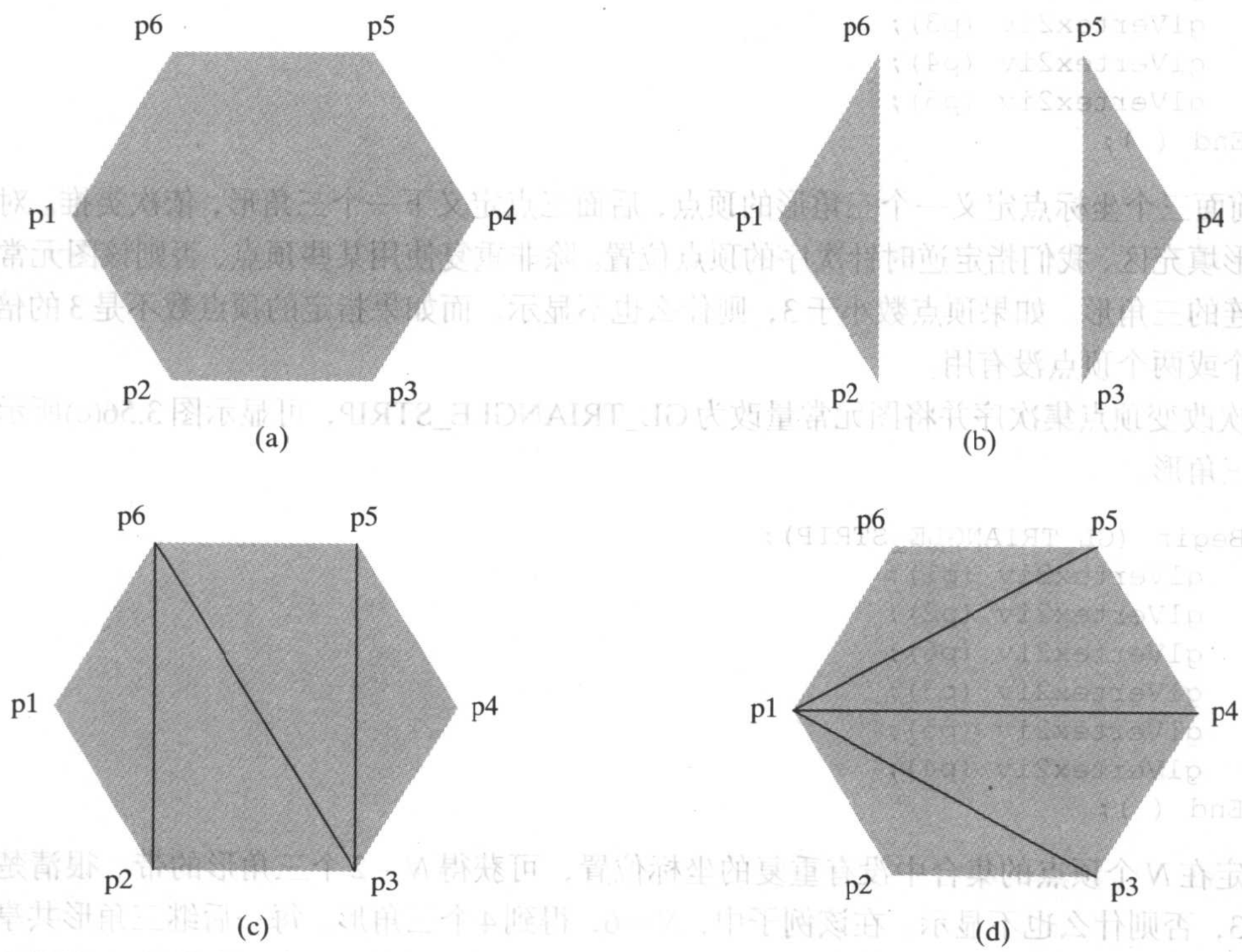


图 3.56 使用六个顶点位置显示多边形填充区域：(a)用图元常量 GL_POLYGON 生成单个凸多边形填充区域；(b)用图元常量 GL_TRIANGLES 生成两个不相连的三角形；(c)用图元常量 GL_TRIANGLE_STRIP 生成四个相连的三角形；(D)用 GL_TRIANGLE_FAN 生成四个相连的三角形

除了三角形和一般多边形的图元函数，OpenGL还可描述两类四边形。用GL_QUADS图元常量和下面两维坐标数组指定的8个顶点，可生成图3.57(a)的显示结果。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glBegin (GL_QUADS);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p3);
    glVertex2iv (p4);
    glVertex2iv (p5);
    glVertex2iv (p6);
    glVertex2iv (p7);
    glVertex2iv (p8);
glEnd ( );
```

前面四个点定义一个四边形的顶点，接下来四个点定义下一个四边形，依次类推。我们为每一个四边形填充区指定逆时针次序的顶点位置。如果没有重复的顶点位置，则显示一组不相连的四边形填充区。使用该图元时至少要列出四个顶点，否则什么也不显示。而如果指定的顶点数不是4的倍数，则多余的顶点将被忽略。

将前面的四边形程序例中的顶点集重新安排，并将图元常量改为GL_QUAD_STRIP，我们得到图3.57(b)中一组相连的四边形。

```
glBegin (GL_QUAD_STRIP);
    glVertex2iv (p1);
    glVertex2iv (p2);
    glVertex2iv (p4);
    glVertex2iv (p3);
    glVertex2iv (p5);
    glVertex2iv (p6);
    glVertex2iv (p8);
    glVertex2iv (p7);
glEnd ( );
```

在先指定两个顶点后，每个四边形用再两个顶点指定，而且我们必须列出能使每一个四边形获得正确的逆时针顶点次序的顶点集。在$N \geq 4$时，N个顶点可生成$N/2-1$个四边形。如果N不是4的倍数，顶点集中多余的坐标位置则不被使用。我们可按$n = 1, n = 2,..., n = N/2-1$对填充多边形和顶点计数。这样，多边形表中第$n$个四边形的顶点次序为$2n-1$、$2n$、$2n+2$、$2n+1$。在本例中，$N = 8$因而有三个四边形组成一个带。因此，第一个四边形（$n = 1$）的顶点次序为(p1, p2, p3, p4)。第二个四边形（$n = 2$）的顶点次序为(p4, p3, p6, p5)。而第三个四边形（$n = 3$）的顶点次序为(p5, p6, p7, p8)。

多数图形软件包使用逼近平面片来显示曲面。这是因为平面方程是线性的，而处理线性方程比二次或其他类曲线方程快得多。因此OpenGL和其他图形软件包提供多边形图元来实施曲面的逼近。对象用多边形网络来建模，而几何和属性信息的数据库按处理多边形面片的目标来建立。在OpenGL中，可用于此目的的图元有三角形带（triangle strip）、三角形扇形（triangle fan）和四边形带（quad strip）。高性能图形系统使用快速多边形硬件绘制，使得显示速度达到每秒形成百万以上的多边形（通常为三角形），包括使用表面纹理和特殊的光照效果。

尽管OpenGL的核心函数库只允许凸多边形，但OpenGL的实用函数库（GLU）提供有关函数处理凹多边形和其他有线性边界的非凸对象。可使用一组GLU多边形细分子程序来将那些形状转换成三角形、三角形网络、三角形扇形和直线段。一旦那些形状被分解，就可以使用OpenGL函数进行处理。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

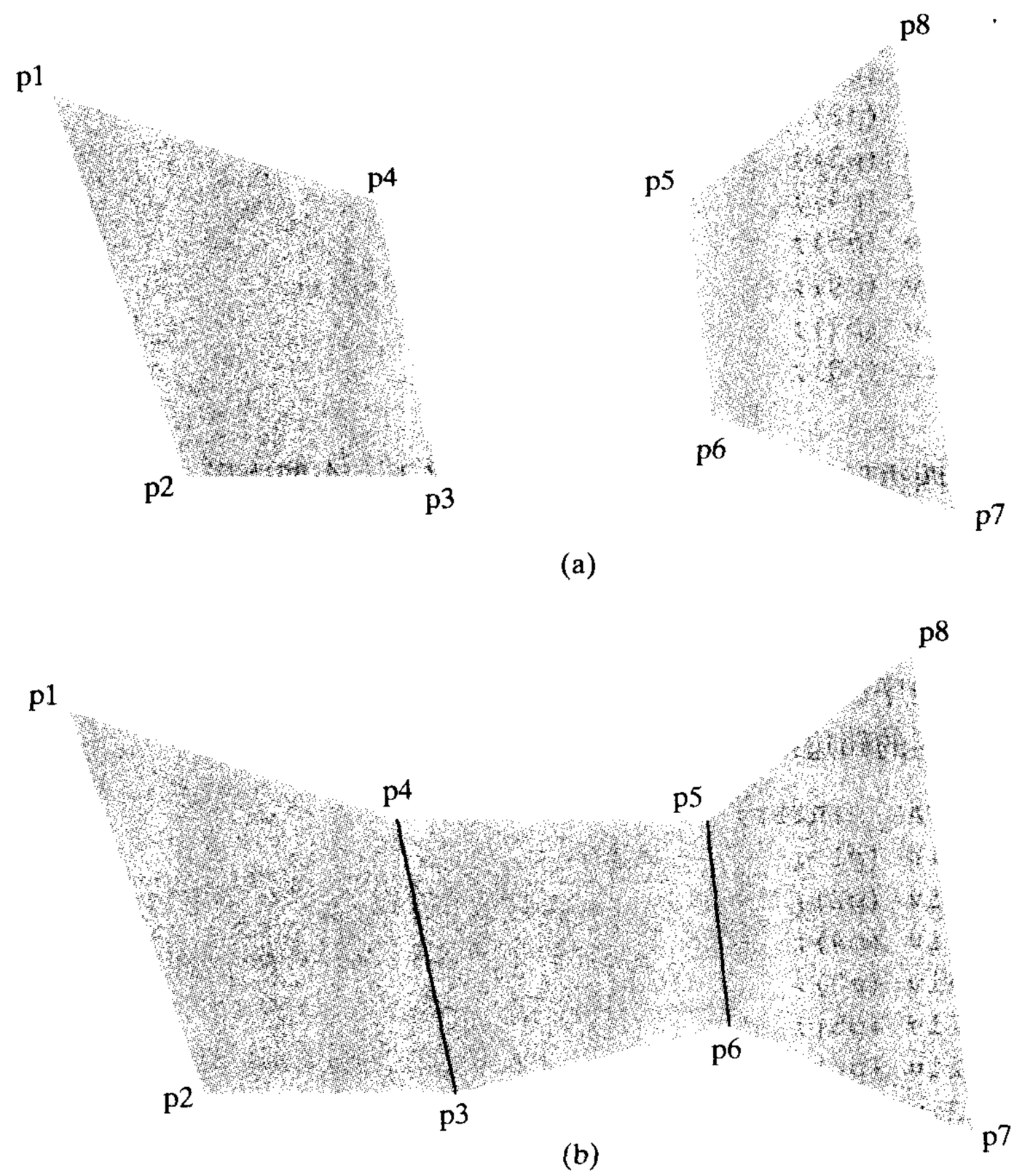


图 3.57 用8个顶点显示四边形填充区：(a)用GL_QUADS生成两个不相连的四边形；(b)用GL_QUAD_STRIP生成三个相连的四边形

3.17 OpenGL 顶点数组

尽管前面给出的例子中只包含少量的坐标位置，但描述包含若干个对象的场景一般会复杂得多。我们先考虑描述一个简单的很基本的对象：图3.58中的单位立方体，为简化后面的讨论而使用了整数坐标。定义顶点坐标的直接方法是用一个双下标数组，例如：

```
GLint points [8][3] ={ {0, 0, 0}, {0, 1, 0}, {1, 0, 0}, {1, 1, 0},
                       {0, 0, 1}, {0, 1, 1}, {1, 0, 1}, {1, 1, 1} };
```

也可以先定义一个三维顶点位置的数据类型，然后给出作为单下标数组元素的每一顶点位置的坐标，例如：

```
typedef GLint vertex3 [3];

 vertex3 pt [8] = { {0, 0, 0}, {0, 1, 0}, {1, 0, 0}, {1, 1, 0},
                    {0, 0, 1}, {0, 1, 1}, {1, 0, 1}, {1, 1, 1} };
```

下面要定义该对象的六个面。为此，分六次调用glBegin (GL_POLYGON)或glBegin (GL_QUADS)。我们必须明确每一个面的顶点顺序符合从立方体外部对其观察时为逆时针次序的要求。在下面的程序段中，我们指定每个立方体面为一个四边形，并且使用一个函数调用将数组下标值传给OpenGL图元子程序。图 3.59 给出了与立方体顶点位置对应的数组 pt 的下标值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

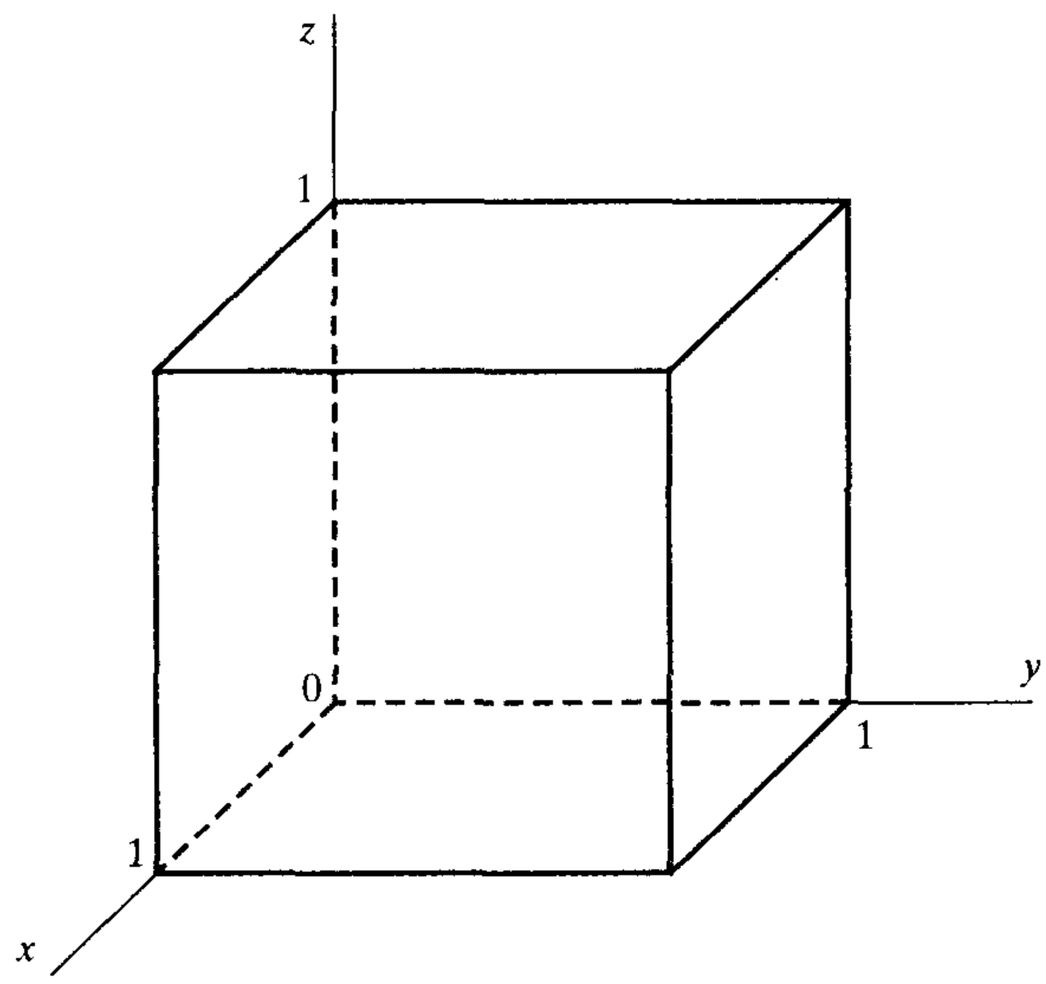


图3.58 边长为1的立方体

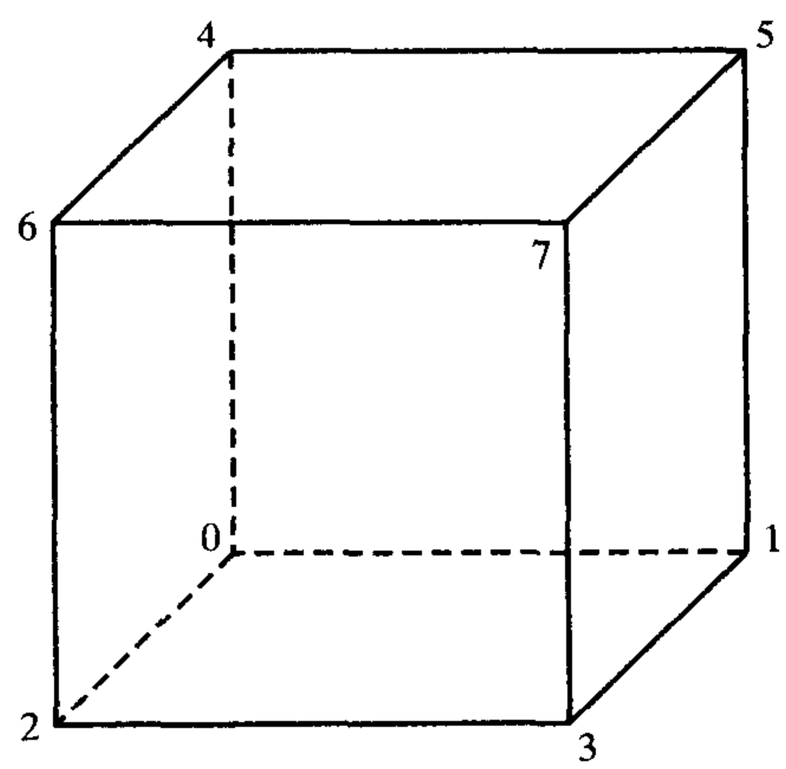


图3.59 与图3.58中立方体顶点对应的数组pt的下标值

```
void quad (Glint n1, Glint n2, Glint n3, Glint n4)
{
   glBegin (GL_QUADS);
      glVertex3iv (pt [n1]);
      glVertex3iv (pt [n2]);
      glVertex3iv (pt [n3]);
      glVertex3iv (pt [n4]);
   glEnd ( );
}

void cube ( )
{
   quad (6, 2, 3, 7);
   quad (5, 1, 0, 4);
   quad (7, 3, 1, 5);
   quad (4, 0, 2, 6);
   quad (2, 0, 1, 3);
   quad (7, 5, 4, 6);
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

这样，指定一个面要用六个OpenGL函数，共有六个面需要指定。在加入颜色描述和其他参数后，显示立方体的程序很容易包含一百个以上的OpenGL函数调用。而有许多复杂对象的场景会需要更多的函数调用。

从上面的例子中可以看出，复杂的场景描述需要使用几百或几千个坐标描述。另外还必须为各个对象建立各种属性和观察参数。因此，对象和场景描述要使用大量的函数调用，这对系统资源提出了要求并减慢了图形程序的执行。复杂显示进一步的问题是对象表面（如图3.58中的立方体）通常有共享顶点。使用已讨论过的方法，这些共享顶点需要多次指定。

为了简化这些问题，OpenGL提供了一种机制来减少处理坐标信息的函数调用数量。使用**顶点数组**（vertex array），可以利用很少的函数调用来安排场景的描述信息。步骤如下：

1. 引用函数glEnableClientState (GL_VERTEX_ARRAY)激活OpenGL的顶点数组特性。
2. 使用函数glVertexPointer指定顶点坐标的位置和数据格式。
3. 使用子程序如glDrawElements显示场景，该子程序可处理多个图元而仅需少量的函数调用。

使用前面定义的pt数组，实现下列程序示例中的三步。

```
glEnableClientState (GL_VERTEX_ARRAY);
glVertexPointer (3, GL_INT, 0, pt);

GLubyte vertIndex [ ] = (6, 2, 3, 7, 5, 1, 0, 4, 7, 3, 1, 5, 4, 0, 2, 6,
     2, 0, 1, 3, 7, 5, 4, 6);

glDrawElements (GL_QUADS, 24, GL_UNSIGNED_BYTE, vertIndex);
```

第一条命令glEnableClientState (GL_VERTEX_ARRAY)，激活了客户/服务器系统中客户端的能力（此时是顶点数组）。因为客户端（运行主程序的机器）保留图形的数据，顶点数组必须在那里。如第2章所指出的，服务器（如工作站）发出命令并显示图形。当然，单个计算机既是客户端又是服务器。OpenGL的顶点数组特性用下列命令来使其无效。

```
glDisableClientState (GL_VERTEX_ARRAY);
```

接下来为函数glVertexPointer提供对象顶点坐标的位置和格式。glVertexPointer的第一个参数在此例中为3，指出每一个顶点描述中的坐标数目。顶点坐标的数据类型用函数中第二个参数OpenGL符号常量来指定。此例中数据类型为GL_INT。另外的数据类型用符号常量GL_BYTE、GL_SHORT、GL_FLOAT和GL_DOUBLE来指定。第三个参数用来给出连续顶点之间的字节位移。这一参数的目的是允许多种类型的数据（如坐标和颜色）捆绑在同一个数组内。由于我们仅给出坐标数据，所以为这个位置参数赋值为0。函数glVertexPointer的最后一个参数指向包含坐标值的顶点数组。

立方体顶点的所有索引存放在数组vertIndex中。其中每一个索引是对应于该顶点值的数组pt的下标。该索引表被当做函数glDrawElements的最后一个参数，并由显示立方体的四边界表面的第一个参数——图元GL_QUADS使用。第二个参数指定数组vertIndex中的元素数量。因为一个四边形有4个顶点，所以我们指定24，glDrawElements函数每次取出4个顶点来显示一个立方体面，直到24个顶点用完。这样，使用一个函数调用完成了整个立方体所有面的显示。glDrawElements函数的第三个参数给出索引值的类型。因为此时所用的索引是小整数，我们指定其为GL_UNSIGNED_BYTE类型。另外两种可用的索引类型是GL_UNSIGNED_SHORT和GL_UNSIGNED_INT。

也可将其他的信息与坐标值一起放进顶点数组中用于场景描述。我们可以在数组（由函数glDrawElements引用）中指定对象的颜色值和其他属性。为了提高效率，可以交替使用各种数组。下一章将讨论实现这些属性的方法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3.18　像素阵列图元

除了线段、多边形、圆和其他图元之外，图形软件包常提供一些子程序用于显示由矩形的彩色阵列定义的各种形状。矩形的网格图案可通过数字化（扫描）一张照片或其他图形来获得，也可以使用图形程序来生成。阵列中每一颜色值映射到一个或多个屏幕像素位置。如第2章所指出的，一个彩色像素阵列称为一个像素图（pixmap）。

像素阵列的参数包括指向颜色矩阵的指针、矩阵的大小及其将要影响的屏幕区域。图 3.60 给出了用像素阵列映射到屏幕区域的例子。

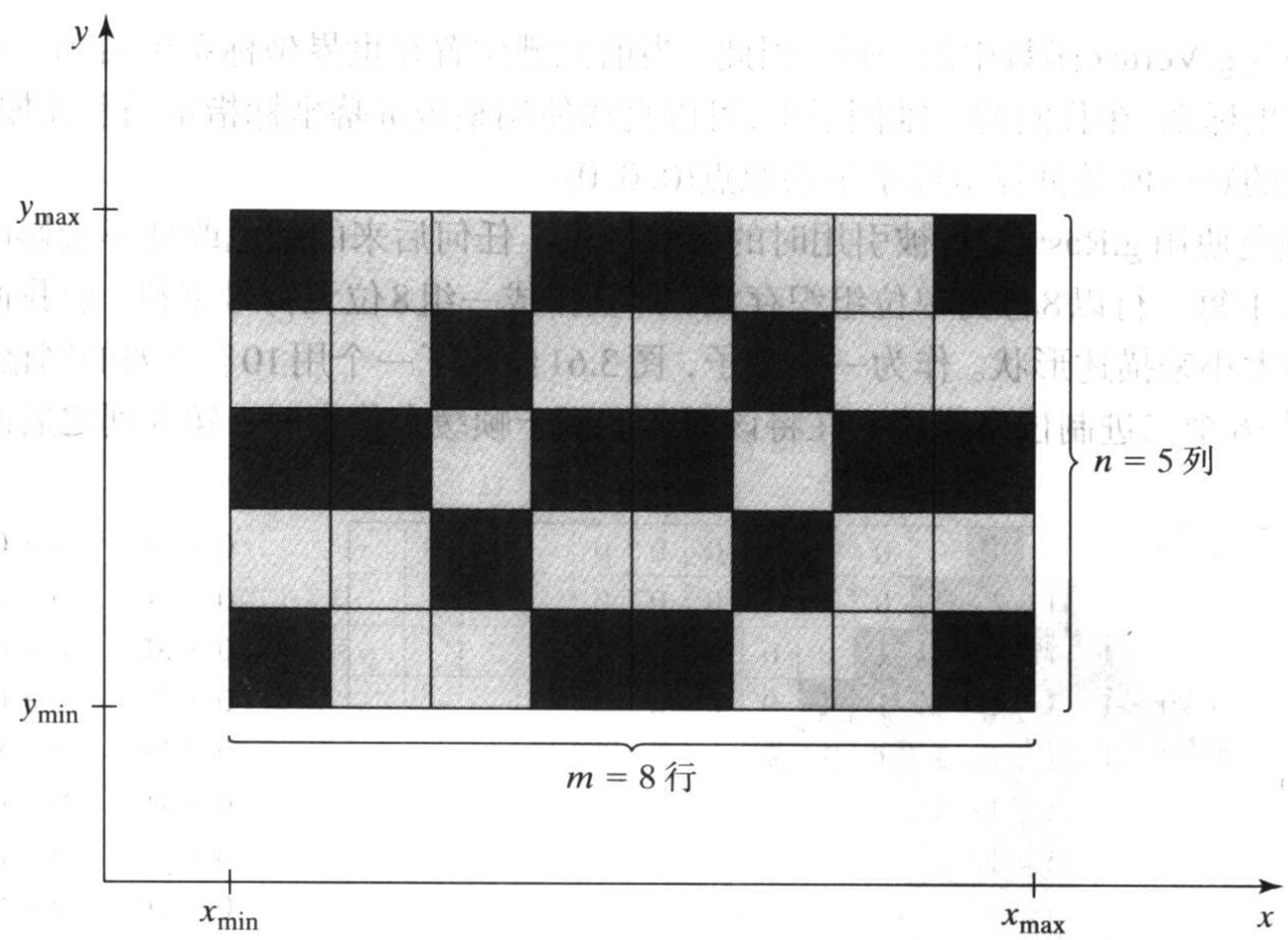


图 3.60　将 $n \times m$ 的彩色阵列映射到屏幕坐标系中的一个区域

实现像素阵列的另一种方法是为矩阵中每一元素赋值为 0 或 1。此时，阵列简化成位图（bitmap），有时也称为掩模（mask），它指出一个像素是否被赋予预定颜色。

3.19　OpenGL 像素阵列函数

OpenGL 中有两个函数可用于定义矩形阵列的形状或图案。一个是位图，另一个是像素图。OpenGL 也提供若干的函数用于存储、复制及管理像素值阵列。

3.19.1　OpenGL 位图函数

下面的函数定义了一个二值的阵列：

```
glBitmap (width, height, x0, y0, xOffset, yOffset, bitShape);
```

函数中的参数 width 和 height 分别给出阵列 bitShape 的列数和行数。bitShape 的每一元素赋值为 0 或 1。值为 1 表示对应像素用前面设定的颜色显示；否则，对应像素不受该位图影响。（作为一个选项，可使用 1 表示将指定颜色与存储在刷新缓存中对应位置的颜色值相结合。）参数 x0 和 y0 定义了矩形阵列“原点”的位置。原点位置指定为 bitShape 的左下角，而 x0 和 y0 可正可负。另外，需要指定帧

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

缓存中应用图案的位置。该位置称为**当前光栅位置**（current raster position），而位图在将原点置于当前光栅位置后显示。赋给xOffset和yOffset的值用做位图显示后更新帧缓存当前光栅位置的坐标位移。

x0、y0、xOffset、yOffset和当前光栅位置的坐标值使用浮点数格式。位图当然用整数的像素位置。但浮点数的坐标允许将一组位图以任意的间隔安排，这在某些应用如用位图字符形成字符串中很有用。

可以使用下面的子程序来设定当前光栅位置：

```
glRasterPos* ( )
```

参数和后缀码与glVertex函数中的一样。因此，当前光栅位置在世界坐标系中给出，由观察变换将其变换到屏幕坐标系。在我们的二维例子中，可以直接使用整数屏幕坐标指定当前光栅位置的坐标。当前光栅位置的默认值是世界坐标系中的原点(0, 0, 0)。

位图的颜色使用glRasterPos被引用时的有效颜色。任何后来的颜色改变不会影响该位图。

矩形位阵中每一行以8位为单位组织存放，即安排成一组8位无符号字符。但我们可以使用任何方便的网格大小来描述形状。作为一个例子，图3.61给出了一个用10行9列的网格定义位阵列，这时每行使用16个二进制位来描述。在将该图案应用于帧缓存像素时，第9列之后的所有位值均被忽略。

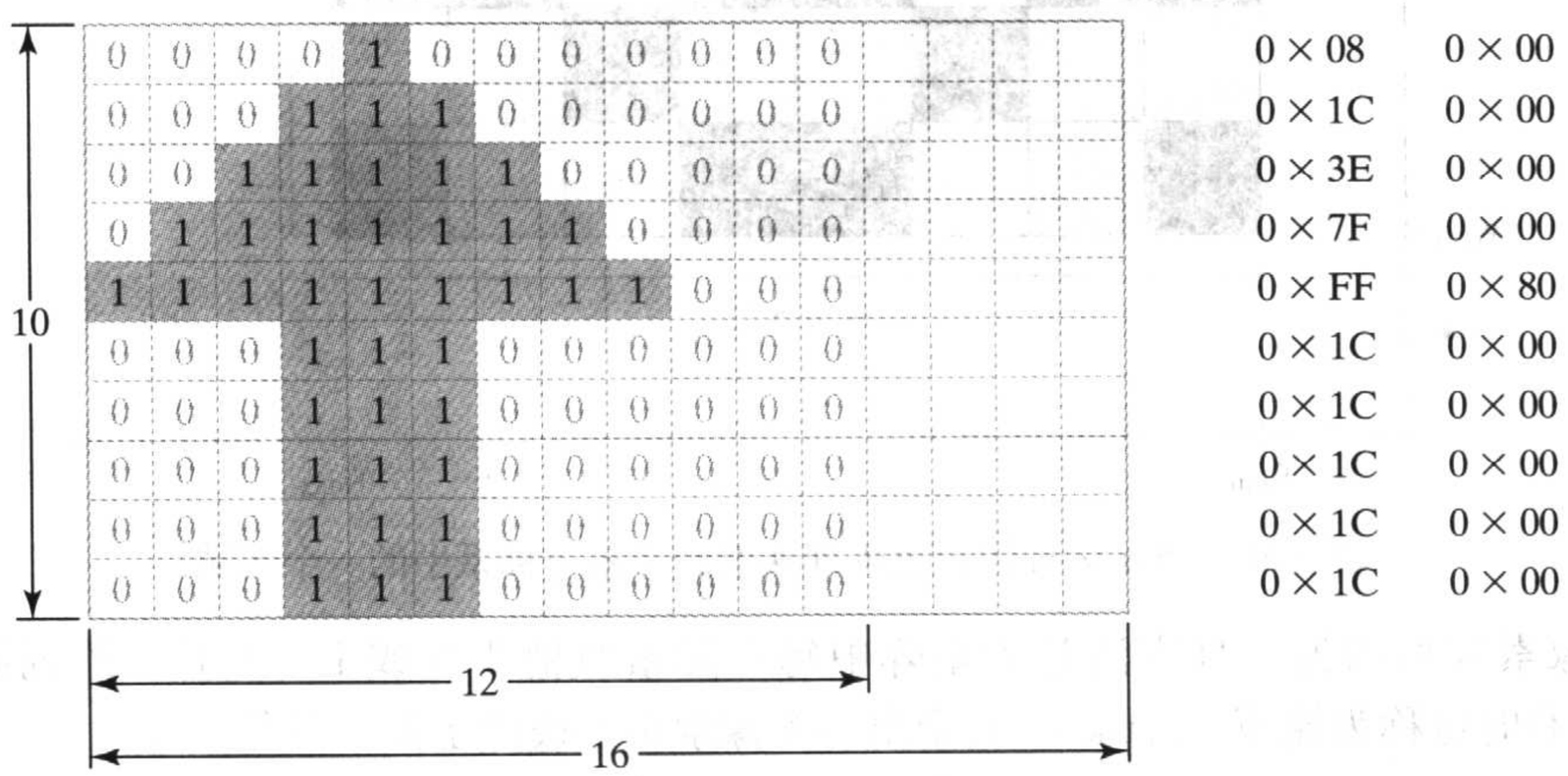


图3.61 用10行和9列阵列指定的一个位图存在每行16位的10行8位块中

使用下面的程序段可将图3.61中的位图案应用于帧缓存中：

```
GLubyte bitShape [20] = {
    0x1c, 0x00, 0x1c, 0x00, 0x1c, 0x00, 0x1c, 0x00, 0x1c, 0x00,
    0xff, 0x80, 0x7f, 0x00, 0x3e, 0x00, 0x1c, 0x00, 0x08, 0x00} ;

glPixelStorei (GL_UNPACK_ALIGNMENT, 1);  // Set pixel storage mode.

glRasterPos2i (30, 40);
glBitmap (9, 10, 0.0, 0.0, 20.0, 15.0, bitShape);
```

bitShape的阵列值从矩形网格的底部开始逐行指定。接着使用OpenGL函数glPixelStorei设定位图的存储模式。该函数中使用参数值1表明数据值用字节边界对齐。glRasterPos用来设定当前光栅位置为(30, 40)。最后，函数glBitmap指定位阵列在阵列bitShape中给出，且该阵列有9列10行。这个阵列的原点坐标在(0.0, 0.0)，即在网格的左下角。我们给出坐标位移为(20.0, 15.0)，尽管在本例中该位移没有用。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3.19.2 OpenGL 像素图函数

函数：

```
glDrawPixels (width, height, dataFormat, dataType, pixMap);
```

将用彩色阵列定义的图案应用到一块帧缓存的像素位置。其中的 width 和 height 也分别给出像素位图的列数和行数（阵列 pixMap）。参数 dataFormat 用一个 OpenGL 常量赋值，指出如何为阵列指定值。例如，使用常量 GL_BLUE 可指定所有像素都使用蓝色，使用常量 GL_BGR 可按蓝、绿、红次序指定颜色分量。还可能有一些其他的颜色指定方法，我们在下一章将详细关注颜色选择的方法。参数 dataType 设定为 OpenGL 常量 GL_BYTE、GL_INT 或 GL_FLOAT，以指定阵列中颜色的数据类型。该颜色阵列的左下角映射到由 glRasterPos 设定的当前光栅位置。作为一个例子，下面的语句显示一个 128 × 128 的 RGB 彩色阵列定义的像素图：

```
glDrawPixels (128, 128, GL_RGB, GL_UNSIGNED_BYTE, colorShape);
```

由于 OpenGL 提供了若干个缓存，将某缓存选为 glDrawPixels 子程序的目标即可将一个阵列送进该缓存。有的缓存存放颜色值，而有的存放另外的像素数据。例如，深度缓存（depth buffer）用来存放对象离开观察位置的距离，而模板缓存（stencil buffer）用来存放场景的边界图案。glDrawPixels 函数中的 dataFormat 参数设定为 GL_DEPTH_COMPONENT 或 GL_STENCIL_INDEX 就可在两个缓存中选定一个。我们需要使用深度值或模板信息设定像素阵列。后面几章将详细考察这些缓存。

OpenGL 有 4 个颜色缓存（color buffer）用于屏幕刷新。两个颜色缓存用于立体显示中的左、右两个场景。对于立体显示缓存中的每一个，有用于动画显示的前－后一对双缓存。在 OpenGL 的特殊实现中，立体显示或双缓存之一、或是二者都得不到支持。如果立体显示和双缓存都不支持，则仅有单一的刷新缓存用做**前－左颜色缓存**（front-left color buffer）。这是在双缓存无效时的默认缓存。如果双缓存有效，默认是后－左和后－右缓存或仅仅是后－左缓存，取决于立体观察的当前状态。同样，还支持一些用户定义的辅助颜色缓存用于任何非刷新用途，如保存图片用于以后复制到刷新缓存去显示。

使用下面的命令可选择一个颜色的辅助或混合缓存来存储像素图：

```
glDrawBuffer (buffer);
```

多种 OpenGL 符号常量可赋给参数 buffer，以指定一个或多个“绘图”缓存。例如，可用 GL_FRONT_LEFT、GL_FRONT_RIGHT、GL_BACK_LEFT 或 GL_BACK_RIGHT 来选定单一的缓存。使用 GL_FRONT 来选择两个前缓存，而用 GL_BACK 选择两个后缓存。这假定立体显示已有效。否则，前面两个符号常量仅指定单一缓存。类似地，我们可以用 GL_LEFT 或 GL_RIGHT 指定一对左缓存或一对右缓存，并且可以使用 GL_FRONT_AND_BACK 来选择所有可用的颜色缓存。辅助缓存通过常量 GL_AUXk 来选择，其中 k 是 0 到 3 的一个整数，某些 OpenGL 实现中有多于 4 个辅助缓存。

3.19.3 OpenGL 光栅操作

除了将像素阵列存入缓存，我们可以从缓存中取出一块值或将一块值复制到另一缓存区域。可以对像素阵列执行各种其他操作。一般情况下，术语**光栅操作**（raster operation）用于描述以某种方式处理一个像素阵列的任何功能。将一个像素阵列的值从一个位置移到另一位置的光栅操作也称为像素值的 **bitblt 移动**（bit-block transfer），尤其是在该功能由硬件实现时。在多层次的系统中，术语 pixblt 用于块移动。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

使用下列函数可在指定缓存中选择一个矩形块的像素值：

```
glReadPixels (xmin, ymin, width, height, dataFormat, dataType, array);
```

要提取的矩形块的左下角是屏幕坐标位置(xmin, ymin)。参数width、height、dataFormat和dataType与glDrawPixels子程序中的相同。存入参数array中的数据类型依赖于选择的缓存。我们可通过给参数dataFormat赋值GL_DEPTH_COMPONENT或GL_STENCIL_INDEX来选择深度缓存或模板缓存。

使用函数：

```
glReadBuffer (buffer);
```

可以为glReadPixels子程序选择颜色或辅助缓存的特殊组合。指定一个或多个缓存的符号常量与glDrawBuffer子程序中的一样，但不能选择所有四个颜色缓存。默认选择是由立体观察状态所确定的前左－前右缓存组合或仅仅是前左缓存。

使用下面的函数可将一块像素数据从OpenGL缓存的一个位置复制到另一个位置：

```
glCopyPixels (xmin, ymin, width, height, pixelValues);
```

块的左下角是屏幕坐标位置(xmin, ymin)，参数width和height被赋以正整数，分别指出要复制的列数和行数。参数pixelValues被赋以GL_COLOR、GL_DEPTH或GL_STENCIL来指定要复制的数据种类：颜色值、深度值或模板值。一块像素值从源缓存（source buffer）复制到目标缓存（destination buffer），其左下角映射到当前光栅位置。源缓存用glReadBuffer命令选择，而目标缓存用glDrawBuffer命令选择。提供和接受复制的两个区域都必须在屏幕坐标的边界内。

为了实现用glDrawPixels或glCopyPixels将一块像素值放入缓存的不同效果，我们可以使用各种方式将取出的值与缓存原来的值进行组合。例如，可使用与（and）、或（or）和异或（exclusive or）等逻辑操作来组合两个块的像素值。在OpenGL中，可以使用下面的函数来选择按位逻辑操作，以组合取来的和目标的像素颜色值：

```
glEnable (GL_COLOR_LOGIC_OP);
glLogicOp (logicOp);
```

参数logicOp可被赋以多种符号常量，包括GL_AND、GL_OR和GL_XOR。另外，取来的和目标的位值可以颠倒（即0和1互换）。使用常量GL_COPY_INVERTED可将取来的颜色位值颠倒后再取代目标值。使用GL_INVERT则仅仅颠倒了目标位值而不使用取来的值。各种颠倒操作还可与逻辑与、或以及异或操作进行组合。其他操作包括将目标位清0（GL_CLEAR），或设定所有的目标位为1（GL_SET）。glLogicOp子程序的默认值是GL_COPY，简单地用取来的值去替代目标值。

还有一些OpenGL子程序用来管理由glDrawPixels、glReadPixels和glCopyPixels等函数处理的像素阵列。例如，glPixelTransfer和glPixelMap子程序可用来移动或调整颜色值、深度值或模板值。我们在以后几章中讲解计算机图形软件包的其他方面时再讨论像素操作。

3.20 字符图元

图形显示中常包括文字信息，如图表上的标记、大楼或汽车上的牌号以及模拟和可视化应用中的标识信息。多数图形软件包中都有生成字符图元的子程序。有些系统提供庞大的字符函数集，而其他的系统则仅提供对字符生成的有限支持。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

字母、数字和其他字符可以显示成不同的大小和风格。一组字符的完整设计风格称为**字样**（typeface）。目前，有上百种字样可供计算机的应用程序使用。常用的字样有Courier、Helvetica、New York、Palatino 和 Zapf Chancery 等。早先，术语**字体**（font）指的是一组按照特定尺寸和格式的模板字符式样。例如 10 磅 Courier 斜体，或 12 磅 Palatino 黑体。14 磅字体的字符高度约为 0.5 厘米。换句话说，72 磅大约等于 2.54 厘米（1 英寸）。现在术语字体和字样经常互用，因为印刷过程不再使用金属模板。

字体可分成两大类：有衬线（serif）和无衬线（sans serif）。有衬线字体在字符主笔划末端带有细线或是笔划加重，而无衬线字体则没有加重。有衬线字体的可读性较好，即在正文的较长段落中容易阅读。另一方面，无衬线字体的单个字符易被识别。由于这个原因，无衬线字体被认为是字迹清楚的。由于可以很快地识别无衬线字符，因而这种字体特别适用于标识和短标题。

字体也按是否为单一宽度（monospace）或比例宽度（proportional）而进行分类。单一宽度字体中所有字符有同样的宽度。比例宽度字体中有多种字符宽度。

存储的计算机字体有两种不同的表示方法。一种表示某种字体字符形状的简单办法是使用矩形网格图案。这样的字符组称为**位图字体**（bitmap font，或**位图化的字体**）。位图化的字符集有时也称为**光栅字体**（raster font）。另一种更灵活的方法是用直线和曲线段来描述字符形状，例如在PostScript中的处理，这种字符组称为**轮廓字体**（outline font）或**笔划字体**（stroke font）。图 3.62 给出了两种字符表示方法。当把图 3.62(a)中的图案复制到帧缓存的某个区域时，值为 1 的位确定监视器上对应的哪一个像素位置将要用指定颜色显示。为了显示图3.62(b)中的字符形状，字符轮廓的内部按填充区处理。

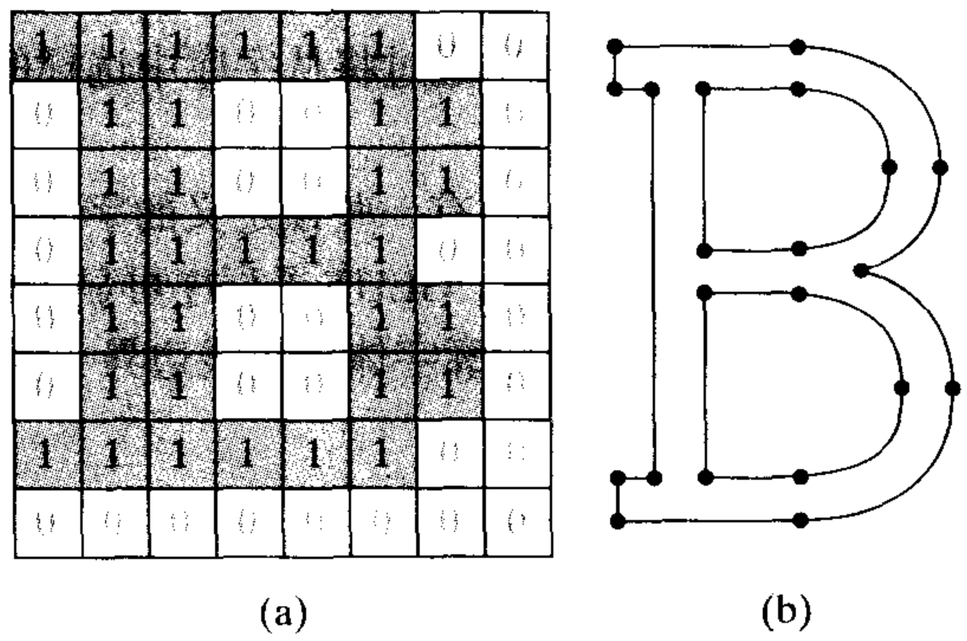


图 3.62　字符 B 表示为：(a)8 × 8 二值点阵图案；(b)用直线和曲线段定义轮廓形状

位图字体的定义和显示最简单，仅需将字符网格映射到帧缓存位置。但是，因为每种（尺寸和格式的）变化都必须存储在字形的高速缓存中，所以位图字体通常需要更多的存储空间。尽管有可能从一组点阵字符生成不同尺寸以及类似粗体、斜体等其他变体的字体，但通常结果并不是很好。我们只能以像素大小的整数倍方式增加或减少字符位图的尺寸。两倍尺寸的字符，需要位图中的四倍像素，但这增加了边缘的粗糙表现。

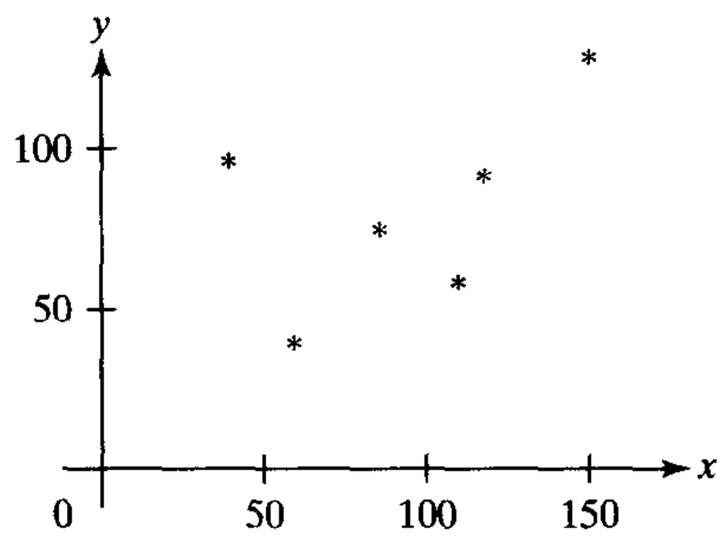
与位图字体相比，轮廓字体在增加大小时其字符形状不会变形。轮廓字体需要较少的存储空间，因为每种变体并不需要各自的字形缓存。通过控制字符轮廓的曲线定义，可以产生粗体、斜体或不同尺寸的字体。但这需要更多的时间来处理轮廓字体，因为必须将它们扫描转换到帧缓存中。

可用于字符显示的函数有许多个。有的图形软件包提供的函数可接受任意的字符串及相应的帧缓存起始位置。另一类函数仅在选定的一处或几处显示字符串。由于字符子程序对为网状布局或离散数据集显示给出标记很有用，因此该子程序显示的字符有时称为**标记符号**（marker symbol）或

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

多点标记（polymarker），这与折线图元相似。另外，标准字符、点、圆和十字等常用做标记符号。图 3.63 给出了用星号作为标记的离散点集图。



x	y
41	94
59	43
85	74
110	59
121	89
149	122

图 3.63 一组数值的多点标记图

和其他图元一样，字符的几何描述在世界坐标系中给出，该信息由观察变换映射到屏幕坐标系。位图字符使用矩形网格的二进制值及网格参考位置来描述。该位置随后被映射到帧缓存中的指定位置。轮廓字符由一组用曲线或线段连接的坐标位置和参考位置来定义，该参考位置随后也映射到给定的帧缓存位置。参考位置可用于一个字符或一个字符串。一般情况下，字符子程序可生成二维或三维字符显示。

3.21 OpenGL 字符函数

OpenGL基本库仅为显示单个字符和文字串提供了基本的支持。我们可以定义图3.61中的位图字符，并将一个位图集作为字库存储。一个文字串通过将从字库中选择的位图序列映射到帧缓存的相邻位置来显示。

但是，OpenGL 实用函数工具包（GLUT）中有一些预定义的字库。因此我们不用创建自己的位图字型库，除非需要显示 GLUT 中没有的字体。GLUT 子程序可显示位图和轮廓字体。GLUT 位图字体由 OpenGL 的 glBitmap 函数来绘制，而轮廓字体由折线边界（GL_LINE_STRIP）生成。

使用下面的函数可显示 GLUT 位图字符：

```
glutBitmapCharacter (font, character);
```

这里参数 font 用 GLUT 符号常量赋值，用来指定一特定字型集，参数 character 赋以 ASCII 编码或其他要显示的字符。这样，要显示大写的“A”，可以使用 ASCII 编码 65 或指定为‘A’。同样，编码 66 与‘B’等效，编码 97 与小写字符‘a’等效，编码 98 与‘b’等效，依次类推。固定宽度或比例间隔字体都可以使用。可用 GLUT_BITMAP_8_BY_13 或 GLUT_BITMAP_9_BY_15 来选择一种固定宽度字体并确定其参数。也可用 GLUT_BITMAP_TIMES_ROMAN_10 或 GLUT_BITMAP_HELVETICA_10 来选择 10 磅的比例间隔字体。12 磅的 Times-Roman 字体和 12 磅及 18 磅的 Helvetica 字体都可以选择。

利用 glutBitmapCharacter 显示的字符以当前光栅位置作为其位图原点（左下角）。在字符位图装入刷新缓存后，当前光栅位置的 x 坐标获得一个字符宽度的增量。例如，我们可以使用下列程序显示一个包括 36 个位图字符的文字串。

```
glRasterPosition2i (x, y);
for (k = 0; k < 36; k++)
    glutBitmapCharacter (GLUT_BITMAP_9_BY_15, text [k]);
```

该字符串用执行 glutBitmapCharacter 函数前指定的颜色来显示。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

使用下面的函数可以显示一个轮廓字符。

```
glutStrokeCharacter (font, character);
```

在这个函数中，我们可为参数font赋值GLUT_STROKE_ROMAN来显示比例空间的字体，或赋值GLUT_STROKE_MONO_ROMAN来显示常量间隔的字体。字符的大小和位置通过在执行glutStrokeCharacter子程序前指定变换操作（参见第5章）来控制。每个字符显示后，自动实施坐标位移，从而使下一字符在当前字符的右边显示。轮廓字符生成的文字串是二维或三维场景的一部分，因为它们用线段构成。这样，它们可从各种方向来观察，并在对它们进行缩放时不会变形，还可以通过其他方式对它们进行变换。但与位图字符相比，绘制速度比较慢。

3.22 图形分割

有的图形软件包中提供了子程序，描述由多个命名部分组合而成的图形并管理每一部分。使用这些函数可以创建、编辑、删除或移动图形的一个组成部分。我们也可使用图形软件的这个功能来进行层次式建模（参见第14章），其中一个对象可以用包括一定层次的子对象的树结构方式来描述。

图形子部分的名称有多种说法。有些图形软件包称它们为结构（structure），另一些则称为段（segment）或对象（object）。同样，在不同的图形软件包中允许的对子部分的操作也不相同。例如，建模软件包提供很多描述和管理图形元素的操作。另一方面，在任意图形库中，我们总可以在C++等高级语言中使用过程元素来构造和管理图形的组成部分。

3.23 OpenGL显示列表

把对象描述成一个命名的语句序列（或任何其他的命令集）并存储起来既方便又高效。在OpenGL中使用称为**显示表**（display list）的结构可以做到这一点。一旦建立了显示表，就可以用不同的显示操作来多次引用该表。在网格中，描述图形的显示表存放在服务器中，以避免每次显示场景时都要传送表中的命令。我们可以为以后的执行来建立并存储显示表，或指定表中的命令立即执行。显示表对层次式建模特别有用，因为一个复杂的对象可以用一组简单的对象来描述。

3.23.1 创建和命名OpenGL显示表

使用glNewList/glEndList函数对包围一组OpenGL命令就可形成显示表。例如：

```
glNewList (listID, listMode);
    .
    .
    .
glEndList ( );
```

该结构用赋予参数listID的正整数作为表名来形成一个显示表。参数listMode可赋以OpenGL符号常量GL_COMPILE或GL_COMPILE_AND_EXECUTE之一。如果希望为以后执行而存储该表，则使用GL_COMPILE。否则，放入表中的命令立即执行，但仍然可以在以后再执行它。

显示表创建后，立即对包含如坐标位置和颜色分量等参数的表示进行赋值计算，从而使表中仅存储参数的值。对这些参数的任何后继修改都不起作用。因为不能修改显示表的值，所以在显示表中不能包含如顶点表指针等OpenGL命令。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们可以创建任意多的显示表并通过调用一个标识来执行特定的显示表。一个显示表还可以嵌套在另一个显示表内。但如果一个显示表被赋予一个已经使用的标识，则它取代原来的显示表内容。因此，为了避免因故重用标识而造成显示表的丢失，可以让 OpenGL 生成一个标识：

```
listID = glGenLists (1);
```

该语句将一个未使用的正整数标识赋给变量listID。如果将glGenLists中的变量由1改成另外一个正整数，则得到一个未使用的显示表标识段。例如，假如引用glGenLists (6)，则保留6个连续正整数并将其中第一个赋给变量listID。如果有错或系统不能产生所要数量的连续整数，则返回0。因此，在使用从 glGenLists 子程序获得的标识之前要先检查它是否为0。

尽管使用glGenList可生成未使用的标识，我们还是可以单独向系统查询指定整数值是否已用做表名。实现该功能的函数是

```
glIsList (listID);
```

如果返回值GL_TRUE，则listID中的值已经用做某显示表的名字。如果该整数尚未被使用，glIsList函数则返回值 GL_FALSE。

3.23.2 执行 OpenGL 显示表

采用下面的语句可执行一个显示表：

```
glCallList (listID);
```

下面的程序段用于创建并执行一个显示表。我们先在 *xy* 平面上建立以(200, 200)为中心坐标、半径为150的圆周上六个等距顶点描述的规则六边形的显示表。然而调用glCallList来显示该六边形。

```
const double TWO_PI = 6.2831853;

GLuint regHex;

GLdouble theta;
GLint x, y, k;

/*  Set up a display list for a regular hexagon.
 *  Vertices for the hexagon are six equally spaced
 *  points around the circumference of a circle.
 */
regHex = glGenLists (1);  //  Get an identifier for the display list.
glNewList (regHex, GL_COMPILE);
   glBegin (GL_POLYGON);
      for (k = 0; k < 6; k++) {
         theta = TWO_PI * k / 6.0;
         x = 200 + 150 * cos (theta);
         y = 200 + 150 * sin (theta);
         glVertex2i (x, y);
      }
   glEnd ( );
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glEndList ( );

glCallList (regHex);
```

使用下列两条语句可以执行多个显示表：

```
glListBase (offsetValue);
glCallLists (nLists, arrayDataType, listIDArray);
```

要执行的显示表数量赋给参数nLists，而参数listIDArray是显示表标识的数组。一般而言，listIDArray可包含任意多的元素，而无效的标识会被忽略。同样，listIDArray中的元素可指定为多种数据格式，而参数arrayDataType用来指出数据类型，例如GL_BYTE、GL_INT、GL_FLOAT、GL_3_BYTES或GL_4_BYTES。显示表标识通过将listIDArray中一个元素的值与offsetValue的整数值相加而得。offsetValue的默认值为0。

这种指定要执行的一串显示表的机制使我们能建立一组相关的显示表，其标识用符号名或编码形成。一个典型的例子是一个字库集，其中每一显示表标识是一个字符的ASCII值。在定义几个字库时，使用glListBase函数中的offsetValue来获得数组listIDArray中描述的特定字库。

3.23.3 删除OpenGL显示表

要删除连续的一组显示表，可调用函数：

```
glDeleteLists (startID, nLists);
```

参数startID给出最前面的显示表标识，而参数nLists给出要删除的显示表总数。例如，语句：

```
glDeleteLists (5, 4);
```

删除4个显示表，其标识为5、6、7和8。没有显示表对应的标识被忽略。

3.24 OpenGL显示窗口重定形函数

在介绍性的OpenGL程序（参见2.9节）中，我们讨论了建立初始显示窗口的函数。但是在生成图形后，常需要用鼠标将显示窗口拖到屏幕的另一位置或改变其形状。改变显示窗口的尺寸可能改变其纵横比并引起对象形状的改变。

为了允许对显示窗口尺寸的改变做出反应，GLUT库提供下面的函数：

```
glutReshapeFunc (winReshapeFcn);
```

该函数可和其他GLUT函数一起放在程序的主过程中，它在显示窗口尺寸输入后立即激活。该GLUT函数的变量是接受新窗口宽度和高度的过程名。我们可以接着使用新尺寸去重新设置投影参数并完成任何其他操作，包括改变显示窗口颜色。另外，我们可以保存宽度和高度给程序中的其他过程使用。

作为一个例子，下列程序展示了怎样构造winReshapeFcn过程。命令glLoadIdentity包含在重定形函数中，从而使前面任意的投影参数值对新的投影设置不起作用。该程序显示了3.23节讨论的规则六边形。尽管本例中的六边形中心（在圆的中心位置）用显示窗口参数的概念描述，但是该六边形的位置不受显示窗口尺寸的任何改变的影响。这是因为六边形在显示表中定义，并且仅仅是最初的中心坐标存储在表中。如果希望在改变显示窗口尺寸时改变六边形的位置，则需要使用另一种方法来定义六边形或改变显示窗口的坐标参考。图3.64给出了该程序的输出。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

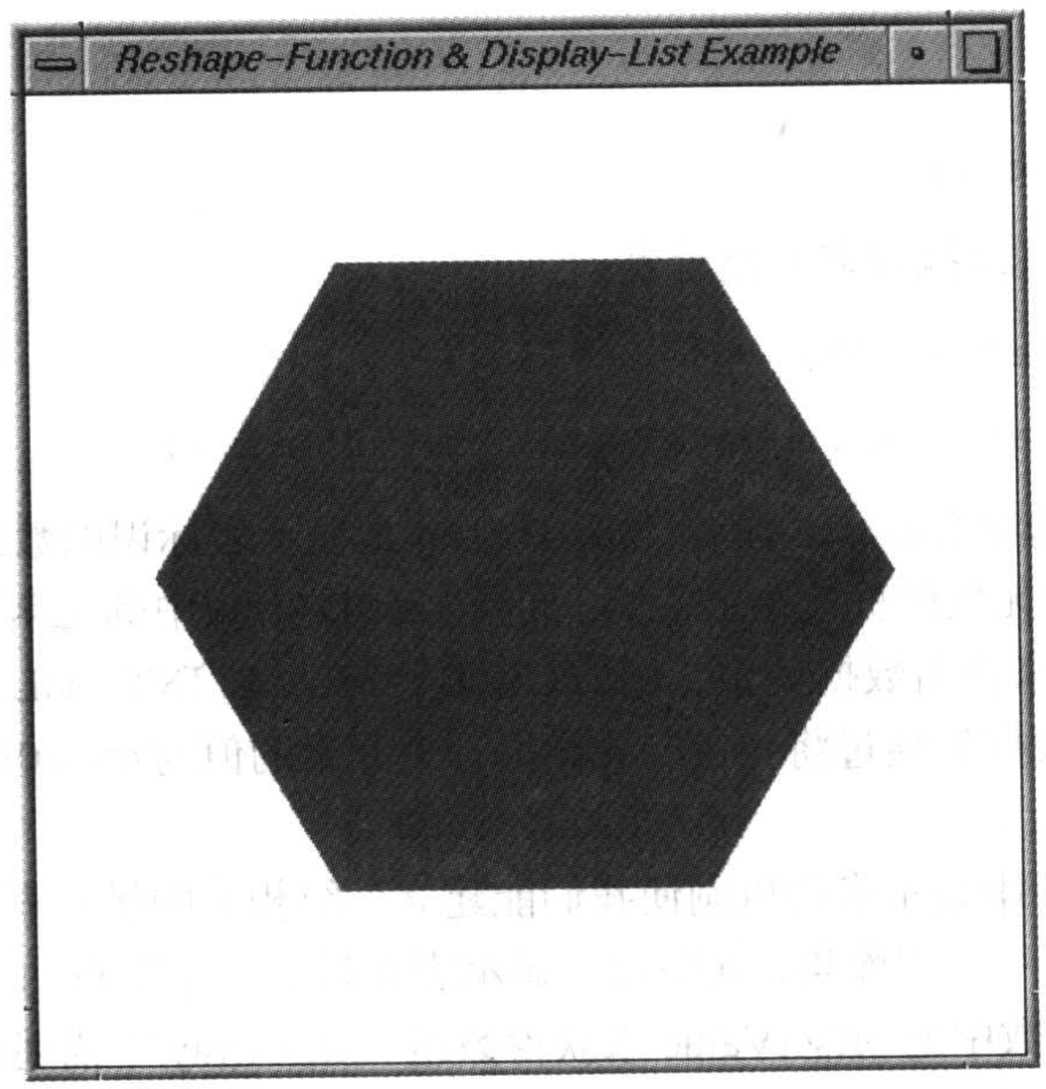


图 3.64　由例子程序给出的显示窗口，展示重定形函数的应用

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>

const double TWO_PI = 6.2831853;

/*  Initial display-window size.  */
GLsizei winWidth = 400, winHeight = 400;
GLuint regHex;

class screenPt
{
      private:
         GLint x, y;

      public:
         /*  Default Constructor: initializes coordinate position to (0, 0).  */
         screenPt ( )  {
             x = y = 0;
         }

         void setCoords (GLint xCoord, GLint yCoord)  {
             x = xCoord;
             y = yCoord;
         }

         GLint getx ( ) const  {
             return x;
         }

         GLint gety ( ) const  {
             return y;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        }
};

static void init (void)
{
   screenPt hexVertex, circCtr;
   GLdouble theta;
   GLint k;

   /*  Set circle center coordinates.  */
   circCtr.setCoords (winWidth / 2, winHeight / 2);

   glClearColor (1.0, 1.0, 1.0, 0.0);   //  Display-window color = white.

   /*  Set up a display list for a red regular hexagon.
    *  Vertices for the hexagon are six equally spaced
    *  points around the circumference of a circle.
    */
   regHex = glGenLists (1);   //  Get an identifier for the display list.
   glNewList (regHex, GL_COMPILE);
      glColor3f (1.0, 0.0, 0.0);   //  Set fill color for hexagon to red.
      glBegin (GL_POLYGON);
         for (k = 0; k < 6; k++) {
            theta = TWO_PI * k / 6.0;
               hexVertex.setCoords (circCtr.getx ( ) + 150 * cos (theta),
                                    circCtr.gety ( ) + 150 * sin (theta));
               glVertex2i (hexVertex.getx ( ), hexVertex.gety ( ));
         }
      glEnd ( );
   glEndList ( );
}

void regHexagon (void)
{
   glClear (GL_COLOR_BUFFER_BIT);

   glCallList (regHex);

   glFlush ( );
}

void winReshapeFcn (int newWidth, int newHeight)
{
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );
   gluOrtho2D (0.0, (GLdouble) newWidth, 0.0, (GLdouble) newHeight);

   glClear (GL_COLOR_BUFFER_BIT);
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (100, 100);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    glutInitWindowSize (winWidth, winHeight);
    glutCreateWindow ("Reshape-Function & Display-List Example");

    init ( );
    glutDisplayFunc (regHexagon);
    glutReshapeFunc (winReshapeFcn);

    glutMainLoop ( );
}
```

3.25 小结

本章中讨论的输出图元为使用直线、曲线、填充区域、单元阵列样式和文本构造图形提供了基本的工具。我们通过在笛卡儿世界坐标系统中给出几何描述来指定图元。在图 3.65 和图 3.66 中给出了用输出图元生成图形的例子。

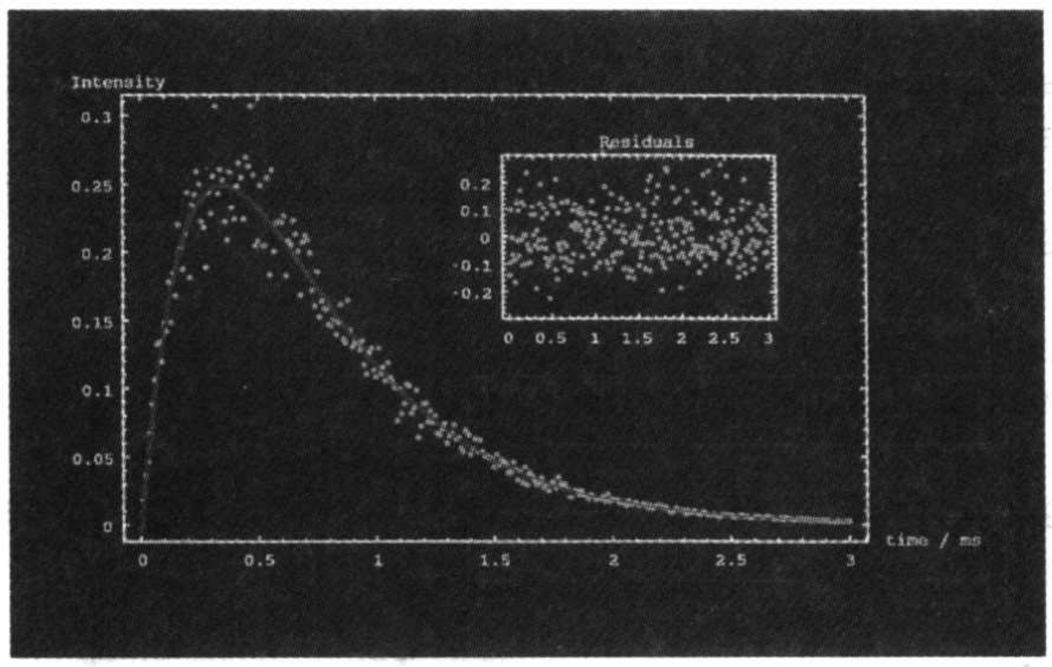


图 3.65 利用直线段、曲线、字符标记符号及文字显示的数据图（Wolfram Research 提供）

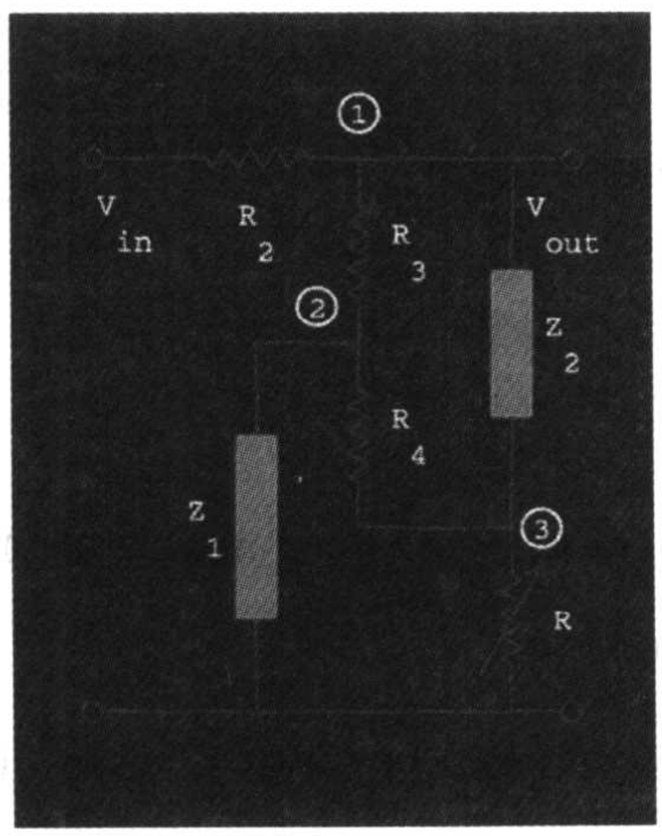


图 3.66 用直线段、圆、填充区和文字绘制的电路图（Wolfram Research 提供）

用于沿线段路径绘制像素的三种方法是DDA算法、Bresenham算法和中点算法。Bresenham 算法和中点算法是等同的并且是最有效的。沿线段路径的像素的颜色值存储按照递增地计算内存地址的方式而有效完成。任何线段生成算法都可以通过分割线段并将分割的线段分布到可用处理器上来获得并行的实现。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

圆和椭圆采用中点算法并根据其对称性进行有效而精确的扫描转换。其他二次曲线(抛物线和双曲线)也可以使用类似的方法进行绘制。分段的连续多项式的样条曲线广泛地应用于动画和计算机辅助设计中。曲线生成的并行实现能通过与并行线段处理方法类似的方法来实现。

为了考虑显示直线和曲线具有有限宽度的事实,我们必须调整对象的像素大小,使之与指定的几何尺寸相一致。可以通过将像素位置看做在左下角的编址方法,或是通过调整直线长度的方法来实现。

一个填充区是一个显示成单色或彩色图案的平面区域。但一般来说,我们可以用任何边界来指定填充区。很多图形软件包仅允许凸多边形填充区。这时,凹多边形填充区可以通过分割成一组凸多边形来显示。三角形是最容易填充的多边形,因为每条扫描线只和三角形的两条边相交(假定扫描线不和三角形顶点相交)。

奇偶规则可用来判定平面区域的内点。其他一些方法也可用于定义对象内部,特别是不规则的自相交对象。一个有代表性的例子是非零环绕数规则。该规则比奇偶规则在处理用多个边界定义的对象时更加灵活。我们还可以根据环绕数规则的变形,使用布尔操作来组合平面区域。

每一多边形都有确定多边形平面空间方向的前向面和后向面。该空间方向可用与多边形平面正交的法向量来确定,并且从后向面指向前向面。可以从多边形平面方程或使用平面上逆时针排列且其三个夹角均小于 180° 的三点求向量叉积来计算法向量的分量。一个场景的所有坐标值、空间方向和其他几何数据分别放入顶点、边和面片表中。

图形软件包中其他一些可用的图元有图案阵列和字符串。图案阵列可用于描述各种二维形状,包括使用矩形结构的二值或彩色值集合表达的字符集。字符串用来为图形提供标记。

使用OpenGL基本库中的图元函数可生成点、直线段、凸多边形填充区和位图或像素图的图案阵列。GLUT中有显示字符串的子程序。圆、椭圆和凸多边形填充区等其他图元能利用这些函数构造或逼近,也可利用GLU和GLUT的子程序生成。所有坐标值在右手笛卡儿坐标系统中用绝对坐标表示。描述场景的坐标位置可在二维或三维参考系中给出。可以用整数或浮点数来给出坐标值,也可用指向坐标值数组的指针来表示位置。场景描述经观察函数变换成视频监视器等输出设备的二维显示。除glRect函数外,顶点、线段或多边形的每一位置均在glVertex函数中指定。定义每一图元的一组glVertex函数用一对语句glBegin/glEnd来包含,其中的图元类型根据作为glBegin函数的变量的符号常量来标识。在描述包含许多多边形填充表面时,可以使用OpenGL顶点数组来指定几何和其他数据,从而高效地生成显示结果。

在表3.1中,我们列出了OpenGL中生成输出图元的基本函数。某些相关子程序也列在此表中。

表 3.1　OpenGL 输出图元函数及相关子程序小结

函数	描述
gluOrtho2D	指定二维世界坐标系统
glVertex*	选择一坐标位置。该函数必须放在 glBegin/glEnd 之间
glBegin (GL_POINTS);	绘出一个或多个点,每个都在 glVertex 函数中指定。该位置串用 glEnd 语句来结束
glBegin (GL_LINES);	显示一组直线段,其端点坐标在 glVertex 函数中指定。该端点串最后由 glEnd 语句来结束
glBegin (GL_LINE_STRIP);	显示用与 GL_LINES 相同的结构所指定的折线
glBegin (GL_LINE_LOOP);	显示用与 GL_LINES 相同的结构所指定的封闭折线
glRect*	显示 *xy* 平面上的一个填充区
glBegin (GL_POLYGON);	显示一个填充多边形,其顶点在 glVertex 中给出且由 glEnd 语句结束
glBegin (GL_TRIANGLES);	显示一组填充三角形,其描述结构与 GL_POLYGON 相同

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

（续表）

函数	描述
glBegin (GL_TRIANGLE_STRIP);	显示一个填充三角形带，其描述结构与 GL_POLYGON 相同
glBegin (GL_TRIANGLE_FAN);	显示一扇形的填充三角形带，所有三角形都与第一顶点相连，其描述结构与 GL_POLYGON 相同
glBegin (GL_QUADS);	显示一组填充四边形，其描述结构与 GL_POLYGON 相同
glBegin (GL_QUAD_STRIP);	显示一组填充四边形带，其描述结构与 GL_POLYGON 相同
glEnableClientState (GL_VERTEX_ARRAY);	激活 OpenGL 的顶点数组设施
glVertexPointer (size, type, stride, array);	指定一坐标值数组
glDrawElements (prim, num, type, array);	从数组数据中显示一指定图元类型
glNewList (listID, listMode);	把一组命令定义为一个显示表，用 glEndList 语句结束
glGenLists	生成一个或多个显示表标识
glIsList	确定一显示表标识是否被使用的查询函数
glCallList	执行一个显示表
glListBase	指定显示表标识数组的位移
glCallLists	执行多个显示表
glDeleteLists	删除指定的一串显示表
glRasterPos*	为帧缓存指定一个二维或三维的当前位置。该位置用来作为位图和像素图图案的参考
glBitmap (w, h, x0, y0, xShift, yShift, pattern);	指定要映射到与当前位置对应的像素位置的位图图案
glDrawPixels (w, h, type, format, pattern);	指定要映射到与当前位置对应的像素位置的像素图图案
glDrawBuffer	选择存储像素图的一个或多个缓存
glReadPixels	将一块像素存入指定的数组
glCopyPixels	将一块像素从一个缓存复制到另一个
glLogicOp	在用常量 GL_COLOR_LOGIC_OP 激活后选择一种逻辑操作来组合两个像素数组
glutBitmapCharacter (font, char);	选择一种字体和一个位图字符进行显示
glutStrokeCharacter (font, char);	选择一种字体和一个轮廓字符进行显示
glutReshapeFunc	指定显示窗口尺寸改变时的工作

示例程序

这里，我们给出一些 OpenGL 程序来展示输出图元的使用。每一程序使用了表 3.1 中列出的一个或几个函数。每个程序使用第 2 章讨论的 GLUT 子程序，并在建立的显示窗口中给出输出。

第一个程序展示使用一条折线、一组多点标记和位图字符标号来生成一年内按月的折线图。尽管等宽字体与图中的位置比较容易对齐，但此处给出了比例空间的字体。由于位图左下角提供了当前光栅位置的参考，所以我们必须移动参考位置以使文字串中心与绘制数据的位置对齐。图 3.67 给出了折线图程序的输出。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
#include <GL/glut.h>

GLsizei winWidth = 600, winHeight = 500;    // Initial display window size.
GLint xRaster = 25, yRaster = 150;          // Initialize raster position.

GLubyte label [36] = {'J', 'a', 'n',   'F', 'e', 'b',   'M', 'a', 'r',
                      'A', 'p', 'r',   'M', 'a', 'y',   'J', 'u', 'n',
                      'J', 'u', 'l',   'A', 'u', 'g',   'S', 'e', 'p',
                      'O', 'c', 't',   'N', 'o', 'v',   'D', 'e', 'c'};

GLint dataValue [12] = {420, 342, 324, 310, 262, 185,
                        190, 196, 217, 240, 312, 438};

void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 1.0);    // White display window.
    glMatrixMode (GL_PROJECTION);
    gluOrtho2D (0.0, 600.0, 0.0, 500.0);
}

void lineGraph (void)
{
    GLint month, k;
    GLint x = 30;                         // Initialize x position for chart.

    glClear (GL_COLOR_BUFFER_BIT);        //  Clear display window.
    glColor3f (0.0, 0.0, 1.0);            //  Set line color to blue.
    glBegin (GL_LINE_STRIP);              //  Plot data as a polyline.
        for (k = 0; k < 12; k++)
            glVertex2i (x + k*50, dataValue [k]);
    glEnd ( );


    glColor3f (1.0, 0.0, 0.0);          //  Set marker color to red.
    for (k = 0; k < 12; k++) {          //  Plot data as asterisk polymarkers.
        glRasterPos2i (xRaster + k*50, dataValue [k] - 4);
        glutBitmapCharacter (GLUT_BITMAP_9_BY_15, '*');
    }


    glColor3f (0.0, 0.0, 0.0);            //  Set text color to black.
    xRaster = 20;                         //  Display chart labels.
    for (month = 0; month < 12; month++) {
        glRasterPos2i (xRaster, yRaster);
        for (k = 3*month; k < 3*month + 3; k++)
          glutBitmapCharacter (GLUT_BITMAP_HELVETICA_12, label [k]);
        xRaster += 50;
    }
    glFlush ( );
}


void winReshapeFcn (GLint newWidth, GLint newHeight)
{
    glMatrixMode (GL_PROJECTION);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    glLoadIdentity ( );
    gluOrtho2D (0.0, GLdouble (newWidth), 0.0, GLdouble (newHeight));

    glClear (GL_COLOR_BUFFER_BIT);
}


void main (int argc, char** argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (100, 100);
    glutInitWindowSize (winWidth, winHeight);
    glutCreateWindow ("Line Chart Data Plot");

    init ( );
    glutDisplayFunc (lineGraph);
    glutReshapeFunc (winReshapeFcn);

    glutMainLoop ( );
}
```

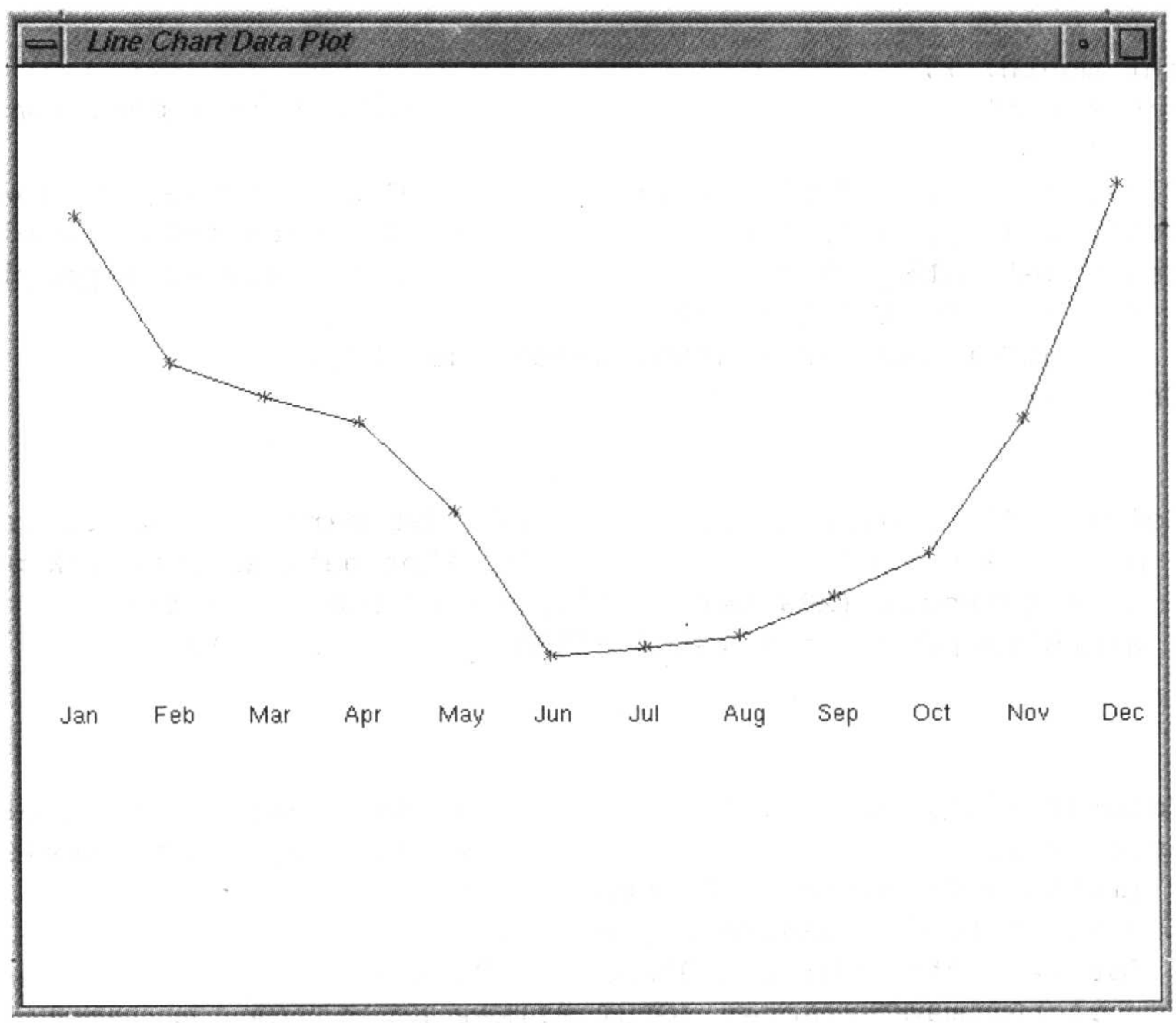


图 3.67　lineGraph 子程序输出的数据点折线和多点标记绘制

第二个程序使用同样的数据集生成图 3.68 所示的直方图。该程序展示了矩形填充区及位图字符标号的应用。

```
void barChart (void)
{
    GLint month, k;

    glClear (GL_COLOR_BUFFER_BIT); //  Clear display window.
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glColor3f (1.0, 0.0, 0.0);      //  Set bar color to red.
for (k = 0; k < 12; k++)
   glRecti (20 + k*50, 165, 40 + k*50, dataValue [k]);

glColor3f (0.0, 0.0, 0.0);      //  Set text color to black.
xRaster = 20;                   //  Display chart labels.
for (month = 0; month < 12; month++) {
   glRasterPos2i (xRaster, yRaster);
   for (k = 3*month; k < 3*month + 3; k++)
      glutBitmapCharacter (GLUT_BITMAP_HELVETICA_12,
                                                  label [h]);
      xRaster += 50;
   }
   glFlush ( );
}
```

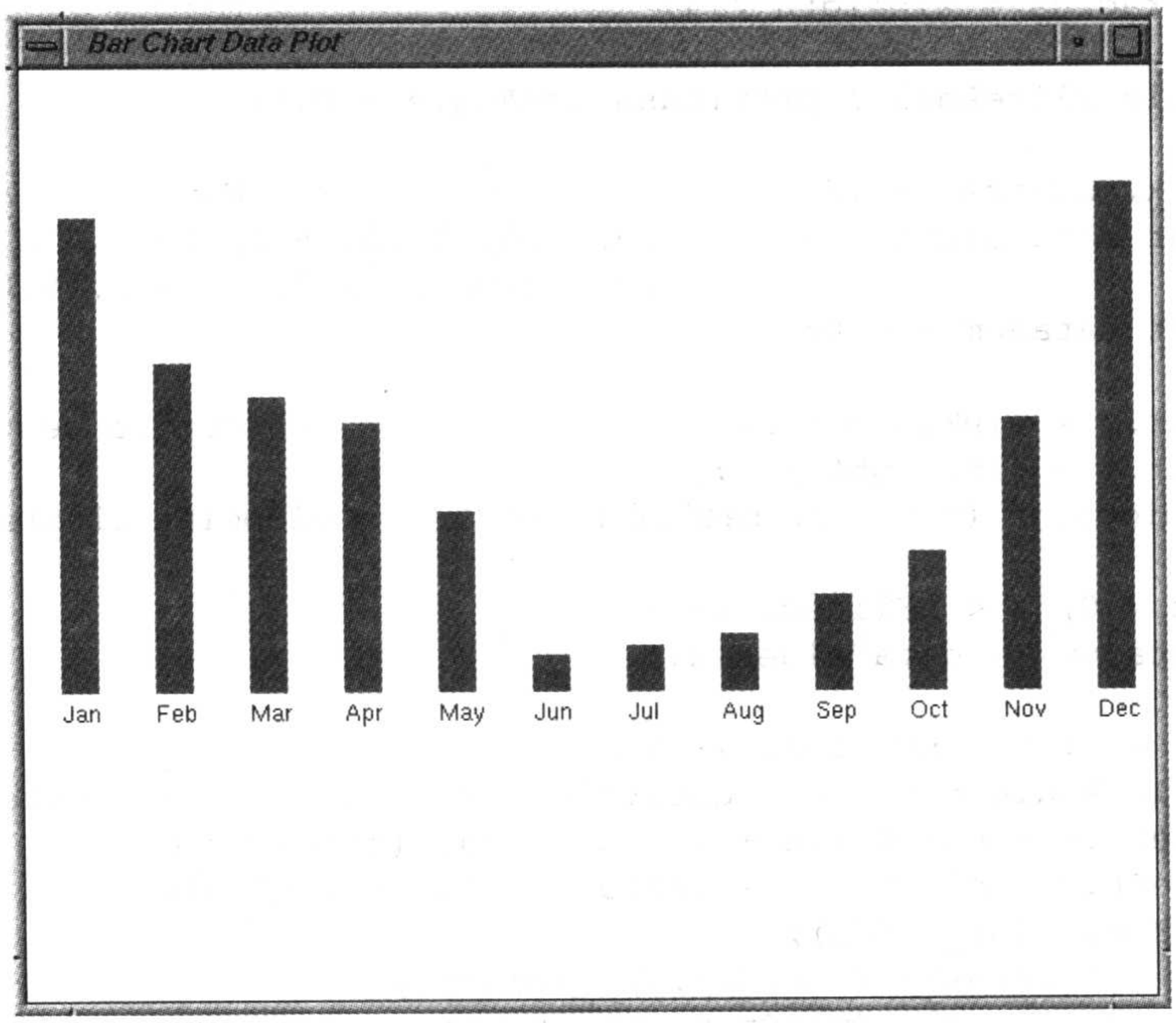


图 3.68 barChart 过程生成的直方图

饼图用来给出整体中各部分的分布比例。下一程序使用中点圆算法子程序来构造一个饼图。例子中的值用于确定扇形的数量和大小，该程序的输出请参见图 3.69。

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

const GLdouble twoPi = 6.283185;

class scrPt {
public:
    GLint x, y;
};

GLsizei winWidth = 400, winHeight = 300;    // Initial display window size.
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 1.0);

    glMatrixMode (GL_PROJECTION);
    gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

    .                          //  Midpoint routines for displaying a circle.
    .
    .

void pieChart (void)
{
    scrPt circCtr, piePt;
    GLint radius = winWidth / 4;                // Circle radius.

    GLdouble sliceAngle, previousSliceAngle = 0.0;

    GLint k, nSlices = 12;                      // Number of slices.
    GLfloat dataValues[12] = {10.0, 7.0, 13.0, 5.0, 13.0, 14.0,
                              3.0, 16.0, 5.0, 3.0, 17.0, 8.0};
    GLfloat dataSum = 0.0;

    circCtr.x = winWidth / 2;                   // Circle center position.
    circCtr.y = winHeight / 2;
    circleMidpoint (circCtr, radius);  // Call a midpoint circle-plot routine.

    for (k = 0; k < nSlices; k++)
        dataSum += dataValues[k];

    for (k = 0; k < nSlices; k++) {
        sliceAngle = twoPi * dataValues[k] / dataSum + previousSliceAngle;
        piePt.x = circCtr.x + radius * cos (sliceAngle);
        piePt.y = circCtr.y + radius * sin (sliceAngle);
        glBegin (GL_LINES);
            glVertex2i (circCtr.x, circCtr.y);
            glVertex2i (piePt.x, piePt.y);
        glEnd ( );
        previousSliceAngle = sliceAngle;
    }
}

void displayFcn (void)
{
    glClear (GL_COLOR_BUFFER_BIT);    //  Clear display window.

    glColor3f (0.0, 0.0, 1.0);        //  Set circle color to blue.

    pieChart ( );
    glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
    glMatrixMode (GL_PROJECTION);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    glLoadIdentity ( );
    gluOrtho2D (0.0, GLdouble (newWidth), 0.0, GLdouble (newHeight));

    glClear (GL_COLOR_BUFFER_BIT);

    /*  Reset display-window size parameters.  */
    winWidth = newWidth;
    winHeight = newHeight;
}

void main (int argc, char** argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (100, 100);
    glutInitWindowSize (winWidth, winHeight);
    glutCreateWindow ("Pie Chart");

    init ( );
    glutDisplayFunc (displayFcn);
    glutReshapeFunc (winReshapeFcn);

    glutMainLoop ( );
}
```

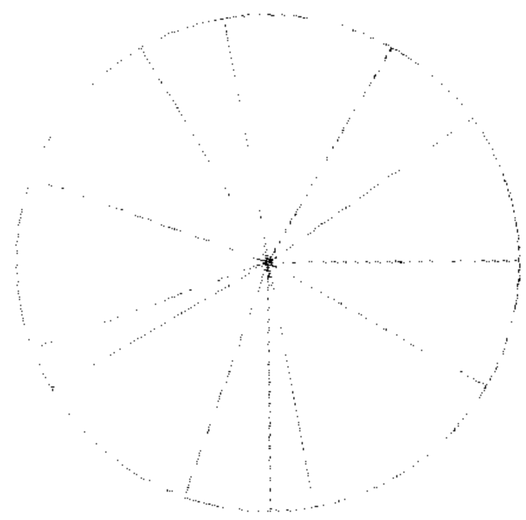


图 3.69 pieChart 过程生成的输出

最后一个程序显示了圆公式的某些变化，其中使用了参数极坐标方程（3.28）来计算曲线路径的点。这些点用做显示弧的逼近折线中直线段的端点。图 3.70 中的弧通过圆半径 r 的变化来生成。按照 r 的不同变化，可生成蜗形线、心形线、螺旋线或其他类似的图形。

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

#include <iostream.h>

struct screenPt
{
    GLint x;
    GLint y;
};

typedef enum { limacon = 1, cardioid, threeLeaf, fourLeaf, spiral } curveName;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
GLsizei winWidth = 600, winHeight = 500;    // Initial display window size.

void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 1.0);

    glMatrixMode (GL_PROJECTION);
    gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void lineSegment (screenPt pt1, screenPt pt2)
{
    glBegin (GL_LINES);
        glVertex2i (pt1.x, pt1.y);
        glVertex2i (pt2.x, pt2.y);
    glEnd ( );
}

void drawCurve (GLint curveNum)
{
    /*  The limacon of Pascal is a modification of the circle equation
     *  with the radius varying as r = a * cos (theta) + b, where a
     *  and b are constants.  A cardiod is a limacon with a = b.
     *  Three-leaf and four-leaf curves are generated when
     *  r = a * cos (n * theta), with n = 3 and n = 2, respectively.
     *  A spiral is displayed when r is a multiple of theta.
     */

    const GLdouble twoPi = 6.283185;
    const GLint a = 175, b = 60;

    GLfloat r, theta, dtheta = 1.0 / float (a);
    GLint x0 = 200, y0 = 250;   // Set an initial screen position.
    screenPt curvePt[2];

    glColor3f (0.0, 0.0, 0.0);        //  Set curve color to black.

    curvePt[0].x = x0;       // Initialize curve position.
    curvePt[0].y = y0;

    switch (curveNum) {
        case limacon:    curvePt[0].x += a + b;  break;
        case cardioid:   curvePt[0].x += a + a;  break;
        case threeLeaf:  curvePt[0].x += a;      break;
        case fourLeaf:   curvePt[0].x += a;      break;
        case spiral:     break;
        default:         break;
    }

    theta = dtheta;
    while (theta < two_Pi) {
        switch (curveNum) {
            case limacon:
                r = a * cos (theta) + b;    break;
            case cardioid:
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
            r = a * (1 + cos (theta));  break;
        case threeLeaf:
            r = a * cos (3 * theta);    break;
        case fourLeaf:
            r = a * cos (2 * theta);    break;
        case spiral:
            r = (a / 4.0) * theta;      break;
        default:                        break;
    }

        curvePt[1].x = x0 + r * cos (theta);
        curvePt[1].y = y0 + r * sin (theta);
        lineSegment (curvePt[0], curvePt[1]);

        curvePt[0].x = curvePt[1].x;
        curvePt[0].y = curvePt[1].y;
        theta += dtheta;
    }
}

void displayFcn (void)
{
    GLint curveNum;

    glClear (GL_COLOR_BUFFER_BIT);   //  Clear display window.

    cout << "\nEnter the integer value corresponding to\n";
    cout << "one of the following curve names.\n";
    cout << "Press any other key to exit.\n";
    cout << "\n1-limacon, 2-cardioid, 3-threeLeaf, 4-fourLeaf, 5-spiral:  ";
    cin  >> curveNum;

    if (curveNum == 1 || curveNum == 2 || curveNum == 3 || curveNum == 4
         || curveNum == 5)
        drawCurve (curveNum);
    else
        exit (0);

    glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ( );
    gluOrtho2D (0.0, (GLdouble) newWidth, 0.0, (GLdouble) newHeight);

    glClear (GL_COLOR_BUFFER_BIT);
}

void main (int argc, char** argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (100, 100);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    glutInitWindowSize (winWidth, winHeight);
    glutCreateWindow ("Draw Curves");

    init ( );
    glutDisplayFunc (displayFcn);
    glutReshapeFunc (winReshapeFcn);

    glutMainLoop ( );
}
```

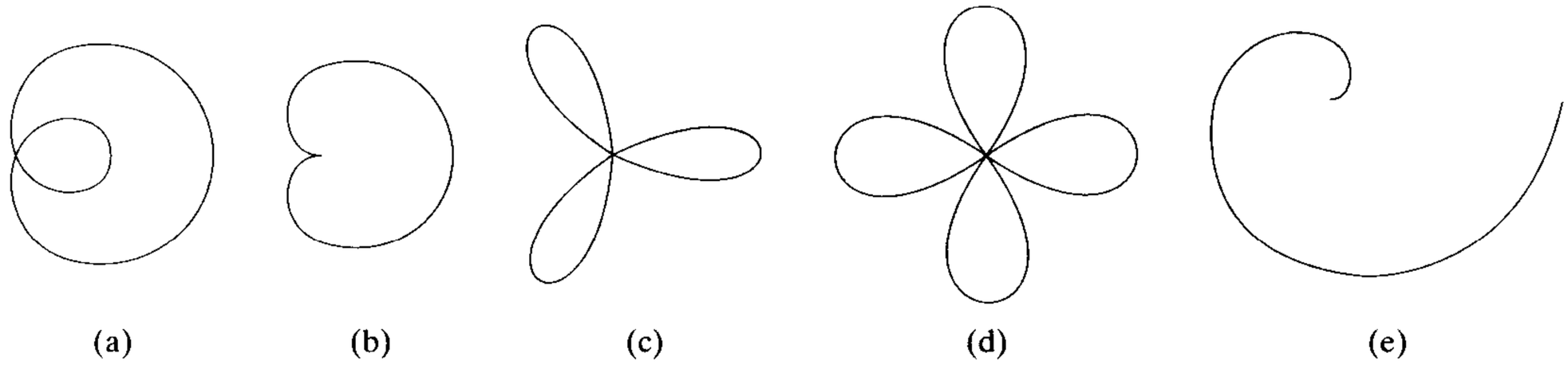


图 3.70 drawCurve 显示的曲线图：(a)蜗形线；(b)心形线；(c)三叶曲线；(d)四叶曲线；(e)螺旋线

参考文献

关于Bresenham算法的基本资料可在Bresenham(1965, 1977)中找到。有关中点算法的信息，请参见 Kappel(1985)，生成直线和圆的并行算法在 Pang(1990)和 Wright(1990)中讨论。生成和处理图元的许多其他方法在Glassner(1990)、Arvo(1991)、Kirk(1992)、Heckbert(1994)和Paeth(1995)中讨论。

使用 OpenGL 图元函数的其他编程例子可参见 Woo, Neider, Davis, and Shreiner(1999)。所有 OpenGL 图元函数的列表可在 Shreiner(2000)中找到。Kilgard(1996)中有完整的 GLUT 资料。

练习题

3.1 利用 DDA 算法实现给定任何数目（n）输入点的 polyline 函数，当 $n = 1$ 时绘制单个点。

3.2 考虑象限间的对称性而扩充 Bresenham 画线算法，从而生成具有任何斜率的直线。

3.3 实现一个折线函数，使用前面的算法来显示连接 n 个输入点的一组线段。当 $n = 1$ 时绘制单个点。

3.4 使用中点算法推导沿斜率在 $0 < m < 1$ 范围内的线段路径生成点的决策参数，并表示出中点决策参数与 Bresenham 画线算法中的相同。

3.5 使用中点算法推导用于生成具有任何斜率的线段的决策参数。

3.6 编写斜率在 $0 < m < 1$ 范围内的 Bresenham 画线算法的并行版本。

3.7 编写具有任何斜率的 Bresenham 画线算法的并行版本。

3.8 假设系统有一个 8 英寸 × 10 英寸、每英寸能显示 100 个像素的监视器。如果存储器的字长为一字节，起始帧缓存地址为0，并且每个像素对应存储器的1个字节，那屏幕坐标(x, y)在帧缓存中的地址是什么？

3.9 假如系统有一个 8 英寸 × 10 英寸、每英寸能显示 100 个像素的监视器，存储器的字长为一字节，起始帧缓存地址为 0，每个像素对应存储器的 6 位，那么具有屏幕坐标(x, y)的像素的帧缓存地址是什么？

3.10 把计算帧缓存地址的递归技术（参见 3.7 节）结合进 Bresenham 画线算法中。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3.11 修改中点画圆算法，使之用保存的输入几何数据来显示圆（参见 3.13 节）。

3.12 编写实现中点画圆并行算法的程序。

3.13 假设起始位置为$(r_x, 0)$，并且以逆时针方向沿曲线路径生成点，推导中点椭圆算法的决策参数。

3.14 编写中点椭圆算法的并行实现的程序。

3.15 利用对称性优点，设计显示一个周期正弦函数（sin）的有效算法。

3.16 修改前一练习题的算法，设计显示任一指定角度范围的正弦函数（sin）的有效算法。

3.17 利用对称性优点，设计显示阻尼振荡运动函数：

$$y = Ae^{-kx}\sin(\omega x + \theta)$$

的有效算法。其中 ω 为角速度，θ 为正弦函数的相位，对正弦函数的多个周期或直到最大振幅减少到 $A/10$ 时绘制 x 的函数 y。

3.18 利用中点算法并考虑对称性，推导在区间 $-10 \leqslant x \leqslant 10$ 上，对下列曲线进行扫描转换的有效算法：

$$y = \frac{1}{12}x^3$$

3.19 利用中点算法并考虑对称性，对抛物线：

$$y = 100 - x^2$$

在区间 $-10 \leqslant x \leqslant 10$ 进行扫描转换。

3.20 利用中点算法并考虑对称性，对抛物线：

$$x = y^2$$

在区间 $-10 \leqslant y \leqslant 10$ 进行扫描转换。

3.21 考虑对称性，建立中点算法对形式为

$$y = ax^2 + b$$

的任意抛物线进行扫描转换，参数 a、b 及 x 的范围从输入值获得。

3.22 建立一个如图 3.50 所示的单位立方体的几何数据表。

3.23 仅使用顶点表和面片表建立一个单位立方体的几何数据表，然后仅用面片表存储同样的信息。将这两种表示单位立方体的方法与使用练习中三张表的表示方法进行比较。估计每一种方法的存储开销。

3.24 为圆柱体定义一种有效的多边形网格表示，并调整对表示的选择。

3.25 编写一个为定义三维对象表面的多边形面片而输入的任意点集建立几何数据表的过程。

3.26 设计一个用来检查图 3.50 中几何数据表的一致性和完整性的子程序。

3.27 编写一个为多边形面片网输入计算参数 A、B、C 和 D 的程序。

3.28 给定多边形的参数 A、B、C 和 D，编写一个确定一输入坐标点是否在该多边形表面的前方或后方的程序。

3.29 如果一场景的坐标从一个右手系统改变为左手系统，我们需要对表面的平面参数 A、B、C 和 D 做什么样的修改来保证平面的方向描述还是正确的。

3.30 开发一个识别非共面四边形顶点表的程序。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3.31 扩展前一练习题的程序，使之识别包含4个以上坐标位置的非共面顶点表。

3.32 编写一个将一组四个多边形顶点位置分解为一组三角形的程序。

3.33 编写一个将一组 n（$n>4$）个多边形顶点位置分解为一组三角形的程序。

3.34 设计一个识别可能包含重复顶点或共线顶点的退化多边形顶点表的算法。

3.35 设计一个识别包含自相交边的多边形顶点表的算法。

3.36 编写一个通过计算一对边向量叉积来识别凹多边形的子程序。

3.37 编写一个利用向量方法分解凹多边形的子程序。

3.38 编写一个利用旋转方法分解凹多边形的子程序。

3.39 利用非零环绕数规则和叉积计算来鉴别相交边方向的方法，设计对任何一组输入顶点确定内部区域的算法。

3.40 利用非零环绕数规则和点积计算来鉴别相交边方向的方法，设计对任何一组输入顶点确定内部区域的算法。

3.41 图3.46中自相交折线的哪一个区域有正的非零环绕数？哪些是有负的非零环绕数的区域？哪个区域的非零环绕数大于1？

3.42 编写一个实现文字串函数的子程序，该子程序有两个参数：一个参数指定一个世界坐标位置而另一参数指定文字串。

3.43 编写一个实现多点标记函数的子程序，该子程序有两个参数：一个参数指定显示的字符而另一参数指定一串世界坐标位置。

3.44 修改3.24节的示例程序使显示的六边形总是在显示窗口的中间，不管显示窗口大小如何变化。

3.45 编写一个显示直方图的程序。对程序的输入包括数据点以及 x 和 y 轴所需的标记，程序应能够缩放数据点，从而使图形充满整个屏幕区域。

3.46 编写一个在显示窗口中的任意选定区域显示直方图形的程序。

3.47 对于任意输入的一组数据点，编写一个在屏幕的任意选定区域内显示折线图的程序，并对输入的数据组进行缩放，使其充满所选择的屏幕区域。数据点位置用星号显示并用线段连接，然后按照输入的规则给 x 和 y 轴加上标记（也可以用小圆或其他符号代替绘制的数据点）。

3.48 利用圆函数，编写一个程序，显示具有合适标记的饼图。程序的输入包括：在某些区间上给定数据分布的数据组，饼图的名称和区间的名称。每部分的标记将显示在饼图边界外靠近对应饼图部分的地方。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第4章　图元的属性

通常，任何影响图元显示方法的参数一般称为**属性参数**（attribute parameter）。诸如颜色和大小等属性参数确定了图元的基本特性。其他则指出在特定条件下怎样显示图元。特定条件属性的例子有在交互式对象选择程序中的可见性或可检测性。这类特定条件属性将在后面几章中详细讨论。这里，我们仅仅考虑控制图元的基本显示特性的属性，而不考虑特定的条件。例如，线段可以是点线或划线、粗线或细线以及蓝色或橙色。区域可以使用一种颜色或多色图案填充。文本可以按从左到右的阅读方式进行显示，也可以沿屏幕对角线的倾斜方向或是按垂直列向进行显示。每一字符可用不同字体、颜色和大小来显示。我们也可以在对象的边上应用亮度变化来平滑光栅阶梯效果。

将属性选择加入图形软件包的一种方法是，为每个输出图元功能扩充相关的参数表，从而包含合适的属性。例如，画线功能除了包括端点坐标以外，还可以包含颜色、宽度和其他属性的参数。另一种方法是提供一张系统当前属性值表，并使用包含在图形软件包中的独立函数来为属性表设置当前值。为了生成一个输出图元，系统要检测相关的属性，并使用当前属性设置来调用该显示程序。有些图形软件包使用两种设定属性值方法的组合，而OpenGL等其他图形库则用更新系统属性表的独立函数来设定属性。

维护属性和其他参数当前值表的图形系统称为**状态系统**（state system）或**状态机**（state machine）。输出图元的属性和当前帧缓存位置等其他参数称为**状态变量**（state variable）或**状态参数**（state parameter）。在给一个或几个状态参数赋值时，系统进入一个特定状态。该状态一直保留到状态参数的值再次改变。

4.1　OpenGL状态变量

属性值和其他参数设置由定义当前OpenGL状态的独立函数指定。OpenGL中的状态变量有颜色和其他图元属性、当前矩阵模式、模型观察矩阵的元素、缓存当前位置和场景光照效果参数等。所有OpenGL参数都有默认值，它们在被指定新值前保持不变并发挥作用。任何时候我们都可以查询系统的状态参数的当前值。本章后面几节中只讨论输出图元的属性设定，后面几章将会涉及其他参数。

OpenGL的所有图元使用当前状态表中的属性显示。改变一个或几个属性设定只能影响OpenGL状态改变后指定的那些图元。因此，我们可以先显示一绿色线段，接着将当前颜色改为红色，然后再定义另一线段。这样可以一起显示绿色线段和红色线段。同样，有些OpenGL状态的值可以在glBegin/glEnd函数对中间和坐标值一起指定，从而使参数设定可以因坐标位置而变化。

4.2　颜色和灰度

颜色是所有图元的一个基本属性。用户可以选择多种颜色，依赖于特定系统的能力和设计目标。颜色可用数值指定也可以从菜单或显示的标尺中选择。对于视频监视器而言，这些颜色编码转换成控制电子束的强度等级。在彩色绘图仪中，颜色码可用来控制喷墨量或笔的选择。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

4.2.1 RGB 颜色分量

在彩色光栅系统中，可选用的颜色数量依赖于帧缓存中提供的存储容量。颜色信息可用两种方式存储在帧缓存中：直接在帧缓存中存储 RGB 编码，或将颜色码存入一个独立的表中并在像素位置存储指向颜色表表项的索引。使用直接存储方案时，当应用程序指定一特定颜色编码后，该颜色信息被放入将用该颜色显示的输出图元的组成像素位置的帧缓存中。表 4.1 给出了使用每像素 3 位的这种方案可以提供的有限颜色。每一位置的 3 位值用来控制 RGB 监视器相应的电子枪的强度等级（此时，开或并）。最左边一位控制红枪，中间位控制绿枪，而最左边的一位控制蓝枪。在帧缓存中增加一些位可增加可选择的颜色数量。每像素有 6 位时，每支枪可用 2 位来控制。这可使三支颜色枪的每支得到 4 个不同强度的控制，而使每个屏幕像素有 64 种颜色可选用。随着提供的颜色数量的增加，帧缓存的容量要求也增加。1024 × 1024 分辨率的全彩色（每像素 24 位）RGB 系统需要 3 MB 的帧缓存容量。

表 4.1 每像素 3 位帧缓存的八种 RGB 颜色编码

颜色码	帧缓存中存储的颜色值			显示颜色
	红色	绿色	蓝色	
0	0	0	0	黑色
1	0	0	1	蓝色
2	0	1	0	绿色
3	0	1	1	青色
4	1	0	0	红色
5	1	0	1	黑色
6	1	1	0	品红
7	1	1	1	白色

颜色表是一种不需大帧缓存而为用户扩充颜色能力的替代方法。这种方法曾经是非常重要的考虑方式。但是现在的硬件成本已大幅下降，甚至在低端的个人计算机系统中扩充颜色能力也已相当普遍。因此，我们的多数例子中将简单地假定 RGB 编码直接存储在帧缓存中。

4.2.2 颜色表

图 4.1 给出了在**颜色查找表**（color lookup table）或**颜色表**（color map）中存储颜色值的一种可能方案。有时颜色表还称为**视频查找表**（video lookup table）。这时帧缓存中的值用做指向颜色表的索引。在本例中，每一像素可引用 256 个表位置中的任意一个，而每一表项使用 24 位来指定一个 RGB 颜色。十六进制颜色码 0x0821 使像素位置(x, y)处显示绿蓝混合色。使用这种特殊的查找表的系统让用户可从近 1700 万种颜色中任选 256 种颜色同时显示。与全彩色系统相比，这种方案减少了可同时显示的颜色量，但也使帧缓存容量减少到只要 1 MB。处理反走样等特殊的绘制应用及有多个输出设备时有时需要多个颜色表。

颜色表用于许多应用中，提供了“合理”数量的可同时显示的颜色而不要求大容量帧缓存。在多数应用中，256 或 512 种颜色对显示一幅图是足够的。另外，表项内容可以在任何时候进行修改，使用户可以很容易地试验设计、场景和图表中的不同颜色组合，而不需改变图形数据结构中属性的设定。当颜色表中某项的值改变时，所有使用该颜色索引的像素都将改成新颜色。如果不使用颜色表，则只能通过在帧缓存位置存入新颜色来修改某些像素的颜色。类似地，数据可视化应用可以将某些物理量（如能量的值）存入帧缓存，并使用查找表试验各种颜色组合而不必改变像素值。在可视化和图像处理应用中，颜色表是很方便的工具，用来设定阈值、使像素值低于指定阈值时均显示同一颜色。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

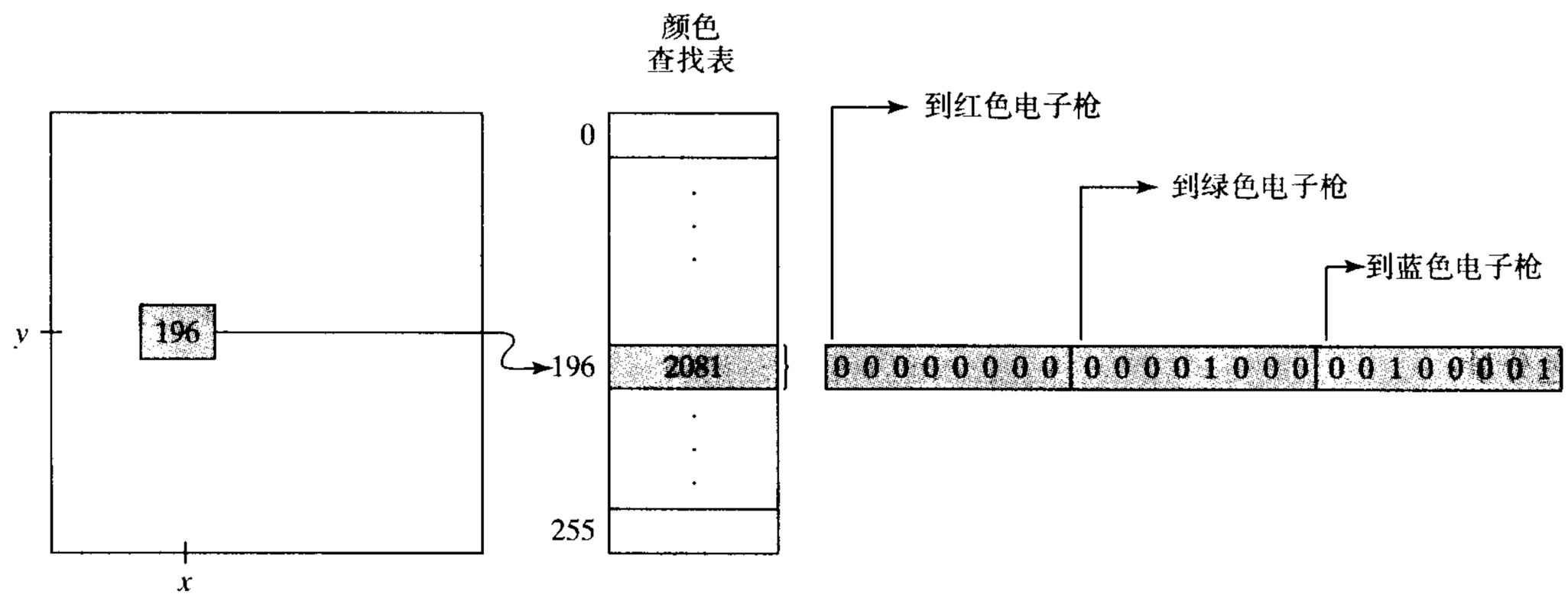


图 4.1　由每像素 8 位的帧缓存来访问的每项 24 位的颜色查找表。像素位置(x, y)存放值 196，指向该表中包含十六进制值 0x0821（十进制值为 2081）的项位置。该项的每一个 8 位的段控制 RGB 监视器中三根电子枪中的一根

4.2.3　灰度

由于计算机图形系统都具有彩色功能，我们可以在应用程序中使用 RGB 颜色函数来设定灰色程度或**灰度**（gray scale）。当 RGB 函数中指定相同量的红色、绿色和蓝色时，结果是某种程度的灰色。靠近 0 的值生成暗灰色，而靠近 1 的值生成亮灰色。灰度显示方法的应用包括增强黑白照片和产生可视化效果。

4.2.4　其他颜色参数

除了 RGB 颜色描述，计算机图形应用还使用一些其他的三分量颜色描述。例如，打印机输出颜色用青色、品红和黄色三分量来描述，而颜色接口有时用亮和暗来选择颜色。通常意义下的颜色和光是一种复杂现象，在光学、辐射度和心理学中提出了许多术语及概念，用来描述光源和光照效果的各个方面。在物理上，一种颜色可以描述为有一定频率范围和能量分布的电磁辐射，但也涉及我们对颜色的感觉。因此，我们使用物理术语强度（intensity）来量化一个时间段中在特定方向的光能辐射，而用心理学术语亮度来描述感觉光亮的特征。

4.3　OpenGL 颜色函数

第2章末尾的示例程序中给出了几个OpenGL颜色子程序。其中一个函数用来设定显示窗口的颜色，另一个指定直线段的颜色。同样，可以用下面的函数设定**颜色显示模型**（color display mode）为 RGB：

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
```

变量表中的第一个参数指示正在使用单个帧缓存，第二个参数设定 RGB 模式，即默认模型。我们可以用 GLUT_RGB 或 GLUT_RGBA 之一来选择颜色模型。如果要用指向颜色表的索引来指定颜色，则用 OpenGL 常量 GLUT_INDEX 取代 GLUT_RGB。

4.3.1　OpenGL 的 RGB 和 RGBA 颜色模型

多数 OpenGL 图元使用 **RGB 模式**（RGB mode），它与 **RGBA 模式**（RGBA mode）基本相同。RGB 和 RGBA 的惟一区别在于是否正在颜色调和中使用α值。在为图元指定一组特殊的颜色时，定

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

义了 OpenGL 的颜色状态（color state）。当前颜色用于其后定义的图元直到再次改变颜色设定。一种新设定的颜色仅影响在其后定义的对象。

在 RGB 模式中，我们指定一种颜色的红色、绿色和蓝色分量。如 2.9 节所述，第四个颜色分量即 **α 系数**（alpha coefficient）是任选的，而四维的颜色描述称为 RGBA 颜色。第四个颜色参数用来控制图元重叠时的颜色调和。颜色调和的一个重要应用是模拟透明效果。在这样的计算中，α 的值与透明性（或不透明性）设定相对应。在 RGB（或 RGBA）模式中，使用下面的函数来选择当前颜色分量：

```
glColor* (colorComponents);
```

后缀码与 glVertex 函数中的相同。使用 3 或 4 及数值的数据类型码和一个任选的向量后缀一起指定 RGB 或 RGBA 模式。该数值数据类型是 b（字节）、i（整数）、s（短整数）、f（浮点数）和 d（双精度浮点数）以及无符号数值。颜色分量的浮点数范围从 0.0 到 1.0，glColor 包括 α 值的默认颜色分量是（1.0, 1.0, 1.0, 1.0），它将 RGB 颜色设成白色而 α 值为 1.0。作为例子，下面的语句在 RGB 模式下使用浮点值将图元当前颜色设定为青色（绿色和蓝色最高强度的混合）。

```
glColor3f (0.0, 1.0, 1.0);
```

使用数组指定三个颜色分量，则可按如下方式设定上例中的颜色：

```
glColor3fv (colorArray);
```

使用 glBegin/glEnd 函数对可为单个点位置设定选择的 OpenGL 颜色。

颜色分量的整数描述依赖于系统的性能。每像素 8 位（每分量 256 等级）的全彩色系统的整数值的范围从 0 到 255。对应的颜色分量浮点数为 0.0, 1.0/255.0, 2.0/255.0, ..., 255.0/255.0 = 1.0。在全彩色系统中，我们可以用整数值指定青色的分量：

```
glColor3i (0, 255, 255);
```

帧缓存实际上存储整数值，因此用整数指定颜色分量可避免给出浮点数必须进行的转换工作。任何格式指定的颜色值都要按比例转换成特定系统有效位所确定的范围中的整数。

4.3.2 OpenGL 颜色索引模式

OpenGL 中也可以用指向颜色表的颜色索引模式来指定颜色。在该模式下，通过指定一个指向颜色表的索引来设定当前颜色：

```
glIndex* (colorIndex);
```

参数 colorIndex 被赋予一个非负整数。该索引值被存储到随后指定的图元对应的帧缓存位置中。我们可以将索引指定为如下数据类型：无符号字节、整数和浮点数。参数 colorIndex 的数据类型由后缀码 ub、s、i、d 和 f 来指定，而颜色表的索引数总是 2 的指数，如 256 或 1024。每一表项的位数依赖于系统的硬件设施。作为索引模式的颜色指定示例，下面的语句将当前颜色索引设定为 196。

```
glIndexi (196);
```

所有在这个语句后定义的图元都被赋予颜色表该位置中存储的颜色，直到当前颜色再次改变。

在 OpenGL 核心库中没有装载颜色查找表的函数，因为表处理子程序是窗口系统的一部分。有的窗口系统支持多个颜色表，而另外一些则仅有一个颜色表及有限的选择。然而，有一个 GLUT 子程序可用来与窗口系统进行交互，从而为给定的一个索引位置指定颜色：

```
glutSetColor (index, red, green, blue);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

颜色参数 red、green 和 blue 被赋予从 0.0 到 1.0 范围中的浮点数。该颜色装入用参数 index 的值指定的表项中。

OpenGL 核心库扩充了处理另外三个颜色表的子程序。它们是 OpenGL **成像子集**（Imaging Subset）的一个部分。存在这些表中的颜色值可通过各个缓存处理以便用来修改像素值。使用这些表的例子有设定照相机的聚焦效果、从图像中过滤掉某些颜色、增强某种强度或调整亮度、将灰度照片转换成彩色以及显示的反走样等。可利用这些表来改变颜色模型，即将 RGB 颜色改变为使用另外三个“基色”（如青色、品红和黄色）的颜色描述。

OpenGL 成像子集中使用 GL_COLOR_TABLE、GL_POST_CONVOLUTION_COLOR_TABLE 或 GL_POST_COLOR_MATRIX_COLOR_TABLE 名字的特定颜色表用 glEnable 函数来激活。我们可以用成像子集中的子程序来选择特定的颜色表、设定颜色表的值、复制表的值或指定需要改变像素颜色的哪一个分量及如何改变它。

4.3.3 OpenGL 颜色调和

在许多应用中，很容易混合重叠对象的颜色或将一个对象与背景调和。这样的例子有模拟画笔效果、将两张或多张照片混合成一张、透明效果建模和场景中对象的反走样。许多图形软件包提供生成多种颜色调和效果的方法，这些函数称为**颜色调和函数**（color-blending function）或**图像混合函数**（image-compositing function）。在 OpenGL 中，通过先将第一个对象装载进帧缓存，再将第二个对象的颜色与帧缓存颜色相混合来实现两个对象颜色的调和。当前帧缓存颜色称为 OpenGL 目标颜色（destination color），而第二个对象的颜色称为 OpenGL 源颜色（source color）。调和方法仅在 RGB 或 RGBA 模式下完成。要在应用中进行颜色调和，必须先用下面的函数激活这个 OpenGL 特性。

```
glEnable (GL_BLEND);
```

使用下面的函数将关闭 OpenGL 的颜色调和子程序：

```
glDisable (GL_BLEND);
```

如果颜色调和没有被激活，则一个对象的颜色将简单地取代帧缓存中相应位置的内容。

颜色可按要达到的效果进行多种调和，通过指定两组调和因子来生成不同的颜色效果。一组调和因子针对帧缓存中的当前对象（“目标对象”），而另一组调和因子针对新来的（“源”）对象。将要装入帧缓存的新的调和颜色计算如下：

$$(S_r R_s + D_r R_d,\ S_g G_s + D_g G_d,\ S_b B_s + D_b B_d,\ S_a A_s + D_a A_d) \tag{4.1}$$

这里，RGBA 源颜色分量为(R_s, G_s, B_s, A_s)，目标颜色分量为(R_d, G_d, B_d, A_d)，源调和因子为(S_r, S_g, S_b, S_a)，而目标调和因子为(D_r, D_g, D_b, D_a)。计算出的组合颜色分量归一到 0.0 到 1.0 之间。即任何大于 1.0 的总和均设为 1.0，而任何小于 0.0 的和设为 0.0。

使用下列函数可选择调和因子的值：

```
glBlendFunc (sFactor, dFactor);
```

参数 sFactor 和 dFactor，即源和目标因子，都用 OpenGL 符号常量赋值以指定为预定义的一组四元素调和系数。例如，常量 GL_ZERO 表示调和因子(0.0,0.0,0.0,0.0)而 GL_ONE 表示(1.0, 1.0, 1.0, 1.0)。我们可以使用 GL_DST_ALPHA 或 GL_SRC_ALPHA 将四个调和因子设为目标 α 值或源 α 值。其他可用来设定调和因子的 OpenGL 常量有 GL_ONE_MINUS_DST_ALPHA、GL_ONE_MINUS_SRC_ALPHA、GL_DST_COLOR 和 GL_SRC_COLOR。这些调和因子常用于模

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

拟透明性，这将在10.19节详细讨论。sFactor的默认值是GL_ONE，而dFactor的默认值是GL_ZERO。因此，这两组调和因子的默认值将导致新来的颜色值取代帧缓存中的当前颜色值。

包含在OpenGL扩充库中的附加函数称为成像子集。这些子程序中包含了设定调和颜色的子程序和指定调和等式的子程序。

4.3.4 OpenGL 颜色数组

我们也可以在顶点数组中和坐标值混合一起指定场景的颜色值（参见3.17节）。这既可在RGB模式下也可在RGBA模式下进行。与顶点数组一样，必须先激活OpenGL的颜色数组：

```
glEnableClientState (GL_COLOR_ARRAY);
```

然后，对RGB模式要指定颜色分量的位置和格式：

```
glColorPointer (nColorComponents, dataType, offset, colorArray);
```

参数nColorComponents赋值为3或4，依赖于是否在数组colorArray中列出RGB或RGBA颜色分量。OpenGL符号常量如GL_INT或GL_FLOAT赋给参数dataType来指向颜色值的数据类型。对于一个单独的颜色数组，我们可将0赋给参数offset。但是如果将颜色数据和顶点数据组合在同一个数组中，则offset的值是数组中每一组颜色分量的字节数。

我们可以修改3.17节中的顶点数组例子使之包含颜色数组，从而给出使用颜色数组的例子。下面的程序段设定立方体前向面所有顶点的颜色为蓝色，设定后向面所有顶点的颜色为红色。

```
typedef GLint vertex3 [3], color3 [3];

vertex3 pt [8] = { {0, 0, 0}, {0, 1, 0}, {1, 0, 0},
    {1, 1, 0}, {0, 0, 1}, {0, 1, 1}, {1, 0, 1}, {1, 1, 1} };
color3 hue [8] = { {1, 0, 0}, {1, 0, 0}, {0, 0, 1},
    {0, 0, 1}, {1, 0, 0}, {1, 0, 0}, {0, 0, 1}, {0, 0, 1} };

glEnableClientState (GL_VERTEX_ARRAY);
glEnableClientState (GL_COLOR_ARRAY);

glVertexPointer (3, GL_INT, 0, pt);
glColorPointer (3, GL_INT, 0, hue);
```

我们可以将颜色和顶点坐标一起装入一个**交错数组**（interlaced array）。每一个指针用适当的位移值指向单一的交错数组。例如：

```
static GLint hueAndPt [ ] =
    {1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
     0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0,
     1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
     0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1};

glVertexPointer (3, GL_INT, 6*sizeof(GLint), hueAndPt [3]);
glColorPointer (3, GL_INT, 6*sizeof(GLint), hueAndPt [0]);
```

该数组的前三个元素指定一个RGB颜色值，接下来的三个元素指定一个顶点坐标(x, y, z)，这样交错下去直到最后。将offset设定为相邻颜色或顶点值之间的字节数，即两者均为6*sizeof(GLint)。颜色值从交错数组的第一个元素即hueAndPt[0]开始，顶点值从第四个元素即hueAndPt[3]开始。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

因为一个场景通常包括若干个对象，每个对象有多个平表面，OpenGL提供一个可以一次性指定所有顶点和颜色数组及其他类型信息的函数。如果我们要将上例中的颜色和顶点值改为浮点数，则需按照下面的格式使用该函数：

```
glInterleavedArrays (GL_C3F_V3F, 0, hueAndPt);
```

第一个参数是一个OpenGL常量，用来指定颜色（C）和顶点（V）的三元素浮点描述。而数组hueAndPt按每个顶点的颜色放在其坐标前的方式交错。该函数也自动地激活顶点和颜色数组。

在颜色索引模式下，使用下面的语句定义一个颜色索引数组：

```
glIndexPointer (type, stride, colorIndex);
```

颜色索引在数组 colorIndex 中列出，而参数 type 和 stride 与 glColorPointer 中的相同。由于颜色表索引用单个值描述，所以不需要 size 参数。

4.3.5　其他 OpenGL 颜色函数

在2.9节的第一个编程例子中，我们介绍了下面这个用来为显示窗口选择RGB颜色分量的函数。

```
glClearColor (red, green, blue, alpha);
```

(red, green, blue)中的每一分量以及 α 参数均赋以 0.0 到 1.0 范围中的浮点数值。四个参数的默认值都是0.0，它们生成黑色。如果每一个颜色分量置为 1.0，则该净颜色是白色。颜色分量为0.0到1.0之间的相等值时得到各种灰色。第四个参数α提供对前面颜色和当前颜色调和的选项。这仅仅在激活了 OpenGL 的调和特性后才会发生；颜色调和不可能作用于颜色表指定的值。

如同我们在3.19节中所指出，OpenGL中有几个颜色缓存可用做显示场景的当前刷新缓存，而函数 glClearColor 指定所有颜色缓存的颜色。然后用下面的命令将净颜色用于这些颜色缓存：

```
glClear (GL_COLOR_BUFFER_BIT);
```

我们也可以用 glClear 函数设定 OpenGL 中有效的其他缓存的初始值。它们是存放调和颜色信息的累计缓存（accumulation buffer）、存放场景对象深度值（离观察位置的距离）的深度缓存（depth buffer）及存放定义图形范围的模板缓存（stencil buffer）。

在颜色索引模式下，使用下面的函数（而不是 glClearColor）设定显示窗口颜色。

```
glClearIndex (index);
```

窗口背景色用存放在颜色表中 index 位置的颜色来指定。执行 glClear (GL_COLOR_BUFFER_BIT)函数后窗口就以该颜色显示。

OpenGL 库中有另外一些颜色函数用来处理各种任务，如改变颜色模式、设定场景光照效果、描述照相机效果及绘制对象表面。我们在分析计算机图形系统的各种处理时将检查其他的颜色函数。但现在，我们把讨论限制在与图元的颜色描述有关的那些函数范围内。

4.4　点的属性

一般情况下，我们可以设定点的两个属性：颜色和大小。在一个描述系统中，点的显示颜色和大小由存放在属性表中的当前值确定。颜色分量用RGB值或指向颜色表的索引值设定。对于光栅系统而言，点的大小是像素大小的一个整倍数，因此一个大的点显示成一个像素方块。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

4.5 线的属性

直线段可以使用三个基本属性来显示：颜色、线宽和线型。线的颜色用对所有图元相同的函数进行设定，而线宽和线型则用单独的线函数选择。另外，线还可生成如画笔和笔刷等其他效果。

4.5.1 线宽

线宽选择的实现取决于输出设备的能力。在视频监视器上的粗线可以用相邻的平行线进行显示，而在笔式绘图仪上则可能需要更换画笔来绘制粗线。

在光栅实现中，类似于 Bresenham 算法，通过在每个采样位置处使用一个像素来生成标准线宽。其他线宽则是作为标准线的正整数倍，通过沿相邻平行线路径绘制额外的像素而显示的。对于斜率绝对值小于 1.0 的直线，可以修改画线程序，通过在沿线的每个列（x 位置）绘制像素的垂直段来显示粗线。每段的像素数目等于线段的整数值。在图 4.2 中，我们通过在原线路径上面生成一条平行线而画出了双倍线宽的直线。在每个取样位置 x 处，计算对应的 y 坐标并用屏幕坐标(x, y)和$(x, y+1)$绘制像素。通过交替地在单宽度线路径上下绘制像素，可以显示宽度为 3 或更宽的直线段。

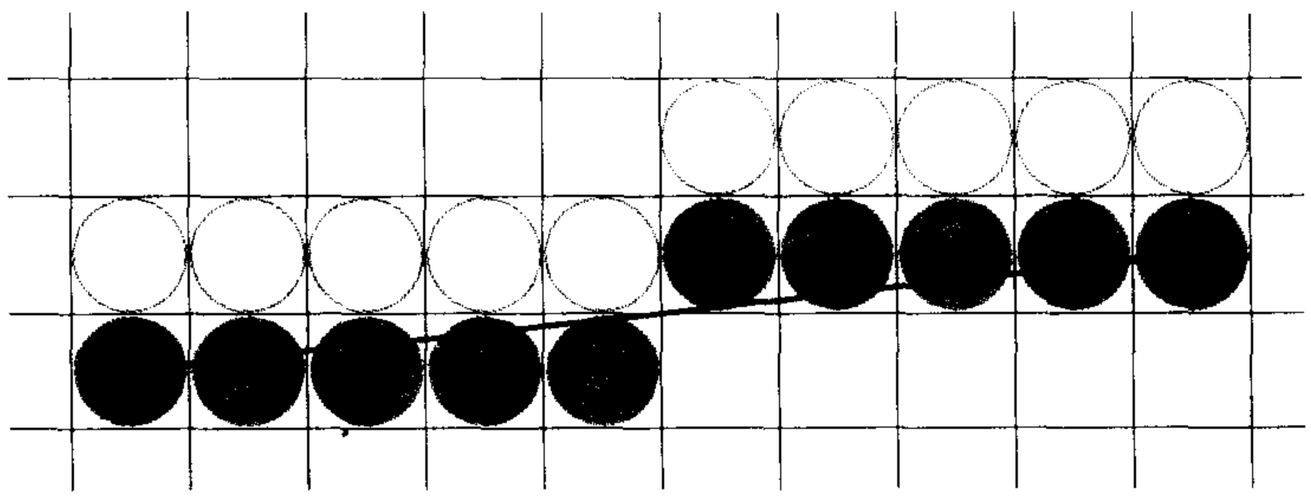


图 4.2 利用垂直像素段生成斜率 $|m| < 1.0$ 的双倍光栅线

对于斜率绝对值大于1.0的直线，可以交替地选择线段路径左边和右边的像素以形成水平段来显示粗线。图 4.3 示例了这个方法，其线宽为 4，使用水平像素段进行绘制。类似地，斜率小于或等于 1.0 的粗线可以用垂直段显示。通过比较线端点水平方向和垂直方向的分离程度（Δx 和 Δy），可以实现这一过程。如果 $|\Delta x| \geq |\Delta y|$，则像素在垂直方向重复。否则，在每一行画出多个像素。

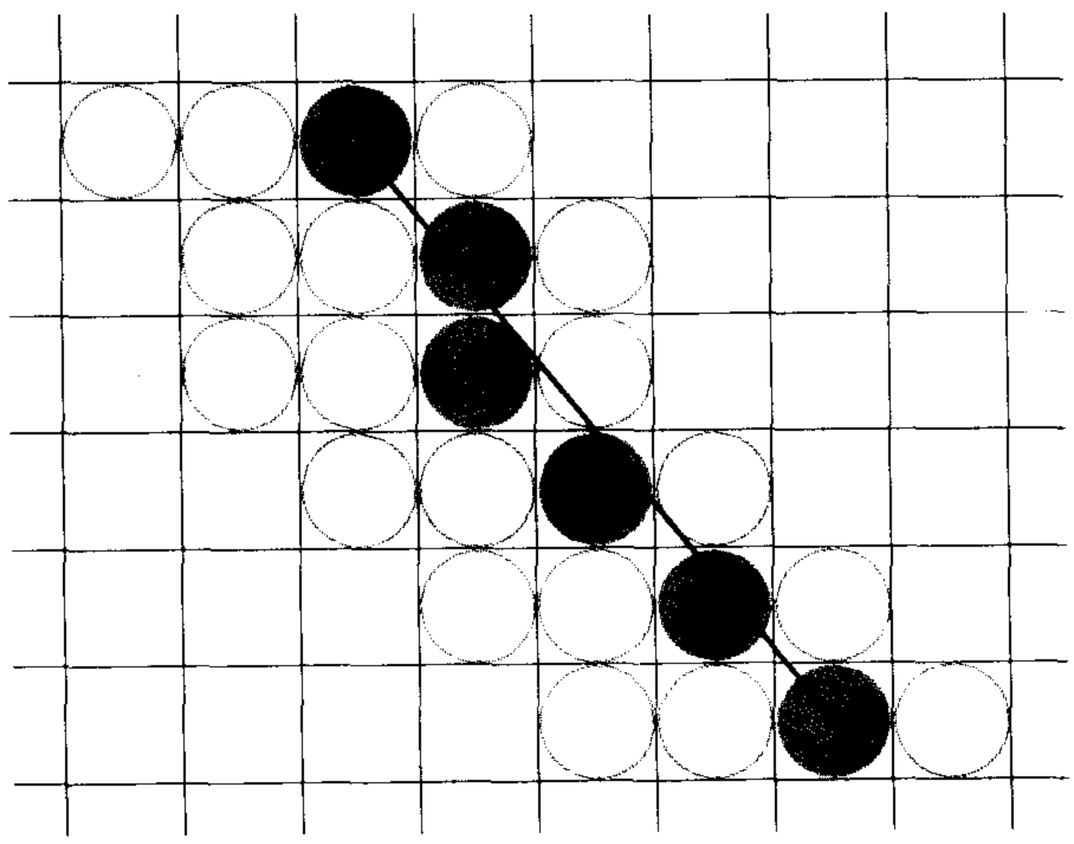


图 4.3 利用水平像素段绘制斜率 $|m| > 1.0$ 且线宽为 4 的光栅线

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

尽管绘制水平或垂直段的方法可以快速地生成粗线，但是显示的线宽（在线路径垂直方向测量）依赖于它的斜率。45° 线的宽度是用等长像素段绘制的水平或垂直线条宽度的 $1/\sqrt{2}$。

使用水平或垂直像素段实现宽度选择的另一个问题是：无论斜率大小，所生成直线的端点是水平的或是垂直的。这对于较粗的直线的影响则更为突出。我们可以通过添加**线帽**（line cap）来调整线端的形状，从而给出更好的外观（参见图 4.4）。线帽的一种形式是方帽（butt cap），这种形式通过调整所构成的平行线的端点位置，使粗线的显示具有垂直于线路径的方端。假如指定直线的斜率为 m，那么粗线的方端的斜率为 $-1/m$。另一种线帽是圆帽（round cap），这种形式通过对每个方帽添加一个填充的半圆而得到。圆弧的圆心在线段的端点，其直径与直线宽度相等。第三种线帽是突方帽（projecting square cap）。这里，我们简单地将线段向两头延伸一半线宽并添加方帽。

生成粗线的其他方法包括将显示线段看做填充的矩形，或用选定的画笔和笔刷图案来生成线段，这些内容将在下一节讨论。为了得到线段边界的矩形表示，可以沿垂直于线路径方向计算矩形顶点的位置，从而使顶点坐标与线段端点的距离为线宽的一半。那么，矩形线就可以显示为图 4.4(a)。然后，可以给填充的矩形添加圆帽或延伸其长度来显示突方帽。

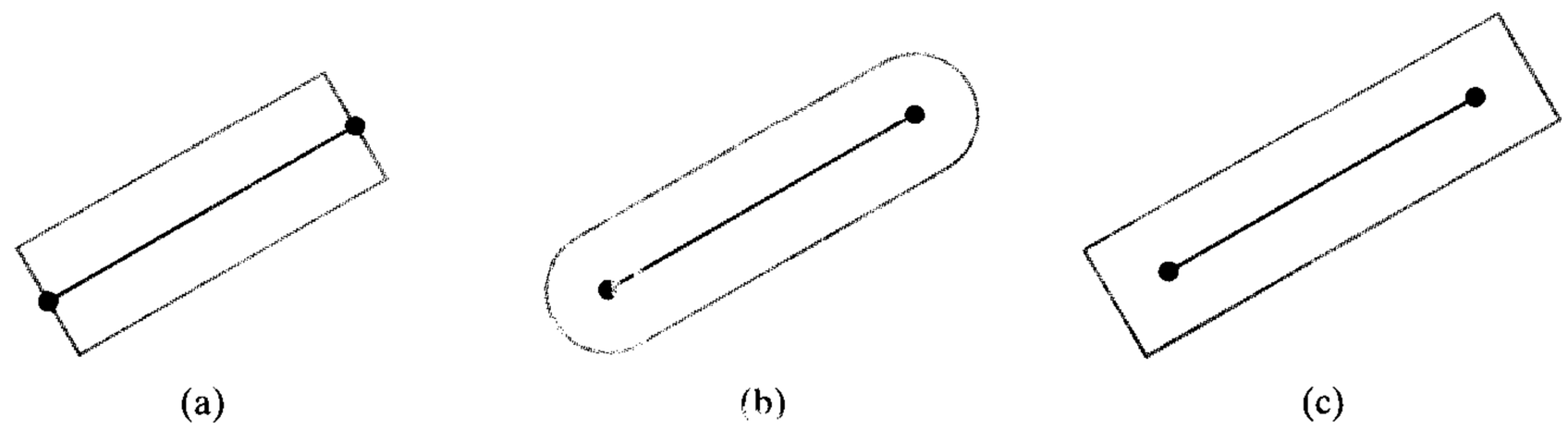


图 4.4　具有(a)方帽、(b)圆帽和(c)突方帽的粗线

生成粗折线需要一些额外的考虑。通常，显示单根线段所用的方法不能生成平滑连接的一系列线段。例如，使用水平或垂直像素段显示粗线，会在不同斜率的线段的连接处，其水平段变成垂直段时留下间隙。我们可以通过在线段端点进行额外的处理来生成平滑连接的粗折线。图4.5给出了两线段平滑连接的三种可能方法。斜角连接（miter join）通过延伸两条线的外边界直到它们相交而形成；圆连接（round join）通过使用直径等于线宽的圆弧边界将两线段连接而形成；斜切连接（bevel join）则是通过使用方帽并在两线段相交处的三角形间隙中进行填充而形成。假如两连接线段间的夹角很小，斜角连接会产生一个较长的尖峰而使折线变形。图形软件包可以在任何连接的两线段于足够小的角度相交时，切换到斜切连接来避免这种情况。

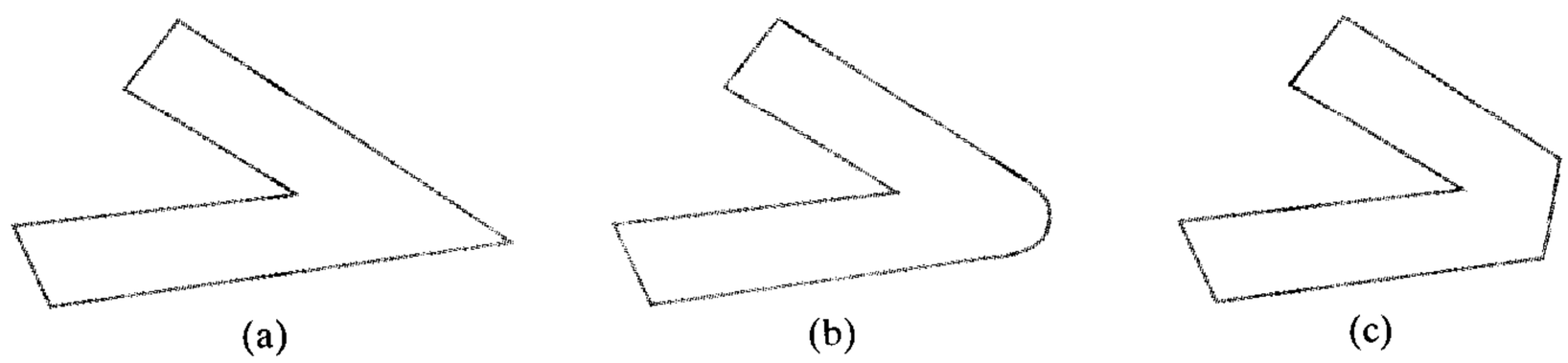


图 4.5　粗线的连接：(a)斜角连接；(b)圆连接；(c)斜切连接

4.5.2　线型

可选用的线型属性包括实线、虚线和点线等。通过设置沿线路径显示的实线线段的长度和间距来修改画线算法，可以生成各种类型的线。可以通过在实线线段之间插入与实线线段等长的空白段来显示虚线，许多图形系统都允许选择划线长度和划线间隔的长度。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

光栅线算法通过绘制像素段来显示线型属性。对于各种划线、点线和点划线样式，画线程序沿线路径输出一些连续像素段。在每两个实心段之间有一个给定长度的空白间隔段，段长度和中间空白段的像素数目可用像素**掩模**（pixel mask）指定。像素掩模是包含数字0和1的字符串，用来指出沿线路径需要绘制哪些位置。例如，掩模11111000可用来显示划线长度为5个像素和间隔空白段为3个像素的虚线。与1对应的像素位置赋以当前颜色，而与0对应的像素位置显示背景色。

使用固定数目的像素来绘制划线会产生如图4.6所示的、在不同的直线方向生成不等长划线的现象。图中所显示的线段都是用4个像素画出，但对角线上的划线要比水平方向长$\sqrt{2}$倍。如果要进行精确的绘制，那么对任何直线方向的划线长度应保持近似的相等。为了实现这一点，需要按照直线的斜率来调整实心段和中间空白段的像素数目。在图4.6中，我们可以通过将对角线的划线像素减少到3个来显示出近似等长的划线。另一个保持划线等长的方法是，将划线看成单独的线段。将每条划线的端点坐标进行定位后，调用沿划线路径计算像素位置的画线程序。

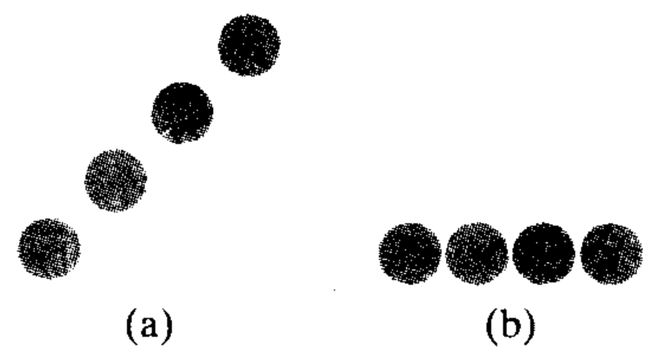


图4.6 用相同数目像素显示的不等长划线

4.5.3 画笔或画刷的选择

在有些图形软件包中，尤其是绘画系统，可以直接选择不同的画笔和画刷类型。这种类型的选项包括形状、尺寸和画笔或画刷的图案。图4.7给出了一些可能的画笔和画刷形状。这些形状可以按像素位置的数值形式存储在一个像素模板中，然后再设置到线路径上。例如，矩形画笔可用图4.8给出的模板，通过将模板的中心（或一角）沿线路径移动而实现，如图4.9所示。为了避免在帧缓存中重复设置像素，可以简单地累计在模板的每个位置上生成的水平段，并沿每一条扫描线跟踪起始和终止的x位置。

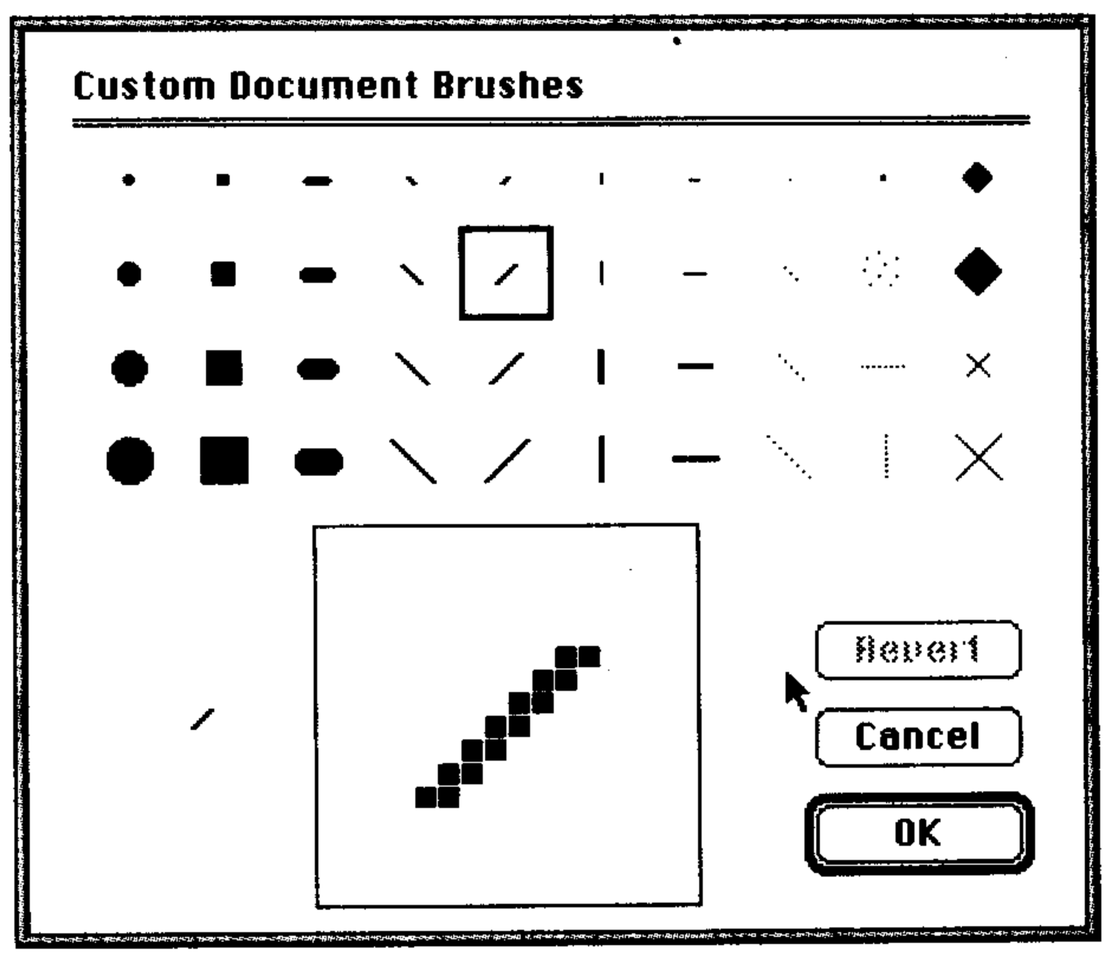


图4.7 显示线的画笔或画刷形状

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

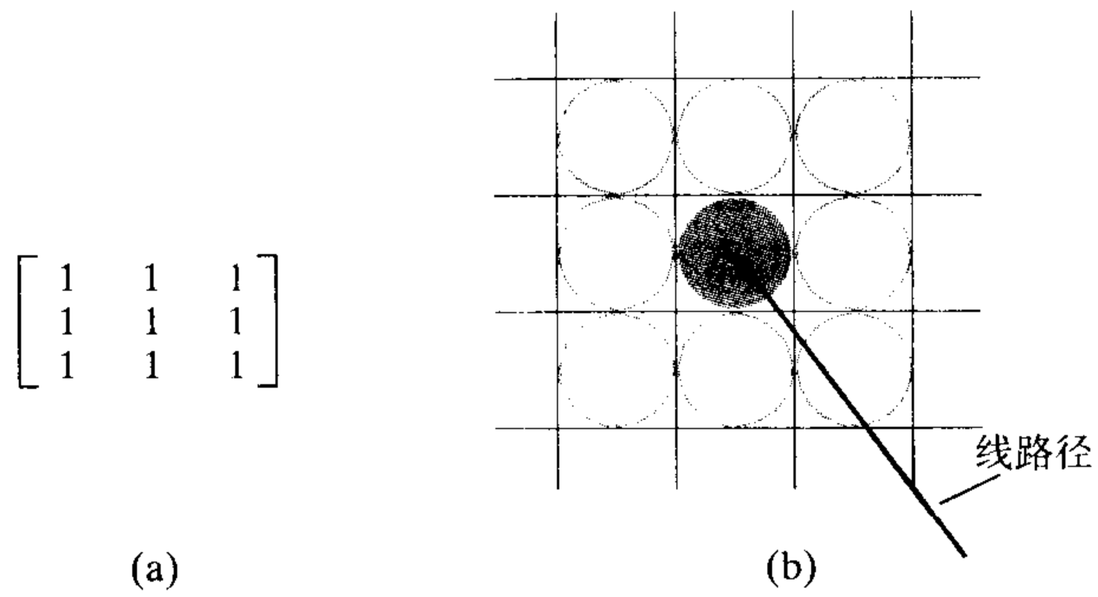


图 4.8　(a)矩形边的像素掩模及相关的像素数组；(b)通过将掩模居中放在指定像素位置的显示

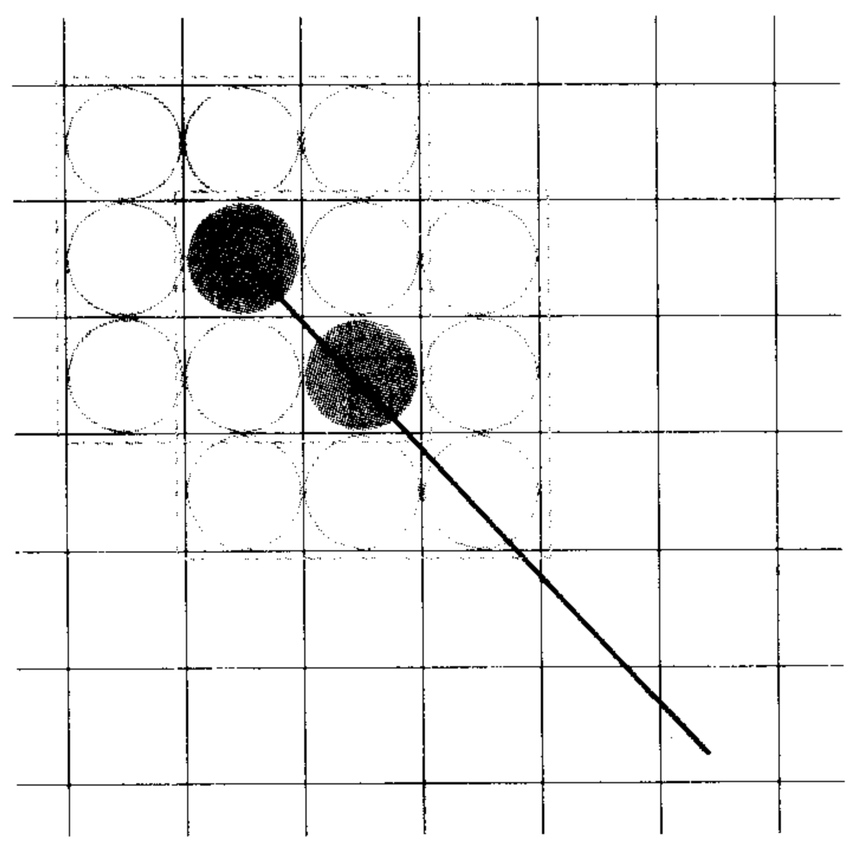


图 4.9　使用图 4.8 中的画笔形状生成直线

通过改变模板的尺寸，使用画笔（或画刷）形状生成的直线可以有多种宽度。例如，图 4.9 中的矩形笔线可以收缩成 2 × 2 矩形模板或放宽成 4 × 4 模板。通过将图案值加到画笔和画刷的标记上，也可以按选定的图案来显示线段。

4.6　曲线属性

曲线属性的参数与线段相同，可以使用各种颜色、宽度、点划线模式和有效的画笔和笔刷选择来显示曲线。采用画曲线算法来实现属性选择，这一点类似于画直线。

各种宽度的光栅曲线可用水平或垂直像素段进行显示。曲线斜率的绝对值小于 1.0 时，使用垂直段；斜率绝对值大于 1.0 时绘制水平段。图 4.10 给出了使用这种方法显示在第一象限中宽度为 4 的圆弧。利用圆的对称性，在从 $x = 0$ 到 $x = y$ 的八分象限内使用垂直段生成圆路径。然后对于直线 $y = x$，将像素位置反射便得到所显示曲线的余下部分。其他四分象限内的圆弧段，可通过对坐标轴反射第一象限中的像素位置而得到。使用这种方法显示的曲线粗细则是根据曲线斜率的函数，圆、椭圆和其他曲线上斜率绝对值为 1 的位置将显得最细。

另一种显示粗曲线的方法是填充两条距离等于预定宽度的平行曲线路径间的区域。我们可以用指定的曲线路径作为第一条边界，并在其内侧或外侧建立第二条边界。然而，这种方法使得原始曲线路径按所选的第二条边界向内或向外偏移。我们可以通过在指定的曲线路径两侧以宽度的一半为距离，设置两条边界曲线来保持原曲线位置。图 4.11 给出了这种方法的一个例子，图中的圆弧半径

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

为16，线宽为4。然后，在半径为16、两侧距离为2处设置两条边界圆弧。为了保持在3.13节讨论的圆弧的合适尺寸，可以将同心边界圆弧的半径设为 $r = 14$ 和 $r = 17$。尽管这种方法对于生成粗线圆是精确的，但通常该方法仅提供对其他粗曲线的真正区域的一个近似范围。例如，使用这种方法生成的扁椭圆，其内部和外部边界的焦点不是相同的。

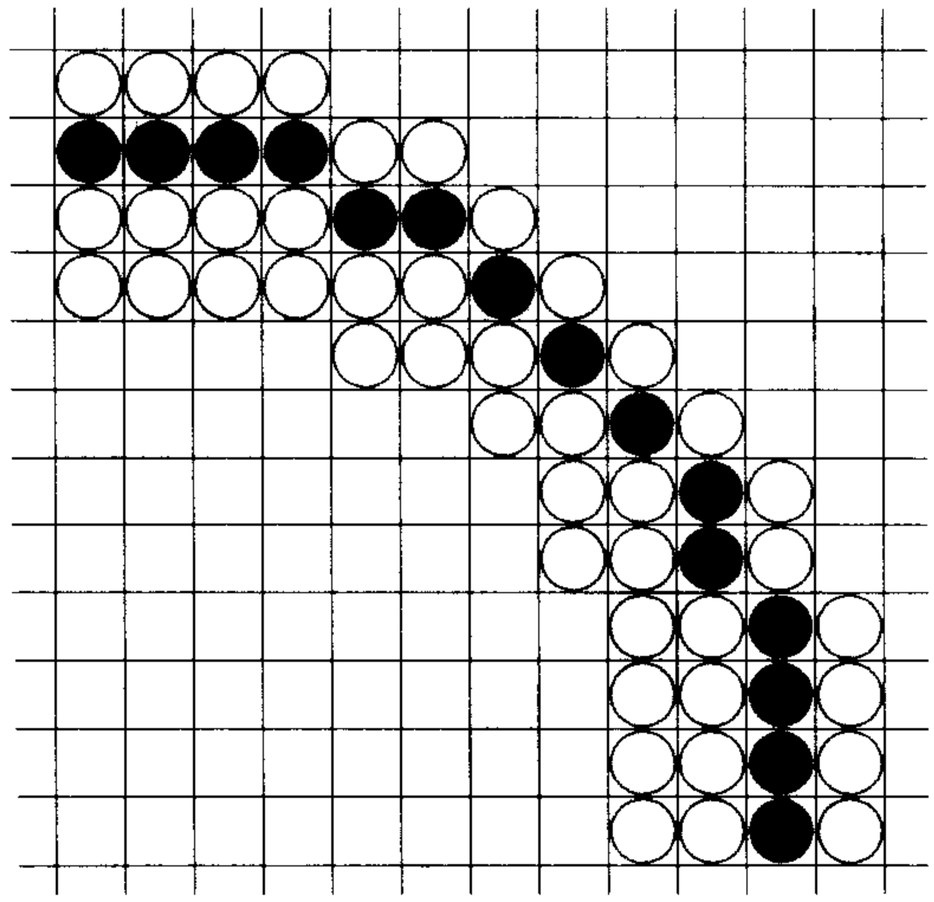


图4.10 按斜率使用水平或垂直像素段绘制的宽度为4的圆弧

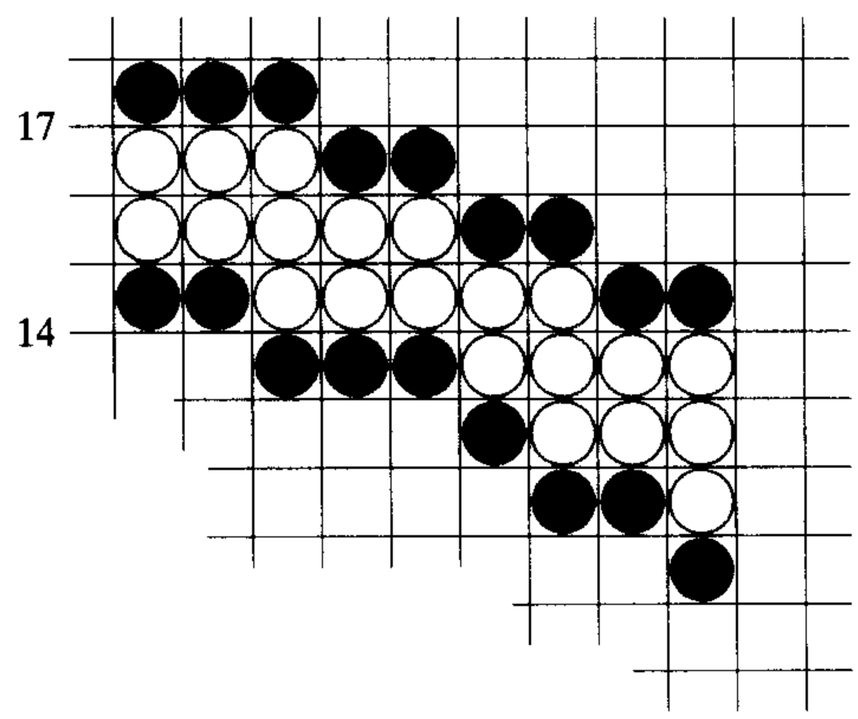


图4.11 通过填充两同心圆弧间的区域而显示的宽度为4、半径为16的圆弧

为实现线型而提出的像素掩模方法也可用于光栅曲线中以生成短划式或点式图案。例如，掩模11100生成图4.12中短划式的弧。我们可以利用对称性在各个八分象限中生成短划，但必须移动像素位置以保持跨越每一八分象限时有正确的短划和间隔序列。和直线段算法一样，像素掩模显示的短划和间隔的长度按曲线的斜率而改变。如果我们要显示定长的短划，则需要在沿圆周移动时调整每段绘制的像素数目。除使用像素掩模保持均匀段长外，还可以沿同角度的弧来生成等长短划。

曲线的画笔（或画刷）显示可以使用在线段中讨论的相同技术来生成。在图4.13中，我们沿线路径重复画笔的形状，从而生成第一象限内的圆弧。这里，矩形画笔的中心移向后继曲线位置，从而产生显示的曲线形状。当曲线斜率为1时，以这种用矩形画笔显示的曲线会比较粗。如果要显示一致的曲线宽度，可以通过旋转矩形画笔，使其在沿曲线移动时与斜率方向一致，或用圆弧画笔形状来实现。使用画笔和画刷形状绘制的曲线，可以使用不同大小和附加的图案或模拟的绘画技巧进行显示。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

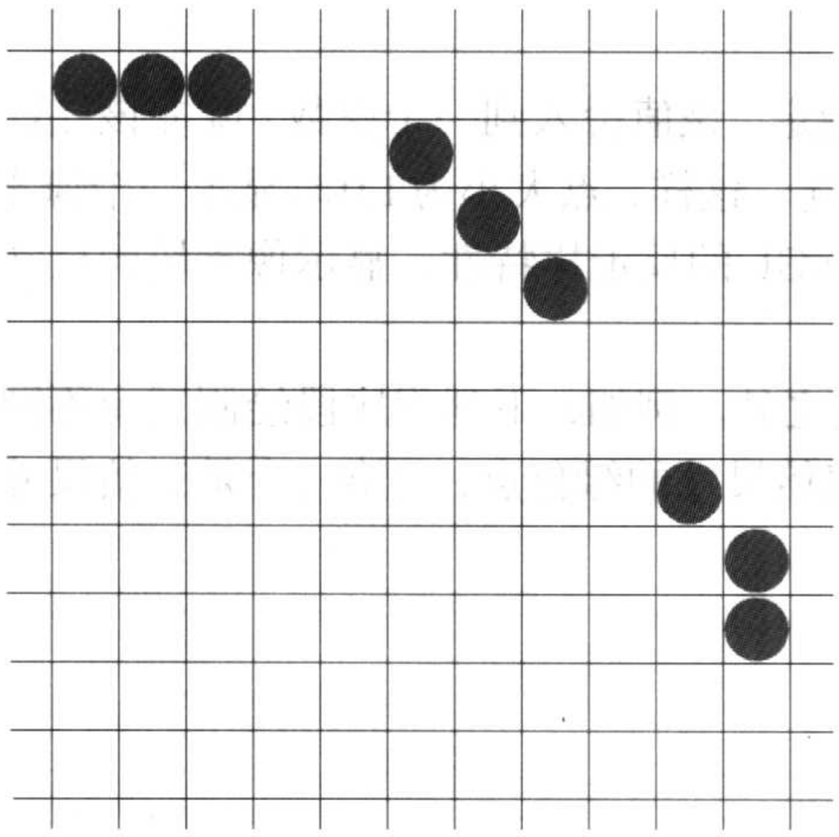


图 4.12　用 3 点短划及 2 点间隔显示的短划圆弧

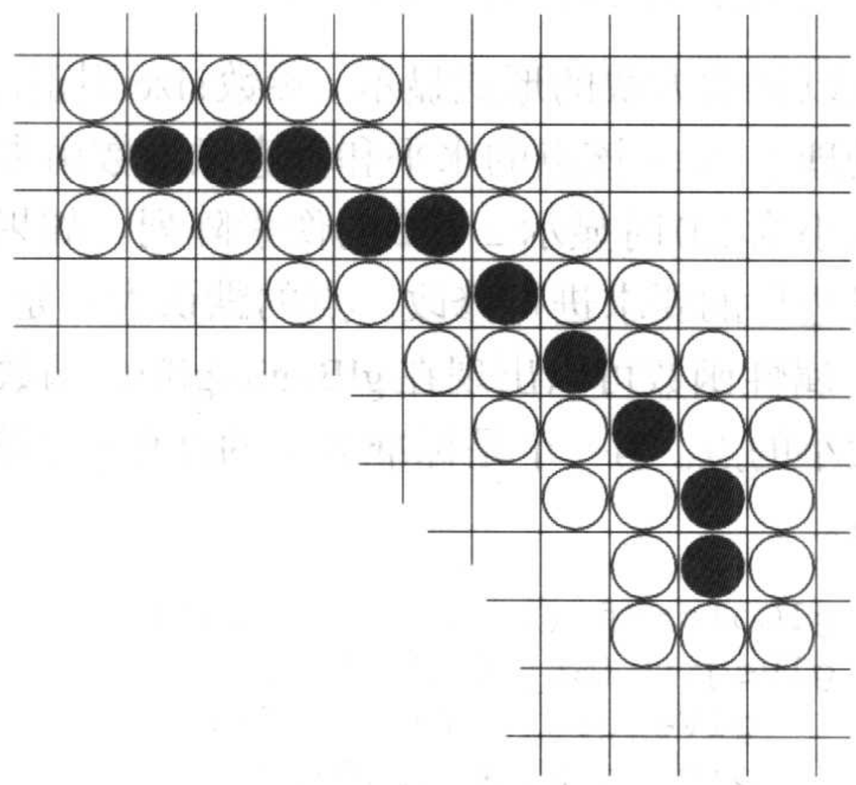


图 4.13　使用矩形画笔来显示的圆弧

绘画程序提供了交互方式，可以使用触笔和图形板等指点设备来构造图形。图 4.14 给出了这类曲线图案的例子。绘画软件中可另外提供模拟毛笔笔划显示的选项。图4.15给出了通过对不同类型毛笔笔划建模而生成的某些图案。

图 4.14　绘画程序用各种形状和图案所画的曲线。从左到右的画刷形状为：方、圆、对角线、点阵和渐变喷刷

图 4.15　计算机艺术家 Koichi Kozaki 用画笔系统绘制的日本幸运标志 daruma 洋娃娃，实际上一开始它没有眼睛，在许愿时喷绘一只眼，在愿望实现时喷绘另一只（Wacom Technology 公司提供）

4.7　OpenGL 点属性函数

指定点位置的显示颜色由状态表中的当前颜色值控制。而该颜色用 glColor 函数或 glIndex 函数来指定。

我们使用下面的函数来指定 OpenGL 中一个点的大小：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glPointSize (size);
```

该点以像素方块的形式显示。参数size用正浮点数值指定，该值舍入到一个整数（除非该点经反走样处理）。显示该点的水平和垂直像素数由参数size决定。这样，点大小为1.0时显示一个像素，而点大小为2.0时显示2 × 2的像素阵列。如果激活OpenGL的反走样特性，显示像素块的尺寸将按平滑边界的要求进行修改。点的默认大小是1.0。

属性函数可以出现在glBegin/glEnd函数对之内或之外。例如，下列程序段绘制三个不同颜色和大小的点。第一个是标准大小的红色点，第二个是双倍尺寸的绿色点，而第三个是三倍尺寸的蓝色点。

```
glColor3f (1.0, 0.0, 0.0);
glBegin (GL_POINTS);
   glVertex2i (50, 100);
   glPointSize (2.0);
   glColor3f (0.0, 1.0, 0.0);
   glVertex2i (75, 150);
   glPointSize (3.0);
   glColor3f (0.0, 0.0, 1.0);
   glVertex2i (100, 200);
glEnd ( );
```

4.8 OpenGL线属性函数

可以用颜色、线宽和线型等三个属性设定来控制OpenGL直线段的外在表示。我们已经看到了如何选择颜色，OpenGL还提供了一个设定线宽的函数和另外一个设定短划或点线等线型的函数。

4.8.1 OpenGL线宽函数

使用下面的OpenGL函数可以设定线宽：

```
glLineWidth (width);
```

参数width赋以实数，该值舍入到最近的非负整数。如果输入值舍入到0.0，则线段用默认的标准宽度1.0显示。然而，在对线段进行反走样时，其边界进行光滑处理以减少阶梯现象，因而有可能出现小数宽度。有些线宽函数的实现仅支持有限的几种线宽，其他一些则不支持1.0以外的线宽。

OpenGL的线宽函数用4.5节的方法实现。即通过比较线段端点的水平和垂直距离Δx和Δy来确定使用垂直段或水平段来生成粗线段。

4.8.2 OpenGL线型函数

默认状态下，直线段显示成实线。但也可以显示划线、点线或短划和点混合的线段。还可改变短划及短划或点之间的长度。可以利用下面的OpenGL函数设定当前线型：

```
glLineStripple (repeatFactor, pattern);
```

参数pattern用来引入描述如何显示线段的一个16位整数。值为1的位对应一个“开”像素，值为0的位对应一个“关”像素。该模式从低位开始应用于线路径。默认模式为0xFFFF（每一位的值均为1），它生成实线。整数参数repeatFactor说明模式中每一位重复应用多少次才轮到下一位。默认的重复值是1。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于一条折线来说，指定的线型模式并非在每一线段的始端重新开始。它从折线的第一个端点开始，到最后一条线段的最终一个端点结束，连续地应用于折线中的所有线段。

作为指定一个线型的例子，假定参数 pattern 赋予了十六进制数 0x00FF 而重复因子为 1。这将显示一条由 8 像素短划和 8 像素短划间隔的划线。同样，由于先应用低位值，线段从始端开始先显示一个 8 像素短划。接着是一个 8 像素间隔，然后再显示一个 8 像素短划，如此直到第二个端点。

在使用当前线型显示线段之前，必须先激活 OpenGL 的线型特性。下面的函数可实现这一点。

```
glEnable (GL_LINE_STIPPLE);
```

如果忘记使用这一激活函数，则显示实线；即使用默认模式 0xFFFF 显示线段。在任何时候都可以使用下列函数来关闭线型特性：

```
glDisable (GL_LINE_STIPPLE);
```

该函数使用默认模式（实线）取代当前线型。

在下面的程序中，通过绘制三条不同线型和线宽的线段来展示 OpenGL 线属性函数的应用。图 4.16 给出了可由该程序生成的数据图。

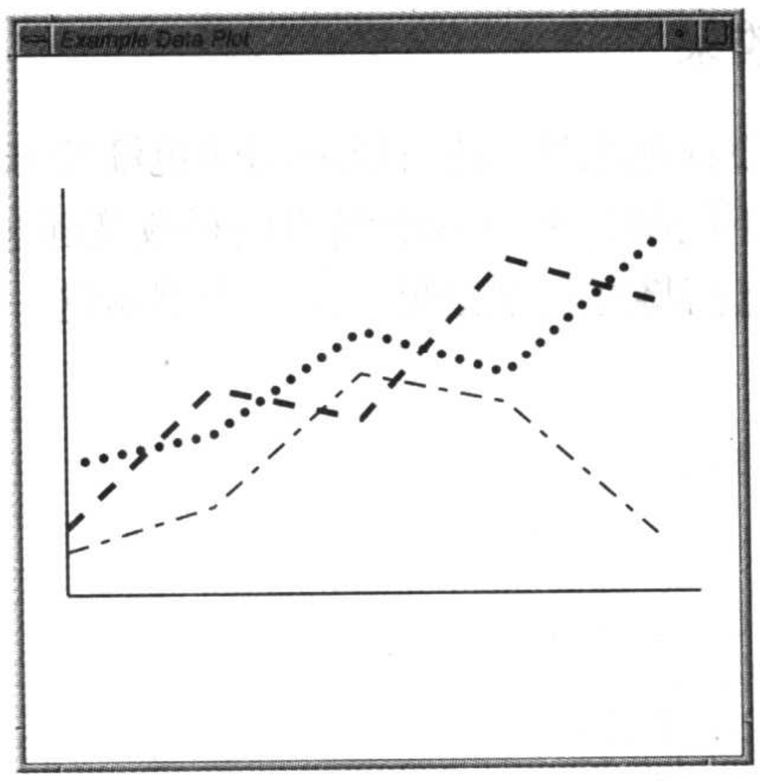


图 4.16　使用不同的 OpenGL 线型和线宽绘制三个数据集：单宽度划 - 点模式、双倍宽短划模式和三倍宽度点模式

```
/* Define a two-dimensional world-coordinate data type. */
typedef struct { float x, y; } wcPt2D;

wcPt2D dataPts [5];

void linePlot (wcPt2D dataPts [5])
{
   int k;

   glBegin (GL_LINE_STRIP);
      for (k = 0; k < 5; k++)
         glVertex2f (dataPts [k].x, dataPts [k].y);

   glFlush ( );

   glEnd ( );
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
/* Invoke a procedure here to draw coordinate axes.  */

glEnable (GL_LINE_STIPPLE);

/* Input first set of (x, y) data values. */
glLineStipple (1, 0x1C47);    // Plot a dash-dot, standard-width polyline.
linePlot (dataPts);

/* Input second set of (x, y) data values. */
glLineStipple (1, 0x00FF);    // Plot a dashed, double-width polyline.
glLineWidth (2.0);
linePlot (dataPts);

/* Input third set of (x, y) data values. */
glLineStipple (1, 0x0101);    // Plot a dotted, triple-width polyline.
glLineWidth (3.0);
linePlot (dataPts);

glDisable (GL_LINE_STIPPLE);
```

4.8.3 其他 OpenGL 线效果

除了指定线宽、线型和实心颜色之外，还可以使用颜色渐变来显示线段。例如，我们可以在定义线段时通过为每一端点赋以不同颜色来显示沿线段路径改变颜色的实线。在下列程序段中，我们通过将蓝色赋给线段的一个端点而将红色赋给另一端点来展示这一点。线段按两端点颜色的线性插值方式进行显示。

```
glShadeModel (GL_SMOOTH);

glBegin (GL_LINES);
   glColor3f (0.0, 0.0, 1.0);
   glVertex2i (50, 50);
   glColor3f (1.0, 0.0, 0.0);
   glVertex2i (250, 250);
glEnd ( );
```

函数 glShadeModel 可以有另一个变量 GL_FLAT。在这种情况下，线段用单一颜色显示：第二个点(250,250)的颜色，即显示一条红色线段。实际上，GL_SMOOTH 是默认，因此，即使未使用该函数也会显示一条均匀颜色插值的线段。

我们还可以通过用不同的颜色和线型显示相邻段来产生另外的效果。还可以通过用不同的α值添加到线段或其他对象中来使用OpenGL的颜色调和特性。使用像素阵列和颜色调和可模拟毛笔笔划和其他画笔效果。像素阵列可通过交互地移动来生成线段。像素阵列中的单个像素可赋予不同的α值，以显示画刷或笔划式线段。

4.9 填充区属性

多数图形软件包将填充区限定为多边形，因为它们用线性方程来描述。更进一步的限制是要求填充区是一个凸多边形，因此扫描线不会与两条以上的边相交。然而，我们一般可以填充任意指定的区域，包括圆、椭圆和其他有曲线边界的对象。而像绘画程序等应用系统则提供针对任意形状区域的填充功能。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在将填充区映射到像素坐标后，光栅系统中填充一个区域有两个基本过程。一个过程先确定穿过区域的扫描线重叠段。然后，沿这些重叠段的位置被设定为填充色。另一个区域填充方法是从一个给定内点开始，逐个像素向外“绘画”，直到碰到指定的边界条件。扫描线方法常用于简单的形状如圆或多边形边界的区域，而一般的图形软件包都使用这一填充方法。使用起始内点的填充方法用于有较复杂边界的填充区及交互式绘画系统。

4.9.1　填充模式

一般的图形软件包提供的基本填充属性是内部的显示模式。我们可以将一区域显示为单一颜色、指定填充图案或只给出边界的“空心”模式。图 4.17 给出了这三种模式。我们也可以使用各种笔刷模式、颜色调和或纹理对场景中的指定区域进行填充。其他选项包括指定填充区域边界。对多边形来说，可以使用不同颜色、线宽和线型来给出其边。还可以为区域的前向面和后向面选择不同的显示属性。

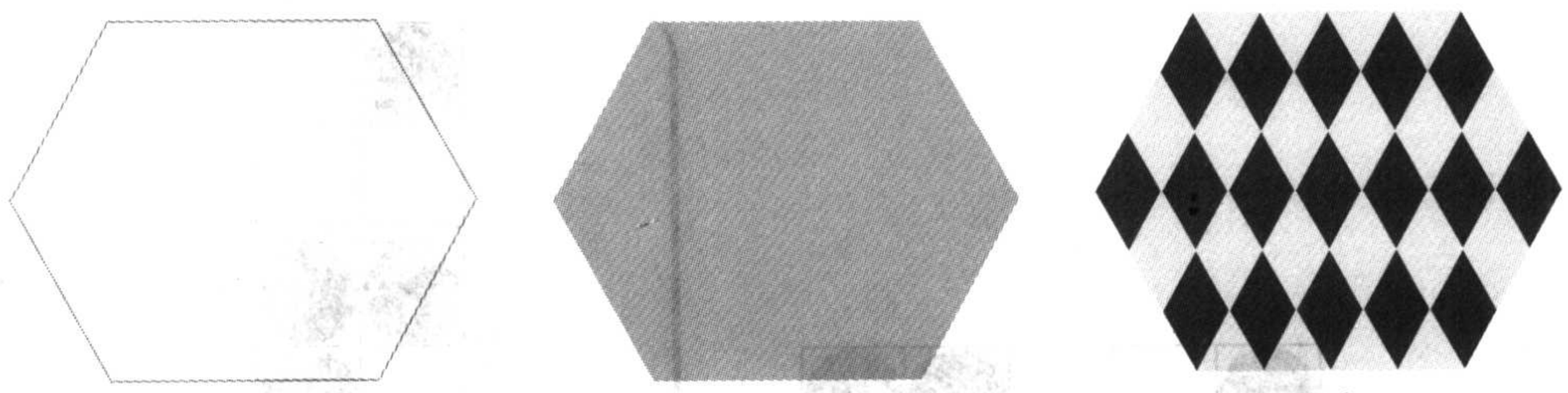


图 4.17　基本的多边形填充模式

填充图案可以用一个为不同位置指定不同颜色的矩形颜色阵列来给出。或者说，一个填充图案可当做一个点阵来指定，其中每一位置指定显示一个选定颜色。描述填充图案的矩阵是一个应用于显示区域的掩模。有些图形系统提供对覆盖掩模的初始位置的选择。从该初始位置开始，掩模在水平和垂直方向反复填充直到所有显示区域都填满了无重叠的掩模。在使用图案覆盖的填充区，矩阵掩模指出哪些像素应该使用特定颜色显示。这种用矩形图案填充一个区域的处理称为**平铺**（tiling），而矩形填充图有时称为**平铺图案**（tiling pattern）。有时，系统中有预定义的填充图案，如图4.18所示的影线。

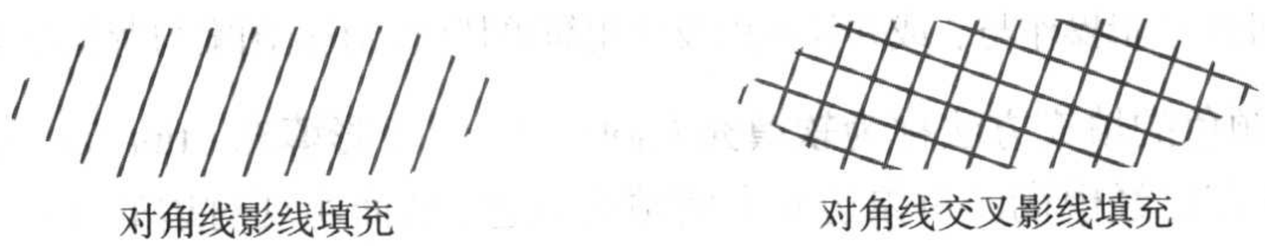


图 4.18　使用影线模式的区域填充

可以通过确定图案在何处覆盖穿过填充区的扫描线来实现图案填充。从一个为图案填充指定的起始位置出发，将矩形图案按在垂直方向上跨越扫描线而在水平方向上跨越扫描线的像素位置的方式进行映射。图案矩阵按掩模的宽度和高度确定的间隔而重复使用。在图案覆盖的填充区域处，像素的颜色按掩模中存储的值进行设定。

影线填充通过在区域内绘制一组线段来显示单影线或交叉影线。影线的间隔和斜率可作为影线表的参数来设定。作为一种选择，影线填充可当做一个生成一组对角线的图案矩阵来指定。

作为填充图案开始位置的参考点(xp, yp)可设定在填充区内的任意位置。例如，该参考点可指定在多边形的一个端点处。也可将参考点设在由区域坐标范围确定的包围矩形（或包围盒）的左下

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

角。为了简化参考点的选择，有些软件包总是将显示窗口的坐标系原点作为图案起始位置。将(xp, yp)总是设定在坐标系原点还简化了在图案的每一元素映射到单个像素时的平铺操作。例如，如果图案中行的位置从1开始、从下往上计数，则图案位置$(y \bmod ny+1, x \bmod nx+1)$的颜色值赋给屏幕上的像素位置$(x, y)$。这里，$ny$和$nx$指定图案矩阵的行数和列数。然而，将图案起始点设在坐标系原点时，图案贴到屏幕背景上比贴到填充区更高效。使用同一图案的相邻或重叠填充不会在区域间生成边界。同样，使用同一图案重新定位或重新填充一个对象可能导致对象内部像素赋值的移位。相对于静止图案背景，移动的对象表现为透明的，而不是带着固定的内部图案移动。

4.9.2 颜色调和填充区域

也可以按照多种方式将填充图案和背景色相混合。图案和背景色混合时使用透明因子来确定背景中有多少应该混合到对象颜色中。我们也可以使用简单的逻辑操作或替代操作。图4.19给出了一个二进制（黑和白）系统中逻辑和替代操作怎样将2 × 2的填充图案与背景图案相混合的例子。

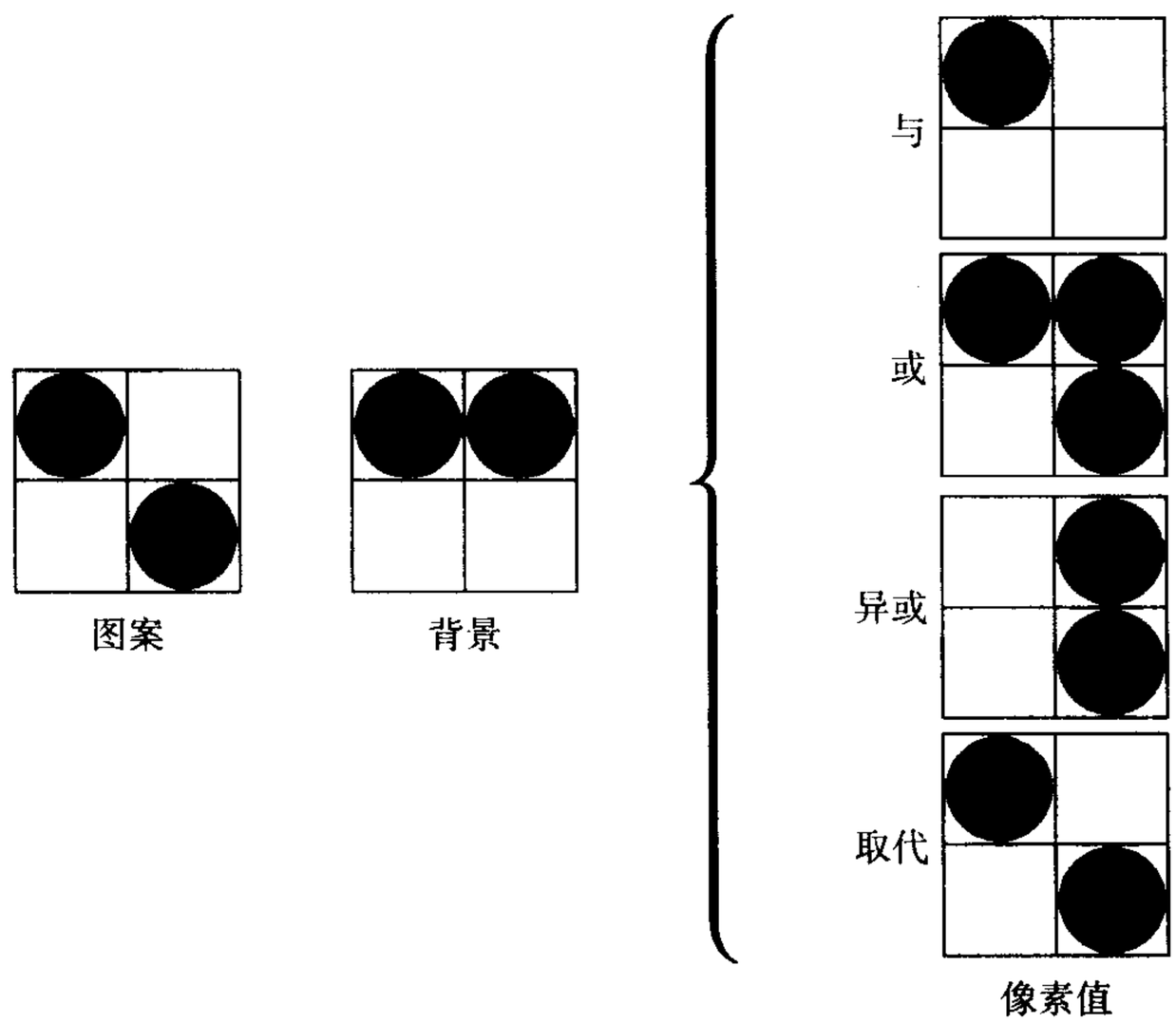


图 4.19 使用逻辑操作与、或和异或以及使用简单替代将填充图案与背景图案进行组合

某些使用调和颜色的填充方法称为**软填充**（soft-fill）或**色彩填充**（tint-fill）算法。这些填充方法的一种作用是，减弱在已经模糊的对象边界上的填充颜色，从而实现对边的反走样。另一种用途是允许对原来用半透明笔刷填充的颜色区域进行重新涂色。这时，当前颜色与笔刷颜色以及区域“后面”的背景色进行混合。无论是何种情况，都要求新的颜色在区域上与当前填充颜色具有相同的变化。

线性软填充算法（linear soft-fill algorithm）是这类填充的一个例子，该算法在将前景色**F**与单一背景色**B**（**F** = **B**）合并后绘制的区域上重新绘制。假如**F**和**B**的值为已知，那么通过检测帧缓存中当前的颜色内容，就可确定这些颜色原来是怎样组合的。区域内将要重新填充的每个像素的当前RGB颜色**P**，则是**F**和**B**的线性组合：

$$\mathbf{P} = t\mathbf{F} + (1 - t)\mathbf{B} \tag{4.2}$$

其中，“透明度”系数t对于每个像素，其值在0与1之间。t值小于0.5，则背景色对区域内部颜色的作用比填充颜色要大。向量方程（4.2）包含了颜色的RGB三个成分，即

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{P} = (P_R, P_G, P_B), \qquad \mathbf{F} = (F_R, F_G, F_B), \qquad \mathbf{B} = (B_R, B_G, B_B) \tag{4.3}$$

因此，我们可以使用任意一个 RGB 颜色成分来计算参数 t 的值：

$$t = \frac{P_k - B_k}{F_k - B_k} \tag{4.4}$$

其中，$k = R$、G 或 B，并且 $F_k \neq B_k$。理论上参数 t 对每个 RGB 成分具有相同的值，但取整到整数会使得 t 对不同的成分具有不同的值。我们可以通过选择 **F** 和 **B** 间具有最大差的成分，从而使取整误差最小，然后采用修改的泛滥填充或边界填充程序，根据这个 t 值将新的填充颜色 **NF** 与背景色相混合。如 4.13 节所述的使用修改的泛滥填充或边界填充过程可实现这一混合。

类似的软填充过程可以用于前景色与多个背景色相混合的区域，例如检测板图案。当两种背景色 B_1 和 B_2 与前景色 **F** 相混合时，产生的像素颜色 **P** 为

$$\mathbf{P} = t_0\mathbf{F} + t_1\mathbf{B_1} + (1 - t_0 - t_1)\mathbf{B_2} \tag{4.5}$$

其中，颜色项系数 t_0、t_1 和 $(1-t_0-t_1)$ 的和必须等于 1。我们可用 RGB 颜色三个分量中的两个来建立两个联立方程，从而求解两个比例项参数 t_0 和 t_1。然后，使用这些参数将新的填充颜色与两种背景色相混合以得到新的像素颜色。对于三个背景色和一个前景色或两个背景色和两个前景色，则需要所有三个 RGB 方程来得到四种颜色的相对量。然而，对于有些前景色和背景色的混合，使用两个或三个 RGB 方程则无法求解。在颜色值非常类似或相互成比例时，将出现这种情况。

4.10　通用扫描线填充算法

要实现区域的扫描线填充必须先确定填充区边界与屏幕扫描线的交点位置。然后，将填充色应用于扫描线上位于填充区域内部的每一段。扫描线填充算法利用奇偶规则识别同一内部区域（参见 3.15 节）。最简单的填充区域是多边形，因为每一扫描线和多边形的交点可通过求解一对联立的线性方程来获得，其中扫描线的方程是 y = 常数。

图 4.20 给出了多边形区域的实心填充的扫描线过程。对每一条与多边形相交的扫描线，与边的交点从左向右排序，且将每一对交点之间的像素位置包括这对交点在内，设定为指定颜色。在图 4.20 的例子中，与边界的四个交点像素位置定义了两组内部像素。这样，填充色应用于从 $x = 10$ 到 $x = 14$ 的 5 个像素和从 $x = 18$ 到 $x = 24$ 的 7 个像素。如果图案填充应用于多边形，则沿一条扫描线的每一像素颜色由与填充图案重叠的位置来确定。

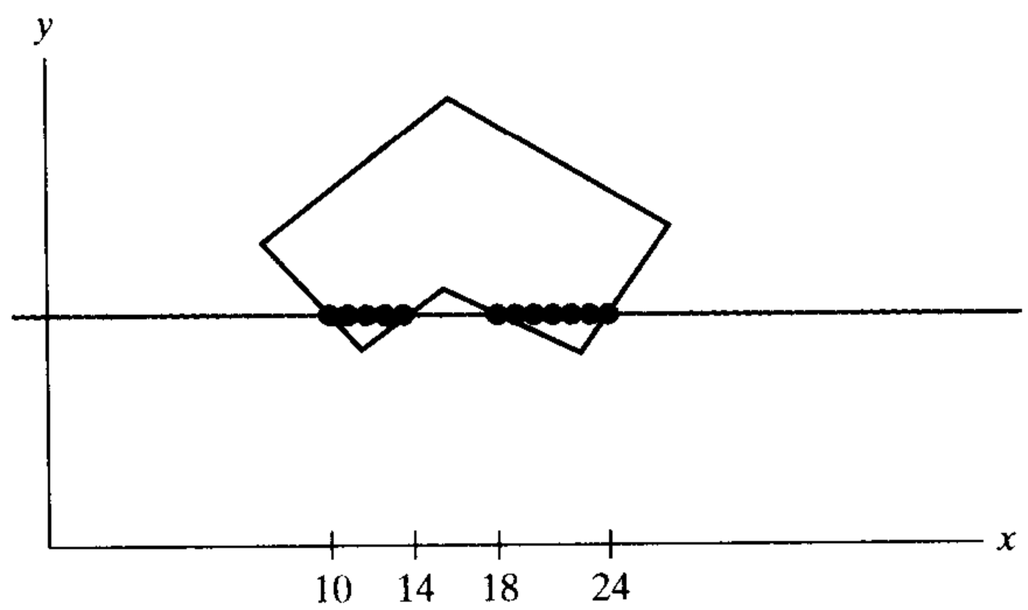


图 4.20　穿过多边形区域的扫描线上的内部像素

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

但是，多边形扫描线填充算法并不如图 4.20 建议的那样简单。每当一条扫描线经过多边形的一个顶点时，扫描线在该顶点处与多边形的两条边相交。这种情况可能导致在这条扫描线的交点列表上要增加两个点。图 4.21 给出了在顶点处与多边形相交的两条扫描线。扫描线 y' 与偶数条边相交，而在该扫描线上的两对交点正确地标识为内部像素段。但是扫描线 y 与多边形的五条边相交。要确认扫描线 y 的内部像素，必须将顶点处的交点看做为一个点。因此，在处理扫描线时，必须区分这些情况。

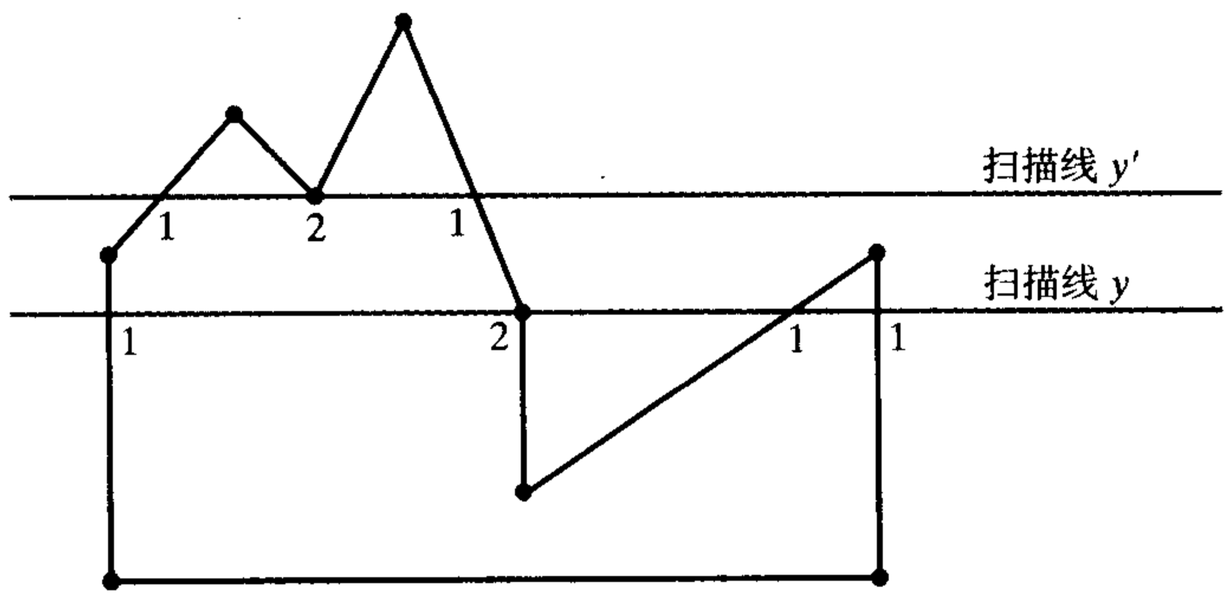


图 4.21 沿与多边形顶点相交扫描线的交点。扫描线 y 生成奇数个交点，扫描线 y' 生成偶数个交点，可以配对地正确确定内部像素段

通过关注相交边相对于扫描线的位置，可以发现图 4.21 中扫描线 y 和 扫描线 y' 间的拓扑差异。对于扫描线 y，共享一个顶点的两条相交边位于扫描线的两侧。但对扫描线 y'，两条相交边在扫描线的同一侧。因此，那些在扫描线两侧有连接边的顶点应该计为一个边界交点。可以通过顺时针或逆时针方向来搜索多边形边界，并观察从一条边移到另一条边时顶点 y 坐标的相对变化来识别这个点。假如两条相邻边的三个端点 y' 值单调递增或递减，那么对于任何穿过该顶点的扫描线，则必须将该共享（中间）顶点计为一个交点。否则，共享的顶点表示多边形边界上的一个局部极值（最大或最小）。这两条边与穿过该顶点的扫描线的交点可以添加到相关列表中。

将顶点交点调整为一个或两个的一种实现方法是将多边形的某些边缩短，从而分离那些应计为一个交点的顶点。我们可以按照指定的顺时针或逆时针方向处理整个多边形边界上的非水平边。在处理每条边时进行检测，确定该边与下一条非水平边是否有单调递增或单调递减的端点 y 值。假如有，可以将较低的一条边缩短，从而保证对通过公共顶点（连接两条边）的扫描线仅有一个交点生成。图 4.22 给出了一条边的缩短情况。当两条边的端点 y 值递增时，将当前边的较高端点 y 值减去 1，如图 4.22(a)所示；当端点 y 值单调递减时，如图 4.22(b)所示，就减去紧随当前边的一条边的较高端点 y 值。

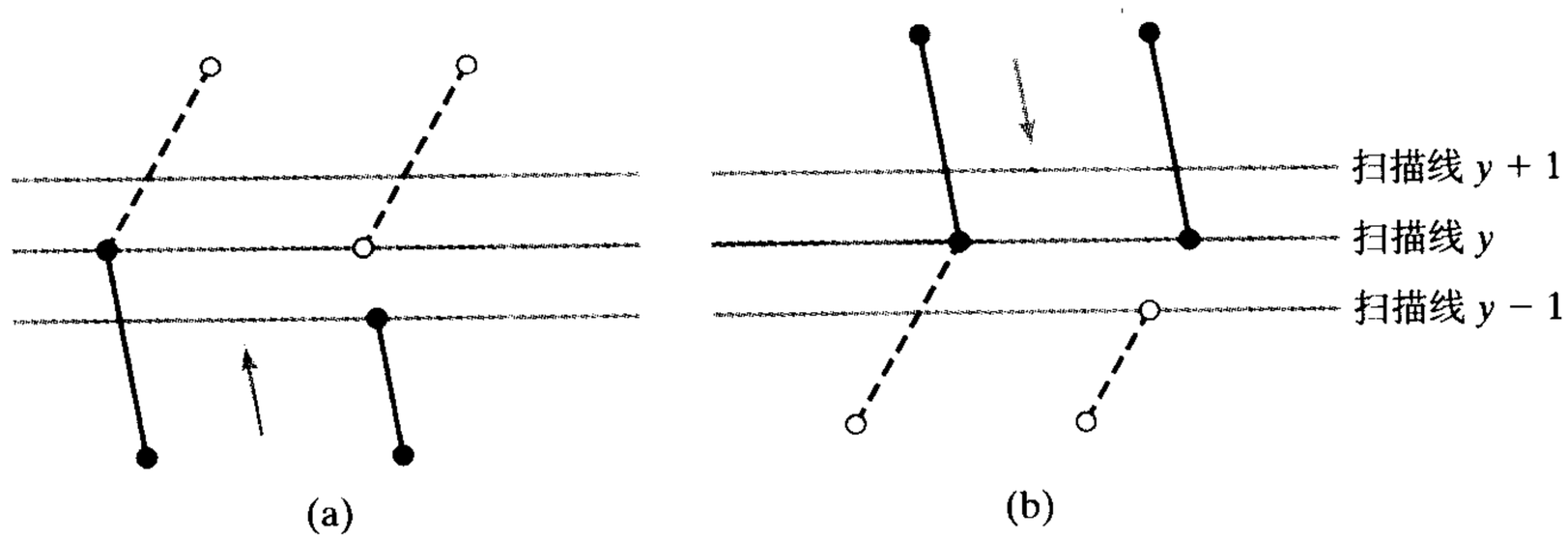


图 4.22 沿多边形边界处理边时调整多边形端点的 y 值。当前正在处理的边标为实线。在(a)中，将当前边的较高端点的 y 坐标值减 1；在(b)中，下一条边的较高端点的 y 坐标值减 1

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

一般情况下，场景一部分的某些特征会以某种方式与该场景另一部分的特征相关，且这些**相关特征**（coherence property）可用于计算机图形算法中以减少处理。相关方法经常包括沿一条扫描线或在连续的扫描线间应用的增量计算。例如，在确定填充区边的交点时，利用沿一条边从一条扫描线到下一条扫描时斜率为常数这一事实，可以沿任一边采用增量坐标计算。图4.23给出了与三角形左面一条边相交的两条连续扫描线，这条边的斜率可以用扫描线交点坐标来表示：

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} \tag{4.6}$$

由于两条扫描线间 y 坐标的变化很简单：

$$y_{k+1} - y_k = 1 \tag{4.7}$$

上面一条扫描线的 x 交点值 x_{k+1}，可以通过前一条扫描线的 x 交点值 x_k 来确定：

$$x_{k+1} = x_k + \frac{1}{m} \tag{4.8}$$

因此，每个后继交点的 x 值都可以通过增加斜率的倒数并取整而计算。

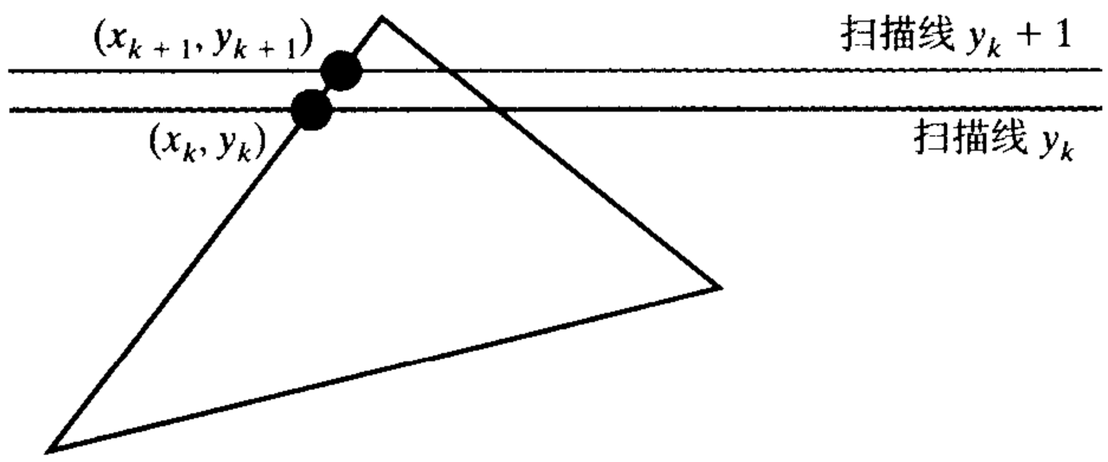


图 4.23　与同一多边形边界相交的两条连续扫描线

有关填充算法的最明显的并行实现方法，是将每条与多边形区域相交的扫描线分配给一个独立的处理器。然后分别完成每个边交点的计算。扫描线 k 沿一条具有斜率 m 的边，相对于最初扫描线的交点 x_k 值可计算为

$$x_k = x_0 + \frac{k}{m} \tag{4.9}$$

在顺序填充算法中，沿一条边 x 方向的增量值 $1/m$，可以通过调用斜率 m 为两整数比的整数运算来完成：

$$m = \frac{\Delta y}{\Delta x}$$

其中，Δx 和 Δy 是该边端点 x 和 y 坐标值之间的差。因此，沿一条边对连续两条扫描线交点的 x 增量计算可表示为

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y} \tag{4.10}$$

利用这个公式，可以完成交点 x 坐标的整数求值：先将计数器初始化为零，然后每当移向一条新的扫描线时，计数器就增加 Δx 值，从而完成交点 x 坐标的整数求值。当计数器的值大于等于 Δy 时，当前交点 x 值增加 1，并将计数器减去 Δy。这个过程相当于保持交点 x 值的整数和小数部分，并增加小数部分直至达到下一个整数值。

作为整数增量的一个例子，假设一条边的斜率为 $m = 7/3$。在起始扫描线处，我们将计数器设置为零，增量为 3。当沿这条边移到其他三条扫描线时，计数器顺序地设置值 3、6 和 9。在初始扫

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

描线以上的第三条扫描线上，计数器的值大于7。因而交点x坐标增加1，并重新将计数器设置为值9–7 = 2。继续以这种方法确定扫描线的交点值，直至到达边界的最高端点。对于负斜率的边，可以通过同样的计算而得到交点。

我们可以不用舍入法以获得整数位置，而是取整到最接近的像素的x值，通过修改边的相交算法使得增量与$\Delta y/2$相比较。在每一步中计数器增加$2\Delta x$值，并将增量与Δy进行比较，当增量大于或等于Δy值时，x值增加1，而计数器值减去$2\Delta y$。在上面$m = 7/3$的例子中，对于这条边上初始扫描线以上的几条扫描线，其计数器值变为6、12（减少到–2）、4、10（减少到–4）2、8（减少到–6）、0、6和12（减少到–2）。在这条边的初始扫描线以上的第2、4、6、9扫描线上，将增加x的值。每条边所需的额外计算是$2\Delta x = \Delta x+\Delta x$和$2\Delta y = \Delta y+\Delta y$，这些计算在预处理中完成。

为了有效地完成多边形填充，可以首先将多边形边界存储在有序边表（sorted edge table）中，其中包含有效处理扫描线所需的全部信息。无论是以顺时针或逆时针沿边处理时，我们都可以使用桶排序来存储各条边，按每条边的最小y值排序，存储在相应的扫描线位置。有序边表中仅存储非水平线。在处理边时，可以缩短某些边以解决顶点相交问题。对于某条特定的扫描线，表中的每个入口包含该边的最大y值、边的x交点值（在较低顶点处）和边斜率的倒数。对于每条扫描线，以从左到右的顺序对边进行排序。图4.24给出了一个多边形及其相应的有序边表。

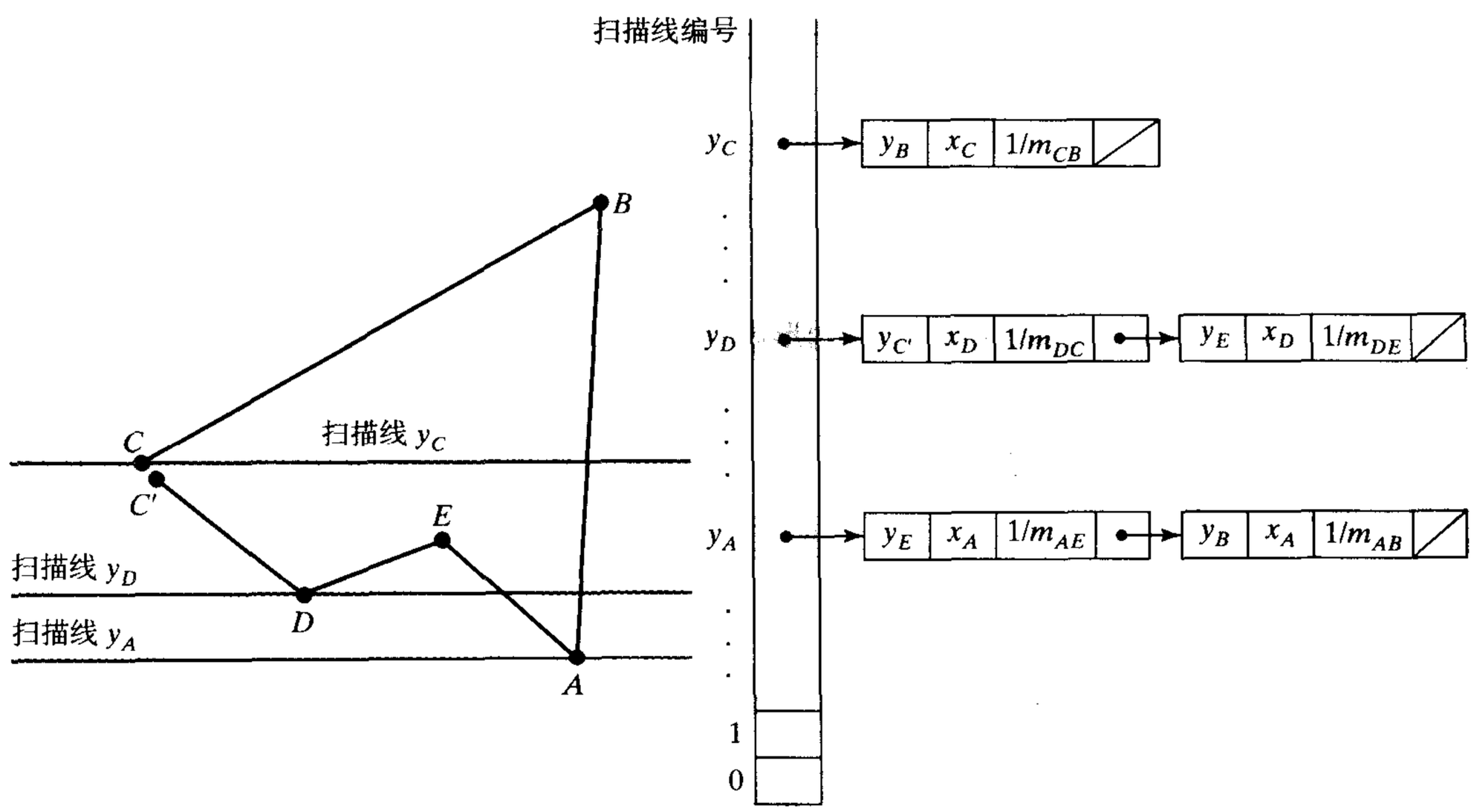


图4.24 多边形及其有序边表，边$\overline{DC}$在y方向缩短了一个单位

接下来，从多边形的底部到顶部处理扫描线。对每条与多边形边界相交的扫描线生成一个活化边表（active edge list）。扫描线的活化边表包含所有与该扫描线相交的边，并使用重复相关性计算来得到边的交点。

边的相交计算也可通过将Δx和Δy值存储在有序边表中而得到简化。此外，为了确保对指定多边形内部的正确填充，可以应用在3.13节中所讨论的考虑情况。对于每条扫描线，对从最左边的x交点值到最右边x交点值之间的每一对x交点间的像素段进行填充。而每条多边形的边可以在顶部端点y方向上缩短一个单位，这种措施也能保证相邻多边形中的像素不会相互覆盖。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

4.11　凸多边形的扫描线填充

在将扫描线填充过程应用于凸多边形时，每一屏幕扫描线上的内部段将不会多于一个。因此，只要在发现与边界有两个交点时才处理该扫描线穿过多边形的内部段。

在上一节中讨论的通用多边形扫描线算法对于凸多边形填充可以简化。我们可再次使用坐标范围来确定哪些边与一扫描线相交。与这些边的交点计算确定该扫描线的内部段，其中任一顶点计为单个边界交点。当扫描线仅与单个顶点相交时（如在多边形的一个顶点），则仅绘出该点本身。有些图形软件包进一步把填充区限定为三角形。这使得填充更易于进行，因为每个三角形只有三个顶点要处理。

4.12　曲线边界区域的扫描线填充

由于曲线边界的区域用非线性方程描述，其扫描线填充比多边形扫描线填充需要更多的时间。我们可以使用4.10节中的通用方法，但边界交点计算用曲线方程完成。并且其边界的斜率不断地改变，因而不能直接使用直线段边可以使用的增量方法。

对于像圆和椭圆这样的简单曲线，可以像凸多边形一样直接应用扫描线填充。每一条与圆或椭圆相交的扫描线仅有两个边界交点。我们可以使用中点方法中的增量计算来确定沿圆或椭圆边界的这两个交点。然后，简单地在一个交点到另一个之间的水平像素段内进行填充。利用四分象限区间（对于圆为八分象限）的对称性可以减少边界计算量。

对于曲线段的填充区域可以使用类似的方法来生成。例如，一个椭圆弧和一直线段为边界的区域（参见图4.25）可以使用曲线和直线段过程的混合方法来填充。只要可以减少计算量，就应充分利用对称性和增量计算。

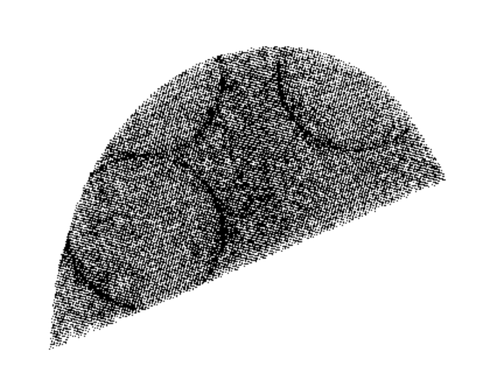


图 4.25　一个椭圆弧的内部填充

填充其他曲线区域可能需要更多的时间。我们可以使用类似的增量方法并与数值方法组合来确定扫描线交点，但那样的曲线边界通常用直线段逼近。

4.13　不规则边界区域的填充方法

区域填充的另一种方法是从区域的一个内部点开始，由内向外逐点绘制直到边界。这对如绘画程序生成的有不规则边界的填充区域是很有用的技术。一般来说，这些方法要求输入待填充区域中的一个起始位置，以及关于边界或内部的颜色信息。

我们可以使用单一颜色或一个颜色图案来填充不规则区域。对于图案填充来说，如4.9节讨论的那样重叠一个颜色掩模。在处理区域中的每一个像素时，其颜色由对应的重叠掩模中的值来确定。

4.13.1　边界填充算法

假如边界是以单一颜色指定的，则填充算法可逐个像素地向外处理，直至遇到边界颜色。这种方法称为**边界填充算法**（boundary-fill algorithm），用于比较容易地指定内点交互式绘画软件。艺术

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

家或设计师可以使用图形板或其他交互设备来勾勒图形的轮廓，从颜色菜单中选择一种填充色，指定一种边界色，并选择一个内部点。然后在图形的内部涂上填充色。内边界和外边界可一起用来定义边界填充的区域，图 4.26 给出了指定颜色范围的例子。

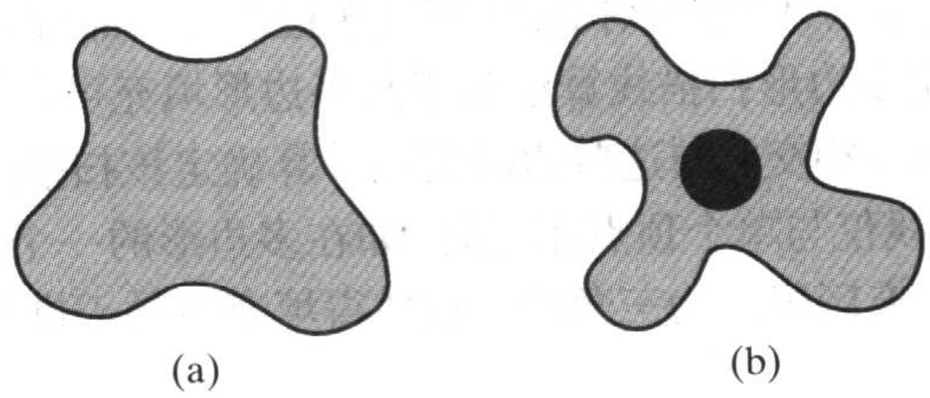


图 4.26 用于边界填充过程的颜色边界示例

基本上，边界填充算法从一个内点(x, y)开始检测相邻位置的颜色。如果检测位置不是该边界颜色，就将它改为填充颜色，并再检测其相邻位置。这个过程延续到检测完区域边界颜色范围内的所有像素为止。

图4.27给出了从一个当前检测位置处理相邻像素的两种方法。在图4.27(a)中，检测四个邻点。即当前像素右面、左面、上面和下面的像素位置。使用这种方法填充的区域称为**4-连通**（4-connected）区域。图 4.27(b)中的第二种方法用于填充更复杂的图形。这里要测试的相邻位置包括四个对角像素。使用这种方法填充的区域称为**8-连通**（8-connected）区域。8-连通区域边界填充算法将正确填充图4.28中定义的区域内部，而4-连通区域边界填充算法只完成如图所示的部分填充。

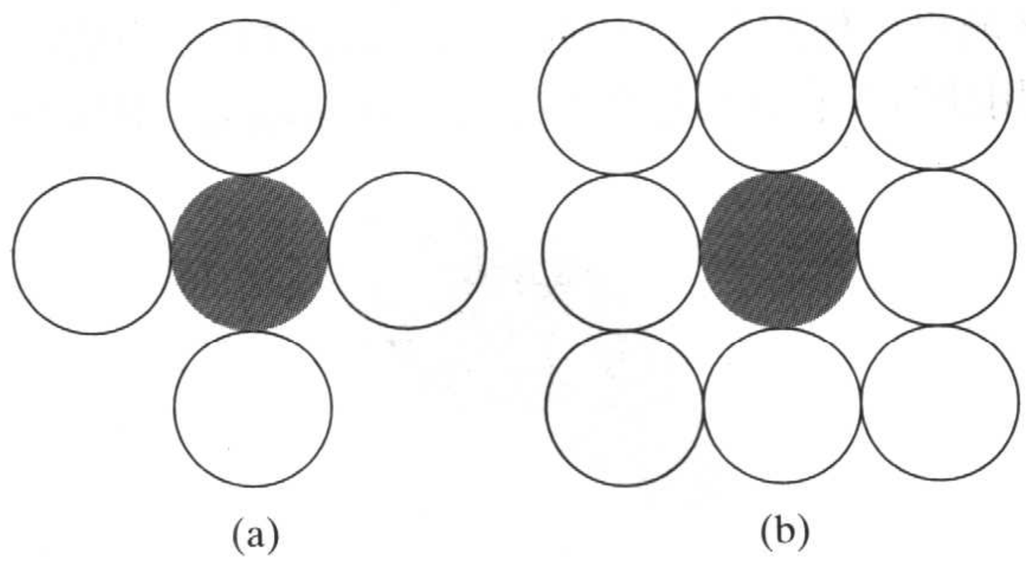


图 4.27 (a)应用于4-连通区域的填充方法；(b)应用于8-连通区域的填充方法。实心圆表示当前测试位置，空心圆表示将要测试的位置

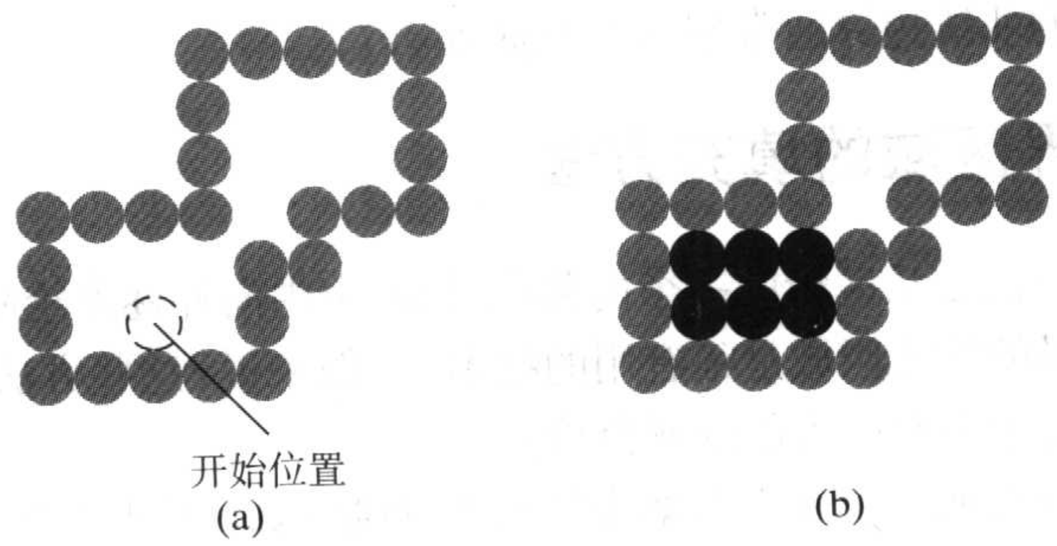


图 4.28 (a)颜色边界内的区域；(b)使用 4-连通的边界填充算法只能进行部分填充

下列过程给出了使用由参数 fillColor 指定的单一颜色及由参数 borderColor 指定的边界颜色来填充 4-连通区域的递归方法。我们可以将它扩充成填充 8-连通区域，只要增加四条测试诸如$(x\pm1, y\pm1)$等对角位置的语句。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void boundaryFill4 (int x, int y, int fillColor, int borderColor)
{
   int interiorColor;

   /* Set current color to fillColor, then perform following oprations. */
   getPixel (x, y, interiorColor);
   if ((interiorColor != borderColor) && (interiorColor != fillColor)) {
      setPixel (x, y);    // Set color of pixel to fillColor.
      boundaryFill4 (x + 1, y , fillColor, borderColor);
      boundaryFill4 (x - 1, y , fillColor, borderColor);
      boundaryFill4 (x , y + 1, fillColor, borderColor);
      boundaryFill4 (x , y - 1, fillColor, borderColor)
   }
}
```

假如有些内部像素已经以填充颜色显示，则递归式的边界填充算法也许不能正确地填充区域。这是因为算法既按边界颜色又按填充颜色来检测下一个像素。遇到一个具有填充颜色的像素将导致该递归分支终止，从而留下一些尚未填充的内部像素。为了避免这种情况，可在应用边界填充程序前，将那些初始颜色改为填充颜色的内部像素的颜色。

此外，由于这个程序需要大量堆栈空间来存储相邻点，因而通常使用更有效的方法。这些方法沿扫描线填充水平像素区段，从而代替处理4-连通或8-连通相邻点。然后，仅需将每个水平像素区段的起始位置放进堆栈，而不需将所有当前位置周围的未处理相邻位置都放进堆栈。如果使用这种方法，将从初始内部点开始，首先填充该像素所在扫描行的连续像素区段，然后将相邻扫描线上各段的起始位置放进堆栈，这些水平段分别由像素（显示为区域边界颜色）包围，接着逐步从堆栈顶部取出一个开始点，并重复上述过程。

图4.29给出了如何使用逼近法对4-连通区域填充像素区段的例子。在这个例子中，从起始扫描线开始向顶部边界顺序地处理。在处理完所有上面的扫描线以后，再向下逐行填充剩余的像素段，直到底部边界。沿着每条扫描线，从左向右将每个水平区段的最左边像素位置放进堆栈，如图4.29所示。在图4.29(a)中，填充完初始区段以后，接下来的一条扫描线（向下和向上）上的段起始位置1和2进栈；在图4.29(b)中，从堆栈中取出位置2并生成填充区段，然后将下一个扫描线上单个区段的起始像素（位置3）放进堆栈。在处理完位置3后，填充过的区段和进栈的位置如图4.29(c)所示。图4.29(d)给出了在处理指定区域右上角的所有区段后的已填充像素。接着处理位置5，并且对区域的左上角填充，然后取出位置4继续对较低的扫描线进行处理。

4.13.2 泛滥填充算法

有时，我们要对一个不是用单一颜色边界定义的区域进行填充（或重新涂色）。图4.30给出了由多个不同颜色区域围成的区域。可以通过替换指定的内部颜色而不是搜索边界颜色值来对该区域涂色。这个方法称为**泛滥填充算法**（flood-fill algorithm）。从指定的内部点(x, y)开始，将期望的填充颜色赋给所有当前设置为给定内部颜色的像素。假如所要涂色的区域具有多种内部颜色，可以重新设置像素值，从而使所有的内部点具有相同的颜色。然后使用4-连通或8-连通方法，逐步连通各像素位置，直到所有内部点都已被涂色。下列过程从输入位置开始，递归地填充一个4-连通区域。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

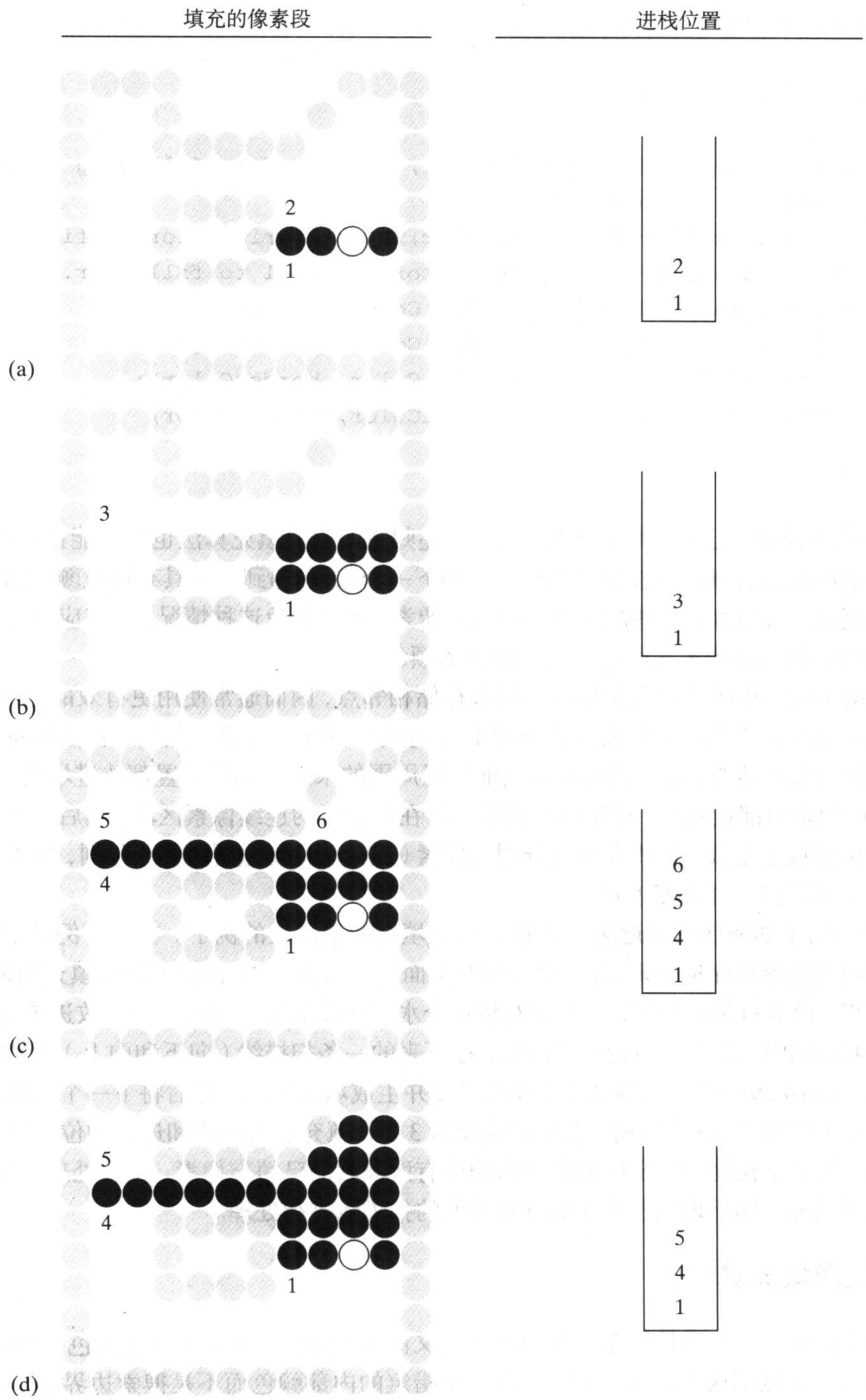


图 4.29　4-连通区域中穿过像素区段的边界填充：(a)填充过的起始像素区段，给出了初始点的位置和储存相邻扫描线上的像素区段位置；(b)初始扫描线之上第一条扫描线填充的像素区段及储存的当前内容；(c)在初始扫描线之上前两条扫描线的填充像素区段及储存的当前内容；(d)定义区域右上角部分的完整像素区段及剩余待处理的储存的位置

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void floodFill4 (int x, int y, int fillColor, int interiorColor)
{
   int color;

   /* Set current color to fillColor, then perform following operations. */
   getPixel (x, y, color);
   if (color = interiorColor) {
      setPixel (x, y);     // Set color of pixel to fillColor.
      floodFill4 (x + 1, y, fillColor, interiorColor);
      floodFill4 (x - 1, y, fillColor, interiorColor);
      floodFill4 (x, y + 1, fillColor, interiorColor);
      floodFill4 (x, y - 1, fillColor, interiorColor)
   }
}
```

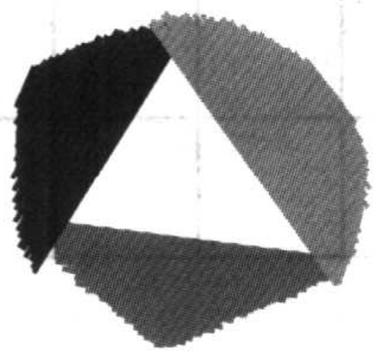


图 4.30　多个颜色边界定义的区域

我们可以像边界填充算法中讨论的那样修改上面的过程，通过填充水平像素区段来减少对堆栈的存储要求。在这种方法中，仅存储具有值 interiorColor 的像素区段的起始位置。这种修改后的泛滥填充算法的步骤与图4.29所示的边界填充相同，从每个区间的第一个位置开始替代像素值直至遇到 interiorColor 以外的值。

4.14　OpenGL 填充区属性函数

OpenGL 中仅提供对凸多边形的填充区子程序。显示一个填充凸多边形要经过下列四个步骤：

1. 定义一个填充图案。
2. 引用多边形填充子程序。
3. 激活 OpenGL 多边形填充特性。
4. 描述要填充的多边形。

多边形填充图案一直显示到包括多边形的边。因此，填充区中没有边界线，除非特别指定要显示边界线。

除了为多边形内部指定一个填充图案，还有其他多个的选项。一个选项是显示一个空心多边形，其中仅生成边而没有内部颜色或图案。空心多边形与显示一个封闭多边形图元等价。另一选项是只显示多边形顶点而没有内部填充且没有边。同样，我们也可以为多边形填充区的前向面和后向面指定不同的属性。

4.14.1　OpenGL 填充图案函数

默认时，凸多边形使用当前颜色设定显示成一个实心颜色区域。为了用OpenGL的图案填充一个多边形，我们使用 32 位 × 32 位的掩模。掩模中值为 1 表示对应像素设为当前颜色，值为 0 表示对应的帧缓存位置的值不变。填充图案使用 OpenGL 数据类型 GLubyte 以无符号字节进行描述，如同在 glBitmap 函数中一样。例如，用十六进制值定义一个位图：

```
GLubyte fillPattern [ ] = {
    0xff, 0x00, 0xff, 0x00, ... };
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如 3.19 节中的 bitShape 一样，这些位必须从图案的底行开始描述，直到图案的最高行（32）。该图案从显示窗口的左下角开始，在整个显示窗口中重复，而指定的多边形用与其重叠的图案部分填充（参见图 4.31）。

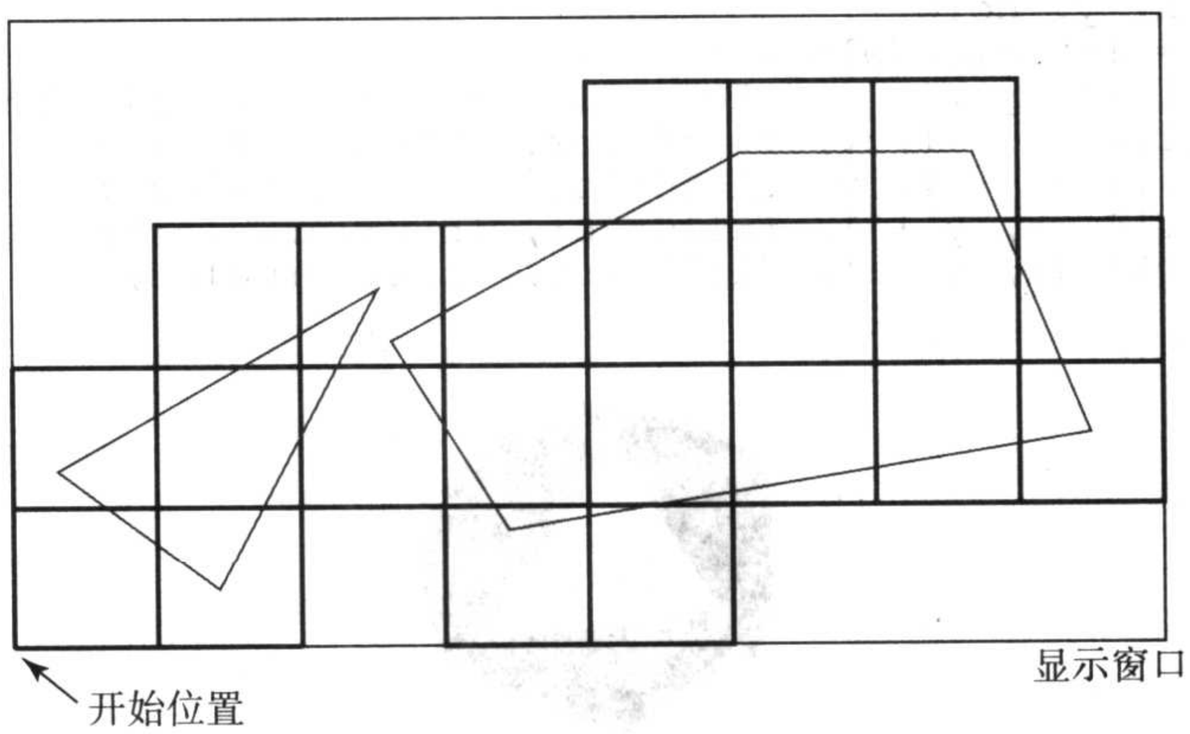


图 4.31　通过将矩形填充图案平铺穿越显示窗口来填充两个凸多边形

一旦建立了一个掩模，可用下列函数将其用做当前填充图案：

```
glPolygonStipple (fillPattern);
```

接下来，在指定要用当前图案填充的多边形顶点之前必须激活填充子程序。这使用下列语句来实现：

```
glEnable (GL_POLYGON_STIPPLE);
```

类似地，使用下列语句关掉图案填充：

```
glDisable (GL_POLYGON_STIPPLE);
```

图 4.32 给出了 3 × 3 的点阵如何覆盖 32 × 32 的位掩模，该掩模可用于填充一个平行四边形。

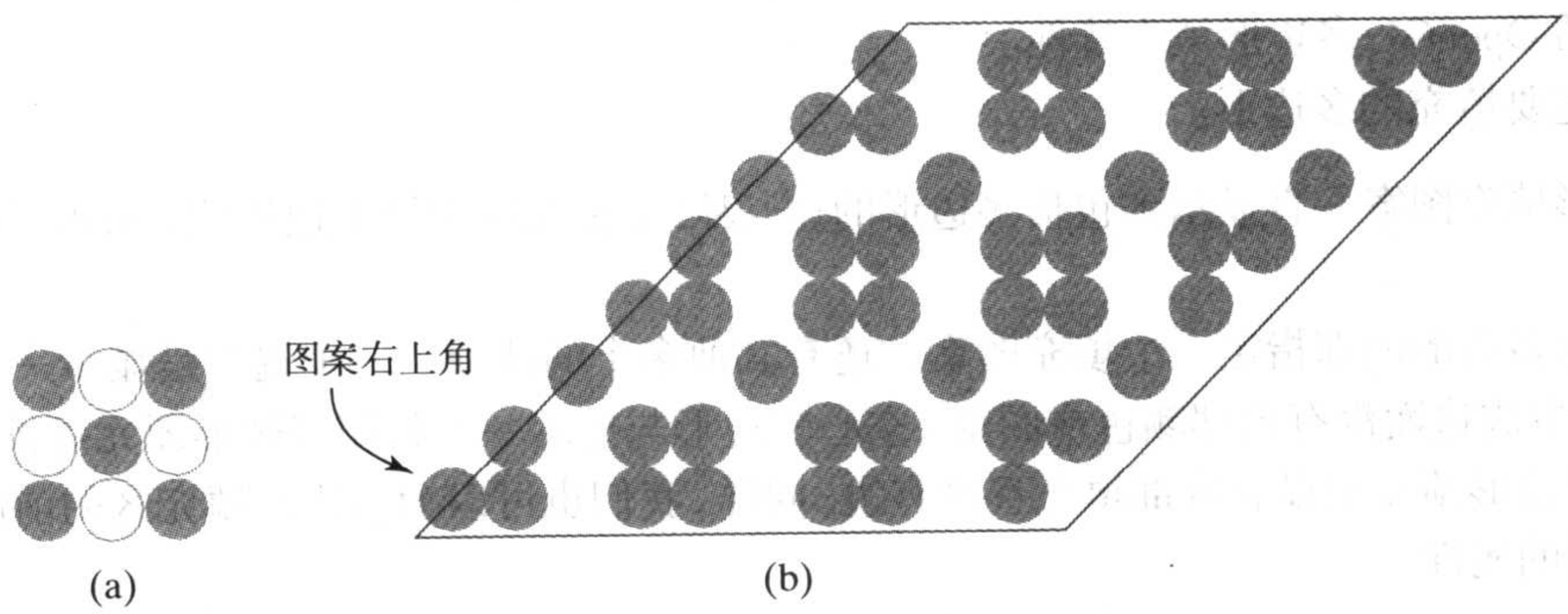


图 4.32　(a)3 × 3 的点阵图案；(b)放到平行四边形中以生成填充区，其中图案的右上角与平行四边形的左下角对齐

4.14.2　OpenGL 纹理和插值图案

填充多边形的另一种方法是使用在第 10 章中将讨论的纹理。这将生成仿真木材、砖、纹钢（brushed steel）或某些其他材料外貌的图案。也可以如线图元中的那样得到多边形内部的插值颜色。为此，我们对多边形的顶点赋以不同颜色。插值填充用来为各种光照条件下的着色表面生成真实感显示。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

作为插值填充的例子，下面的程序段将蓝色、红色和绿色分别赋给一个三角形的三个顶点。多边形填充就是在三个顶点间的颜色插值。

```
glShadeModel (GL_SMOOTH);

glBegin (GL_TRIANGLES);
   glColor3f (0.0, 0.0, 1.0);
   glVertex2i (50, 50);
   glColor3f (1.0, 0.0, 0.0);
   glVertex2i (150, 50);
   glColor3f (0.0, 1.0, 0.0);
   glVertex2i (75, 150);
glEnd ( );
```

当然，如果为三角形整体设定一个单色，则用一种颜色来填充多边形。而如果将 glShadeModel 函数中的变量改变为GL_FLAT，则多边形用最后指定的颜色（绿色）来填充。值GL_SMOOTH是着色的默认值，但包含这一描述可提醒我们多边形将用顶点插值法填充。

4.14.3　OpenGL 线框图方法

我们也可以仅显示多边形的边，这生成线框图或多边形的空心显示。也可以通过显示一组顶点来显示多边形。这些选项通过下面的函数来选定：

```
glPolygonMode (face, displayMode);
```

参数 face 用来指定在多边形的哪一个面上仅显示边或顶点。它赋以 GL_FRONT、GL_BACK 或 GL_FRONT_AND_BACK。然后，如果选择仅显示多边形的边，就给参数 displayMode 赋值 GL_LINE。如果仅绘出多边形顶点，则将GL_POINT赋给参数displayMode。第三个选项是GL_FILL。但这是默认的显示模式，所以我们通常仅在要设定多边形边或顶点属性时引用 glPolygonMode。

另一个显示多边形的选项是在填充内部的同时用不同的颜色或图案来显示它的边或顶点。这通过指定该多边形两次来实现：先将 displayMode 设定为 GL_FILL 然后再设定为 GL_LINE（或 GL_POINT）。例如，下面的程序段用绿色填充多边形内部，而用红色显示边。

```
glColor3f (0.0, 1.0, 0.0);
\* Invoke polygon-generating routine. */

glColor3f (1.0, 0.0, 0.0);
glPolygonMode (GL_FRONT, GL_LINE);
\* Invoke polygon-generating routine again. */
```

对于一个三维多边形（即并不是所有顶点都在 xy 平面内）来说，这种显示填充多边形边的方法可能在边之间的生成缝隙。这种称为**缝线**（stitching）的效果由扫描线填充算法和边的画线算法的计算差别造成。在对一个三维多边形进行填充时，深度值（离 xy 平面的距离）按每一(x, y)位置计算。但是在多边形一条边上的这个深度值通常与在同一(x, y)位置用画线算法计算所得的深度值不完全相同。因此，在进行可见性测试时，内部填充色可用来代替边的颜色以显示沿多边形边界的点。

消除三维多边形显示边的缝隙的一个办法是移动由填充子程序计算的深度值，使它们与多边形的边深度值不重叠。下列两个函数用来实现这一点。

```
glEnable (GL_POLYGON_OFFSET_FILL);
glPolygonOffset (factor1, factor2);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第一个函数激活扫描线填充的位移子程序，而第二个函数用来设定一对计算深度位移总量的浮点值 factor1 和 factor2。这一深度位移的计算是

$$\text{depthOffset} = \text{factor1} \cdot \text{maxSlope} + \text{factor2} \cdot \text{const} \tag{4.11}$$

这里，maxSlope 是多边形的最大斜率，const 是实现常数。对于 xy 平面上的一个多边形来说，斜率为0。否则，最大斜率用多边形深度的变化除以 x 或 y 的变化而得。两因子的典型值是0.75和1.0，尽管某些经验因子值对生成好的结果是必要的。作为对位移因子赋值的例子，我们可以把前一程序段修改如下：

```
glColor3f (0.0, 1.0, 0.0);
glEnable (GL_POLYGON_OFFSET_FILL);
glPolygonOffset (1.0, 1.0);
\* Invoke polygon-generating routine. */
glDisable (GL_POLYGON_OFFSET_FILL);

glColor3f (1.0, 0.0, 0.0);
glPolygonMode (GL_FRONT, GL_LINE);
\* Invoke polygon-generating routine again. */
```

现在将多边形的内部填充往更深处加强了一些，因此它不妨碍其边的深度值。也有可能通过把位移应用于画线算法，即通过将glEnable函数的变量改变为GL_POLYGON_OFFSET_LINE来实现该方法。此时，我们要使用负因子使边的深度值靠近些。如果仅需要显示多边形顶点位置的不同颜色点而不是醒目的边，那么 glEnable 函数的变量将是 GL_POLYGON_OFFSET_POINT。

另一种消除多边形边上缝线效果的方法是使用OpenGL的模板缓存来限制多边形内部填充，从而使它和边不重叠。但这种方法较为复杂且一般较慢，因而多数使用深度位移方法。

为了使用OpenGL函数显示一个凹多边形，必须先将它分割为一组凸多边形。凹多边形一般用3.15节讨论的方法分割为一组三角形。然后通过填充这些三角形来显示填充的凹多边形。类似地，如果只要显示多边形顶点，我们可以只绘制三角形顶点。但为了显示原凹多边形的线框图，就不能将显示模式设定为GL_LINE，因为这将把作为多边形内部的所有三角形的边显示出来（参见图4.33）。

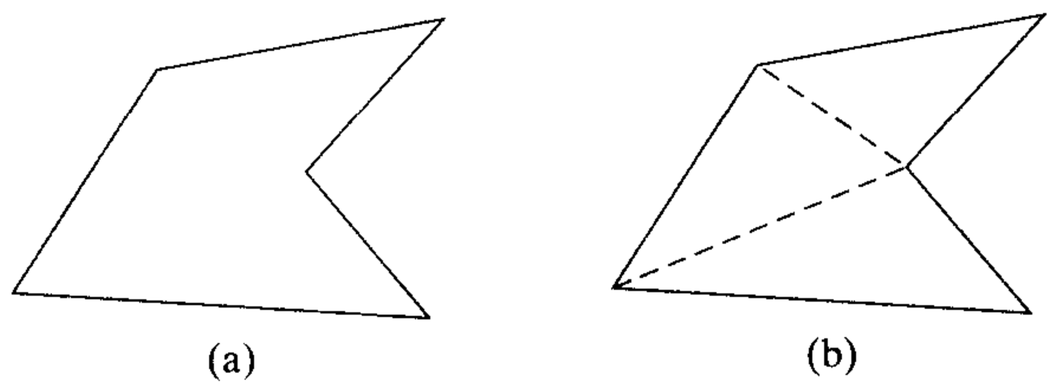


图 4.33 将(a)中的凹多边形分割为(b)中的一组三角形，生成的三角形边（虚线）是原凹多边形的内部

幸好，OpenGL提供了从线框图显示中消除选定边的机制。每一个多边形顶点与一个指示该顶点是否通过边界上的边与下一顶点连接的一位标志一起存储。我们要做的就是将该位标志设为“关”，从而使在这个顶点之后的边不被显示。使用下列函数可以设定一条边的这个标志。

```
glEdgeFlag (flag);
```

要指明一个顶点不在边界的边的前面，需要将OpenGL常量GL_FALSE赋给参数flag。这作用于所有其后的顶点直至再次调用 glEdgeFlag。OpenGL 常量 GL_TRUE 再次将标志置回工作状态，即默认状态。函数 glEdgeFlag 可放在 glBegin/glEnd 之间。作为边标志应用的一个展示，下列程序段仅显示所定义三角形的两条边（参见图 4.34）。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glPolygonMode (GL_FRONT_AND_BACK, GL_LINE);

glBegin (GL_POLYGON);
   glVertex3fv (v1);
   glEdgeFlag (GL_FALSE);
   glVertex3fv (v2);
   glEdgeFlag (GL_TRUE);
   glVertex3fv (v3);
glEnd ( );
```

多边形边标志也可以在一个数组中指定，该数组可与顶点数组混合或作为其附加数组（参见 3.17 节和 4.3 节）。创建一个边标志数组的语句为

```
glEnableClientState (GL_EDGE_FLAG_ARRAY);

glEdgeFlagPointer (offset, edgeFlagArray);
```

参数 offset 指出数组 edgeFlagArray 中边标志值之间的字节数。参数 offset 的默认值是 0。

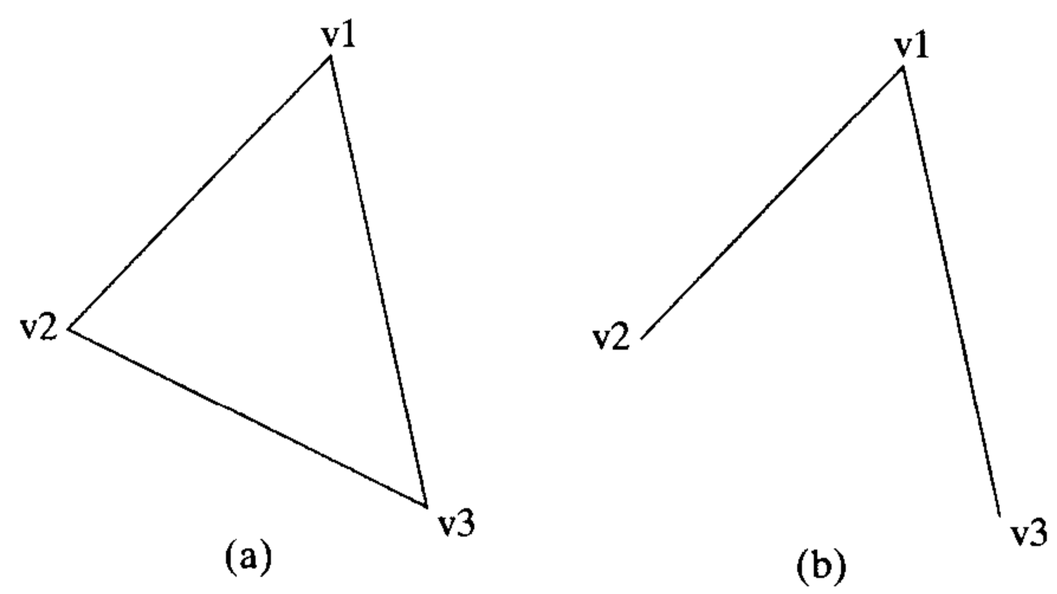


图 4.34　设顶点按逆时针次序指定，通过将顶点 v2 的边标志设定为 GL_FALSE 可将(a)中多边形显示成(b)中那样

4.14.4　OpenGL 前向面函数

尽管默认情况下由多边形顶点的次序来控制前向面和后向面的确认，我们还是可以使用下列函数单独地指定一场景中的前向面和后向面：

```
glFrontFace (vertexOrder);
```

如果设定参数 vertexOrder 的值为 OpenGL 常量 GL_CW，则随后定义的顺时针多边形可看做前向面。该 OpenGL 特性用来交换以顺时针次序指定顶点的多边形的面。常量 GL_CCW 标志多边形顶点的逆时针次序为前向面朝向，即默认次序。

4.15　字符属性

显示的字符外观由字体、大小、颜色和方向这些属性控制。在许多软件包中，既可对整个字符串（文本）设置属性，也可对如绘制数值图等特殊应用中的单个字符设置属性。

有许多可供图形程序员使用的文本选项。首先是选择字体，字体就是使用类似 New York、Courier、Helvetica、London、Times Roman 等特定设计风格的一组字符和其他一些特殊的符号组。所选字体的字符也可以使用附加的下划线风格（实线、虚线和双线）、**黑体**、*斜体*、[illegible]或影线风格。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

显示文本的颜色设置存储在系统属性表中，并由将字符定义装入帧缓存中的程序所使用。显示字符串的时侯，使用当前颜色来设置与字符形状和位置相对应的帧缓存中的像素值。

我们也可以通过缩放字符的整体尺寸（高度和宽度）或者仅缩放字符高度或宽度来调整文本大小。字符大小（高度）则由打印机和排字机以磅（point）为单位进行指定，其中1磅是0.035 146厘米（或0.013 837英寸、大约1/72英寸）。例如，本书的文本是10磅字体。磅值计量指定了字符体的大小（参见图4.35），但具有相同磅数的不同字体，按其字体设计的不同而具有不同的字符大小。在指定大小的一种字体中，所有字符的底线（bottomline）和顶线（topline）间的距离是相同的，但字符体的宽度可能不同。在成比例间距字体（proportionally spaced font）中，窄字符*i*、*j*、*l*和*f*的字符体宽度比宽字符*W*或*M*要小。字符高度（character height）则定义为字符基线（baseline）和帽线（capline）之间的距离。像图4.35中*f*和*j*这样的有核字符通常超出字符体的限制。下行字符（*g*、*j*、*p*、*q*、*y*）要扩展到基线以下。每个字符由设计师定位在字符体以内，并允许沿打印行或在打印行之间以字符体相接方式显示时，可以有适当的间距。

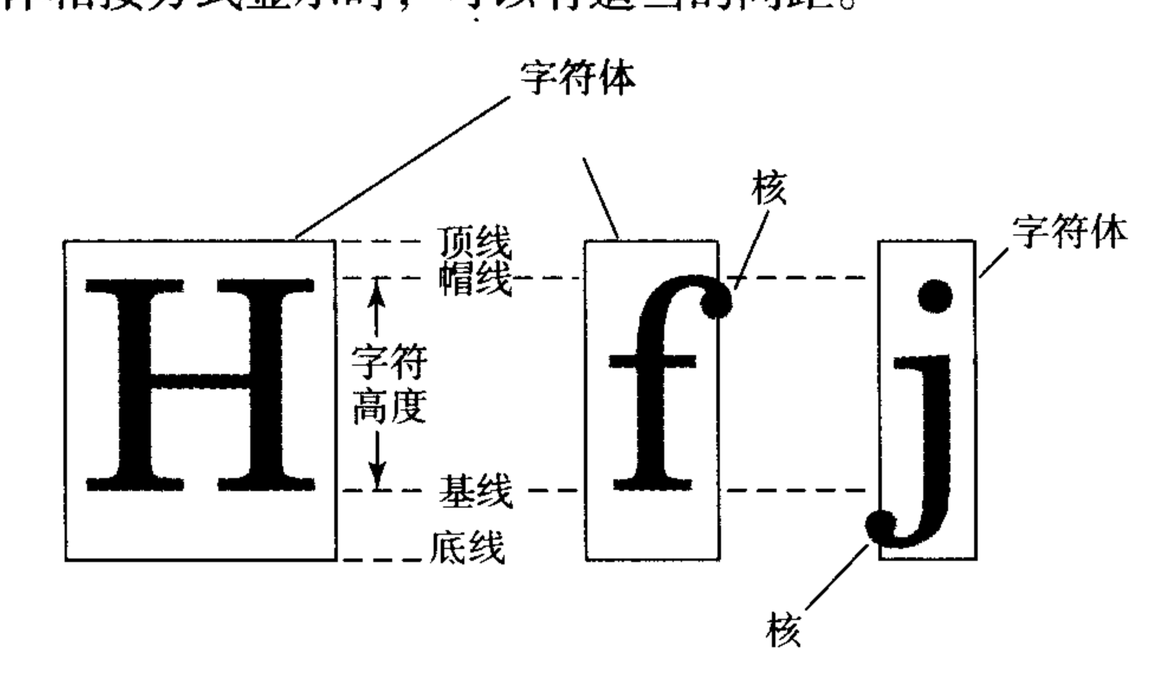


图4.35 字符体示例

有时可以在不改变字符的宽高比的情况下调整文本大小。图4.36给出了在保持宽高比不变时三种不同字符高度的字符串显示。图4.37给出了使用常量高度和可变宽度的文字显示。

Height 1

Height 2

Height 3

图4.36 使用不同高度设定和常量的宽高比显示的文本

width 0.5

width 1.0

width 2.0

图4.37 使用固定高度和可变宽度显示的文本

字符之间的间隔是字符串经常要设定的另一个属性。图4.38给出了用三种字符间隔设定而显示的字符串。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Spacing 0.0

S p a c i n g　0 . 5

S　p　a　c　i　n　g　　1　.　0

图 4.38　使用不同的字符间隔值显示的字符串

字符串的方向按**字符向上向量**（character up vector）设定。字符串显示成其字符底线到帽线的方向与向上向量一致。例如，在向上向量的方向为45°时，字符串显示成如图4.39所示。为文字定向的过程将字符旋转到字符体两侧从底线到帽线的方向并符合向上向量。旋转后的字符形状再扫描转换到帧缓存。

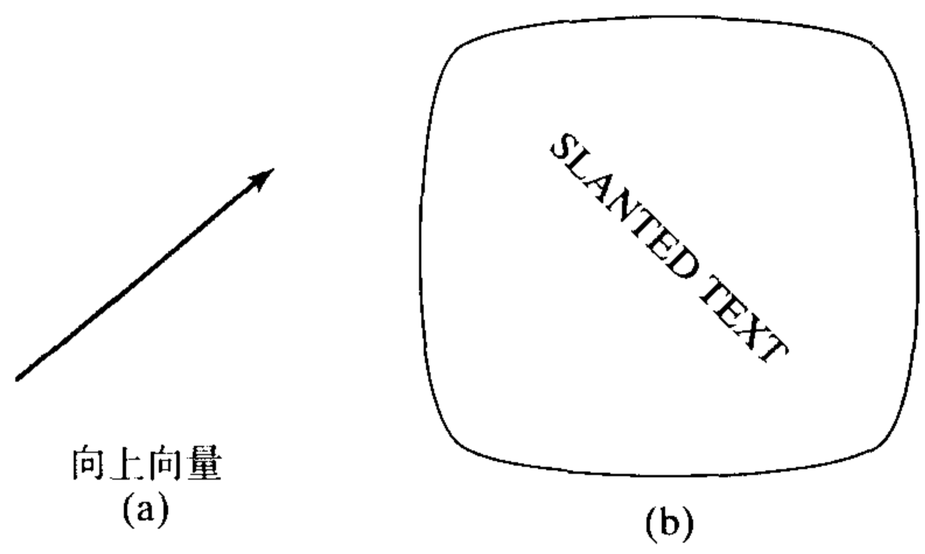


图 4.39　(a)中向上向量的方向控制；(b)中文本的显示方向

将字符串安排成垂直或水平方向的能力在许多应用中是很有用的。图4.40给出了这一例子。字符串还可以按向前或向后顺序显示。使用这些选项的文字显示在图4.41中给出。实现文字路径方向控制的程序按所选择的选项来调整每一字符在帧缓存中的位置。

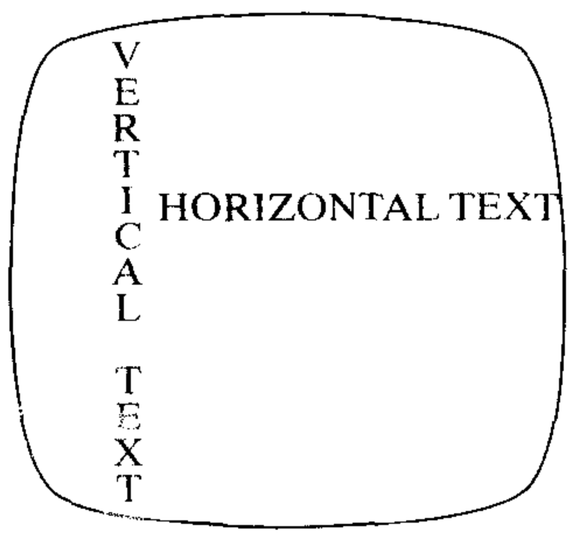


图 4.40　文本路径属性可以设定为生成水平或垂直安排的字符串

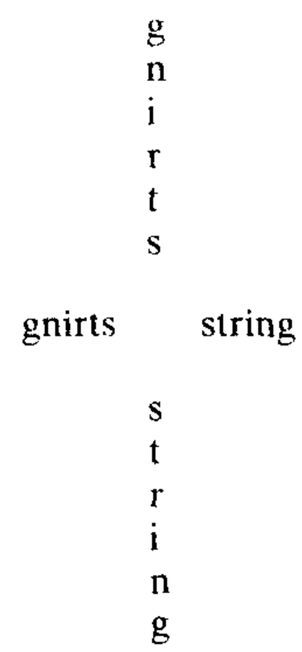


图 4.41　使用四种文本路径：向左、向右、向上和向下所显示的文本

使用向上向量与文本路径说明相结合的方法，可以对字符串定向而生成倾斜的文本。图4.42给出了由多种文字路径设定和45°角向上向量组合导致的字符串方向。图4.43给出了在这一向上向量及文字路径为向下和向右时的字符串显示。

对齐是另一个可能使用的字符串属性，这个属性指定如何依赖参考坐标来定位文本。例如，单个字符可按其基线或字符中心来对齐。图4.44给出了字符水平和垂直对齐的一般位置。字符串也可对齐，图 4.45 给出了水平和垂直文字标记的普通对齐位置。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

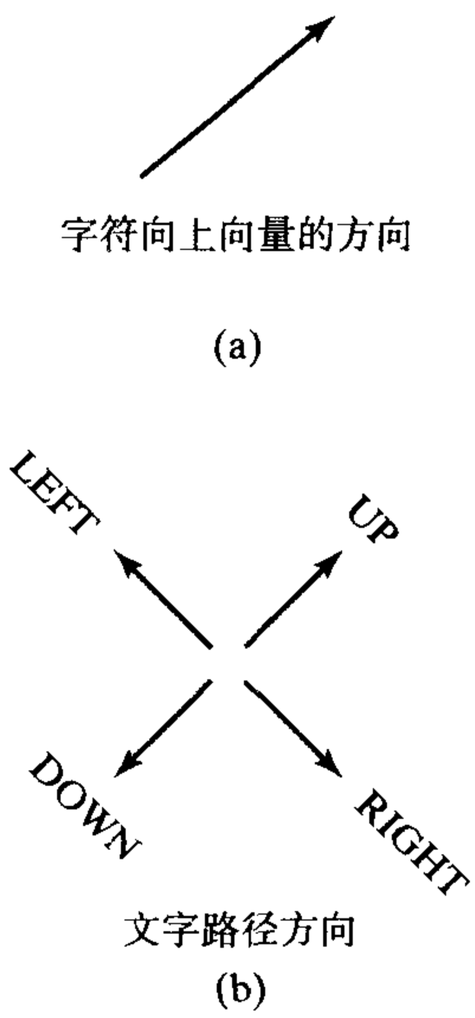


图 4.42 (a)一个向上向量描述；(b)与之配合的文本路径

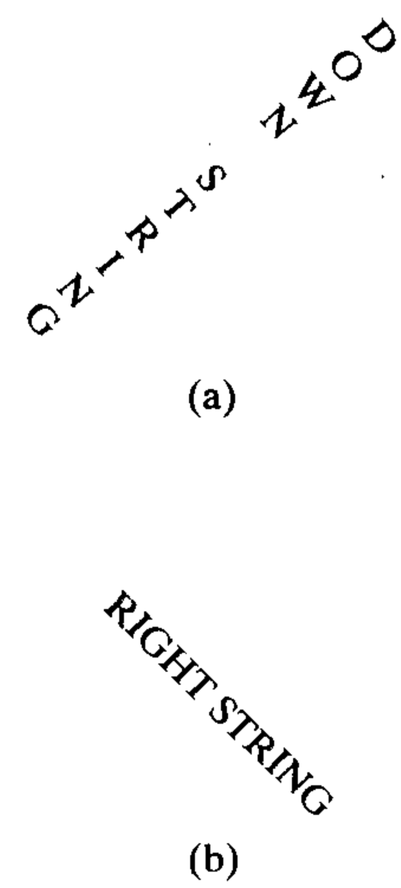


图 4.43 图 4.42 中的 45° 向上向量：(a)与“向下”路径配合的显示文本；(b)与“向右”路径配合的显示文本

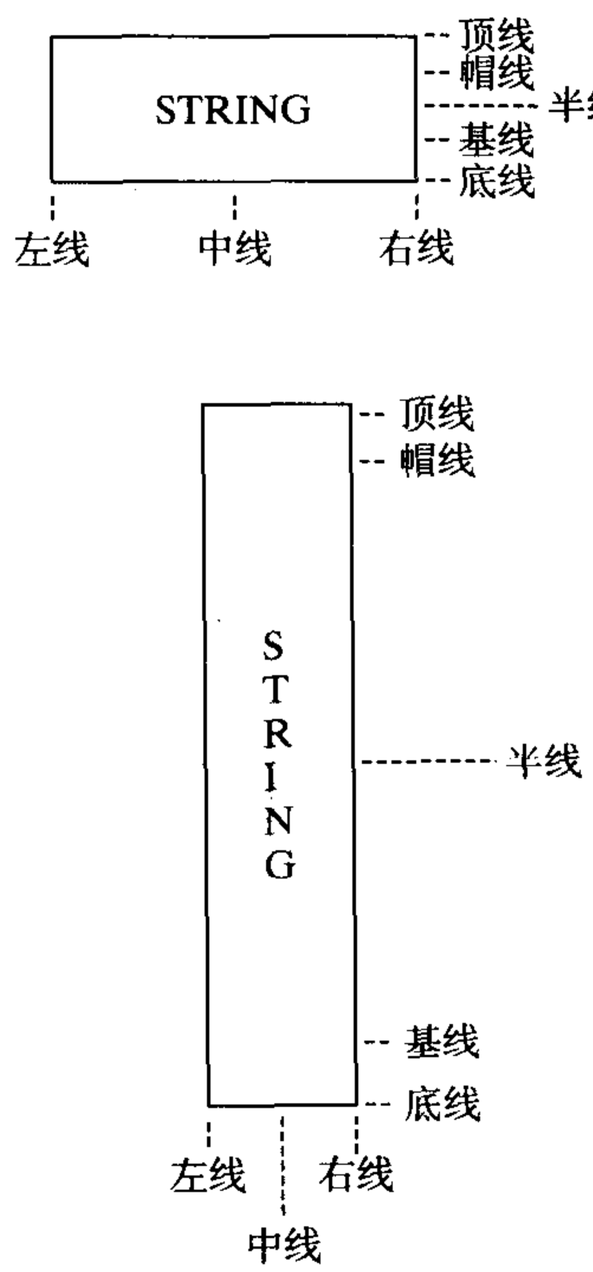


图 4.44 水平和垂直字符串的对齐

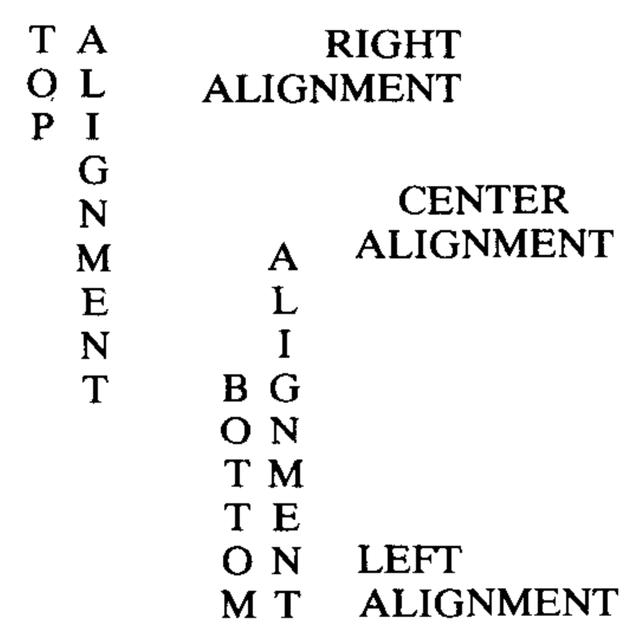


图 4.45 字符串对齐

有些图形软件包中还提供了文字精度属性。该参数指定字符串可用的所有细节及特别处理选项。对于低精度字符串，忽略文字路径等许多属性选项，并使用快速算法通过观察流水线来处理字符。

最后，文字处理函数库常提供一组在各种应用中很有用的特殊字符，如小圆或十字。这些字符常用做网络布局或数据集图中的标记符号。这些标记符号的属性主要有颜色和大小。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

4.16　OpenGL 字符属性函数

使用 OpenGL 软件包显示字符有两种方法。使用核心库中的位图函数来设计字体集，或引用 OpenGL 字符生成函数。GLUT 库中包含显示预定义点阵和笔划字符集的函数。因此，可以设定的字符属性是那些能应用于位图或线段的属性。

对点阵或轮廓字体，显示颜色由当前颜色状态来确定。一般而言，字符的间隔和大小由字符描述确定，如 GLUT_BITMAP_9_BY_15 和 GLUT_STROKE_MONO_ROMAN。但也可以为轮廓字体设定线宽和线型。我们用 glLineWidth 函数指定宽度，用 glLineStipple 函数选择线型。然后 GLUT 笔划字体用指定的当前线宽和线型属性值来显示。

我们可使用第 5 章叙述的变换函数实现其他一些文字显示特色。这些变换子程序允许在二维或三维空间中缩放、定位和旋转 GLUT 笔划字符。另外，三维观察变换（参见第 7 章）可用于生成其他显示效果。

4.17　反走样

由于第3章讨论的取样过程将物体上的坐标点数字化为离散的整数像素位置，因此光栅算法生成的图元显示具有锯齿形或阶梯状外观。这种由于低频取样（不充分取样）而造成的信息失真称为**走样**（aliasing）。可以使用校正不充分取样过程的**反走样**（antialiasing）方法来改善所显示的光栅线的外观。

图4.46给出了不充分取样的效果。为了避免从这种周期性对象中丢失信息，必须把取样频率至少设置为对象中出现的最高频率的两倍，这个频率称为 **Nyquist 取样频率**（或 Nyquist 取样速率）：

$$f_s = 2f_{\max} \tag{4.12}$$

另一种说法是，取样间隔不应超过循环间隔（**Nyquist 取样间隔**）的一半。对于 x 间隔取样，Nyquist 取样间隔 Δx_s 为

$$\Delta x_s = \frac{\Delta x_{\text{cycle}}}{2} \tag{4.13}$$

其中，$\Delta x_{\text{cycle}} = 1/f_{\max}$。在图 4.46 中，取样间隔是循环间隔的一倍半，因此取样间隔至少要三倍大。假如要为这个例子恢复所有的对象信息，那么就需要将图中的取样间隔缩小到三分之一。

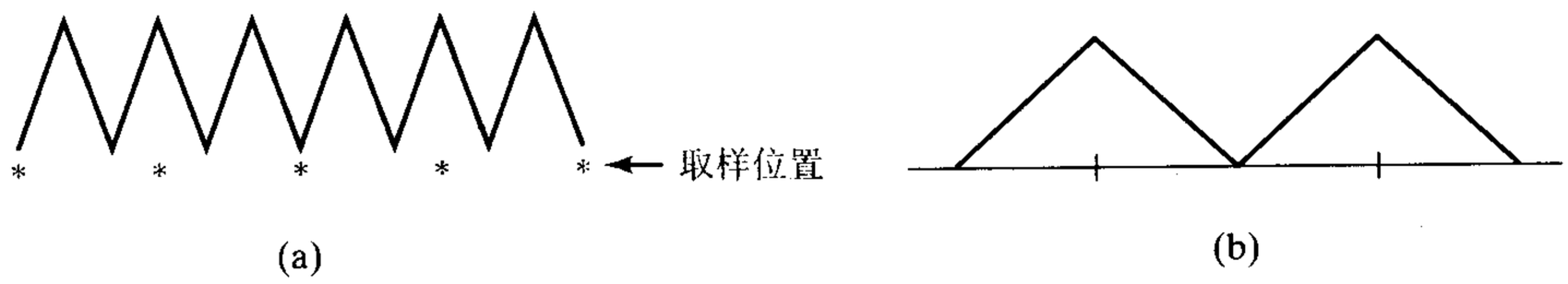


图 4.46　对周期性形状(a)取样，在标记位置产生走样的低频表示(b)

增加光栅系统取样频率的一种方法是简单地以较高分辨率显示对象。但是，即使用当前技术能达到的最高分辨率，仍会在一定范围内出现锯齿形。由于在实现帧缓存的最大容量并且保持刷新频率在每秒 60 帧或以上等方面存在一定的限制，而且要用连续参数精确地表示对象，则需要任意小的取样间隔。因此，除非硬件技术能发展到实现任意大的帧缓存，仅仅增加屏幕分辨率还不能完全解决走样问题。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于能显示两级以上亮度（颜色或灰度等级）的光栅系统，可以使用反走样方法来修改像素亮度。通过适当地改变沿图元边界的像素亮度，可以平滑边界以减小锯齿现象。

一种简单、直接的反走样方法，就是把屏幕看成由比实际更细的网格所覆盖，从而增加取样频率，然后根据这种更细网格，使用取样点来确定每个屏幕像素的合适亮度等级。这种在高分辨率下对于对象特性的取样并在较低分辨率上显示其结果的技术称为**过取样**（supersampling），也称为**后过滤**（postfiltering，因为常用方法包括计算像素网格位置上的亮度，并且将结果进行组合而得到像素亮度）。所显示的像素位置则为覆盖屏幕有限区域的光点，而非无限小的数学点。在画线和填充区域算法中，每个像素的亮度则是由对象边界上单个点的位置所决定的。通过使用过取样方法，可以根据多个点对一个像素总体亮度的作用而得到亮度信息。

可以代替过取样的另一种方法是，通过计算待显示的每个像素在对象上的覆盖区域来确定像素亮度。计算覆盖区域的反走样称为**区域取样**（area sampling；也称为**前滤波**，prefiltering），因为像素亮度是作为一个整体来确定的，所以不用计算子像素亮度。像素覆盖区域通过确定对象边界与单个像素边界的相交位置而得到。

也可以移动像素区域的显示位置而实现光栅对象的反走样，这种技术称为**像素移相**（pixel phasing）。通过与对象几何形状相关的电子束的“微定位”来应用该技术。

4.17.1 直线段的过取样

直线段的过取样可以使用多种方式来完成。对于直线段，可以把每个像素分成一定数目的子像素，并统计沿线路径的子像素数目，然后将每个像素的亮度等级设置为正比于子像素数目的值。图4.47给出了这种方法的一个例子。其中，每个正方形像素区域被分成9个大小相等的正方形子像素，阴影区域表示由Bresenham算法选择的子像素。由于任何像素中可供选择的子像素的最大数目为3，因此这种方法提供零以上的三种亮度设置。对于这个例子，位置(10, 20)的像素将设置为最高亮度（级别3）；(11, 21)和(12, 21)的像素将设置为次高亮度（级别2）；(11, 20)和(12, 22)处像素都将设置为最低亮度（级别1）。因此，线亮度分布在较大数目的像素上，并且通过在阶梯状（水平线之间）附近显示有些模糊的线路径，从而平滑阶梯状效果。如果要使用更多的亮度等级来实现直线段的反走样，就需要增加每个像素中的取样位置数。例如，16个子像素给出零以上的4个亮度等级；25个子像素给出5个等级；等等。

在图4.47的过取样例子中，考虑了有限尺寸的像素区域，但是我们把直线处理成具有零宽度的数学实体。实际上，显示的直线具有与像素大约相等的宽度。假如考虑线的有限宽度，则可以通过将每个像素亮度设置成正比于表示线区域的多边形内的子像素数目来完成过取样。如果子像素的左下角在多边形的边界内，那么可以把该子像素当做在线内。这种过取样程序的一个好处是，每个像素可能的亮度等级数目等于像素区域内子像素的总数。对于图4.47中的例子，可以通过对平行于线路径的多边形边界进行定位，从而以有限宽度来表示这样的直线，如图4.48所示。并且，每个像素现在可设置成零以上的9个亮度级别之一。

对有限宽直线段进行过取样的另一个优点是，总的线亮度分布在更多的像素上。在图4.48中，网格位置(10, 21)上的像素打开（以亮度等级2），并且吸收了位置(10, 21)下方和左方的像素的作用。同样，假如有颜色显示，我们也可以扩充此方法以考虑背景色。一条特定的直线可能穿过许多不同的颜色区域，我们可取平均子像素亮度来设置像素颜色。例如，一个特定像素区域内的5个子像素被确定在红线的边界内，而其余4个像素则落在蓝色背景区域内，我们可以将这个像素的颜色计算为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

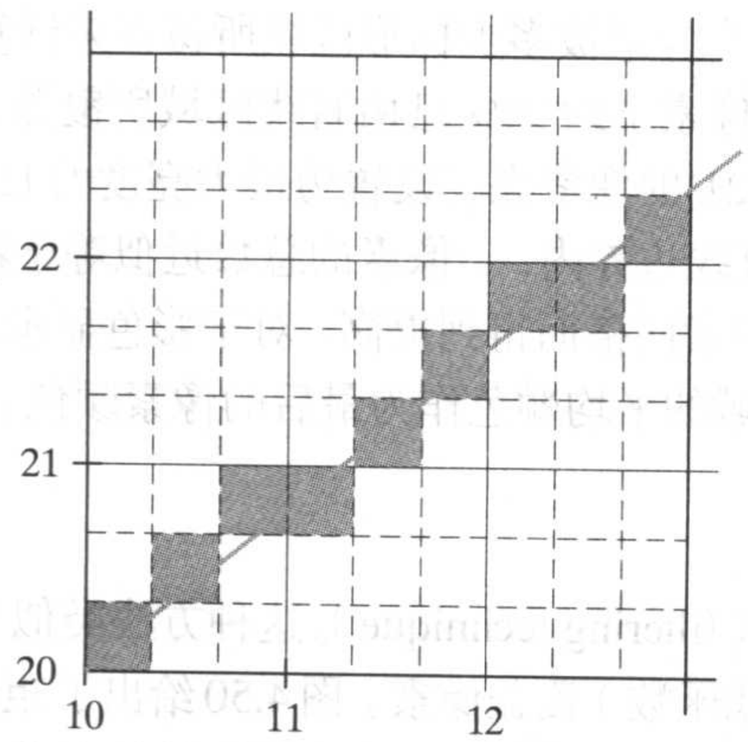


图 4.47　沿左端点在屏幕坐标(10, 20)处的直线段的过取样子像素位置

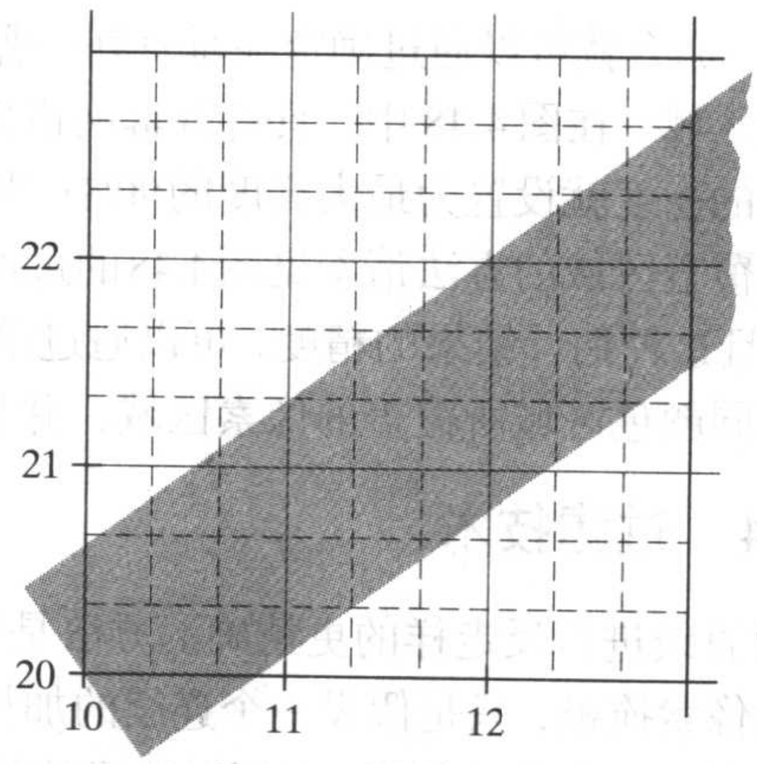


图 4.48　与有限宽线段内部相关的过取样子像素位置

$$\text{pixel}_{\text{color}} = \frac{(5 \cdot \text{red} + 4 \cdot \text{blue})}{9}$$

通过对有限宽线段进行过取样，鉴别内部子像素比起简单地确定沿线路径的子像素需要更多的计算。我们同样要考虑相对于线路径的线边界的定位。这种定位取决于直线的斜率。对于45°的直线，线路径在多边形区域的中央；但对于水平线或垂直线，都要求线路径是多边形的边界之一。例如，经过网格坐标(10, 20)的水平线将由水平网格线 $y = 20$ 和 $y = 21$ 表示为多边形边界。类似地，表示通过(10, 20)的垂直线的多边形，具有沿网格线 $x = 10$ 和 $x = 11$ 的垂直边界。对于具有斜率 $|m| < 1$ 的直线段，可以适当地将数学线路径定位在最接近于下面的多边形边界；而对于斜率 $|m| > 1$ 的直线段，则将这条线路径定位在较接近于上面的多边形边界。

4.17.2　子像素的加权掩模

过取样算法经常在实现时将更大的权值赋给接近于像素区域中心的子像素，因为我们希望这些子像素在确定像素的整体亮度中可以实现更重要的作用。图4.49给出了3 × 3像素部分所采用的加权方案。其中，中心子像素的加权是角子像素的4倍，是其他像素的2倍。然后对9个子像素的每个网格所计算出的亮度进行平均。因此，中心子像素的加权系数为1/4，顶部和底部及两侧子像素的加权系数为1/8，而角子像素的加权系数为1/16。指定子像素的相对重要性的数值数组有时称为加权掩模（weighting mask）。也可以为较大的子像素网格建立类似的掩摸。而且，经常扩展这些掩模以包含来自相邻像素中子像素的作用，从而对相邻像素进行平均以获得亮度。

1	2	1
2	4	2
1	2	1

图 4.49　3 × 3 子像素网格的相对权值

4.17.3　直线段的区域取样

通过将每个像素亮度设置为正比于像素与有限宽直线段的重叠区域，可以完成对直线段的区域取样。将直线段看成矩形，而将两相邻的垂直（或两相邻的水平）屏幕网格线间的直线区域看做一

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

个梯形，那么就可以通过确定在垂直列（或水平行）中每个像素被多少梯形区域所覆盖来计算像素的重叠区域。在图4.48中，具有屏幕网格坐标(10, 20)的像素中约90%已由直线区域所覆盖，那么该像素的亮度就设置为最大亮度的90%；类似地，在(10, 21)的像素亮度设置为最大亮度的15%。估计像素覆盖区域的方法请参见图4.48的过取样例子。在直线边界内，子像素的总数近似等于覆盖区域，并且这种估计结果的精度，可以通过采用更细的子像素网格而得到提高。对于彩色显示，则计算被不同颜色区域所覆盖的像素区域，并且将各覆盖区域的平均颜色作为最后的像素颜色。

4.17.4 过滤技术

对直线进行反走样的更精确的方法是采用**过滤技术**（filtering technique）。这种方法类似于应用加权的像素掩模，只是假设一个连续的加权表面（或过滤函数）覆盖像素。图4.50给出了矩形、圆锥和高斯过滤函数的例子。应用过滤函数的方法类似于应用加权掩模，但过滤函数是集成像素曲面来得到加权的平均亮度。为了减少计算量，经常使用查表法来得到整数值。

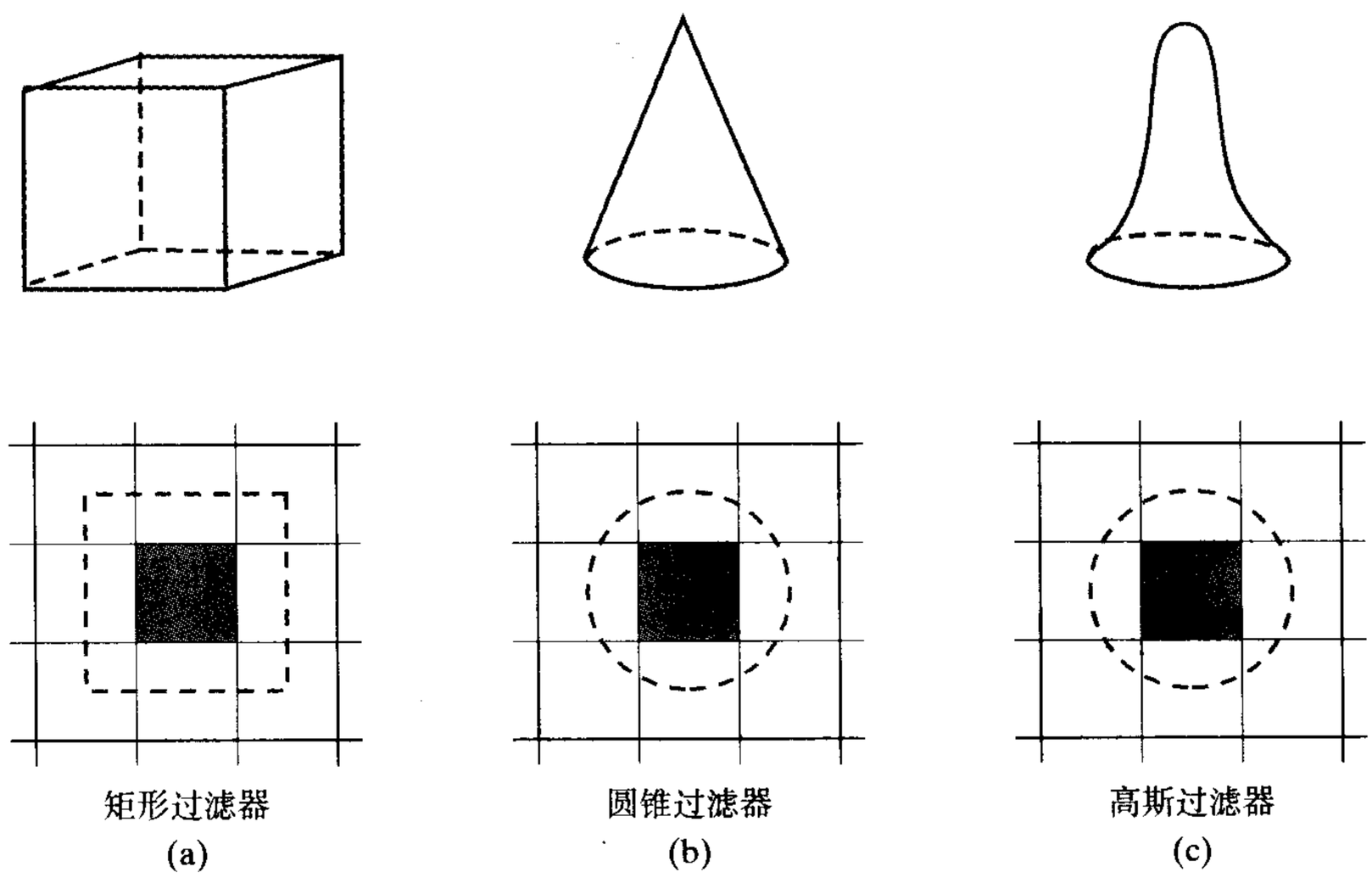


图4.50 用于反走样线路径的常用过滤函数，每个过滤器的体积被规范化为1.0，且用任一子像素位置上的高度给出该子像素的相对权值

4.17.5 像素移相

在能够对屏幕网格内的子像素位置进行编址的光栅系统上，可以使用像素移相来进行对象的反走样。通过将像素位置移动（微定位）到更接近线路径来实现显示线段的平滑。结合像素移相（pixel phasing）的系统设计成可以根据像素直径的小数部分来移动电子束。典型情况中，电子束根据像素直径的1/4、1/2、3/4进行移动，从而绘制接近于直线或对象边界真实路径的点。有些系统也允许对单个像素的尺寸进行调整，以作为分配亮度的附加方法。图4.51给出了在各种线路径上像素移相的反走样效果。

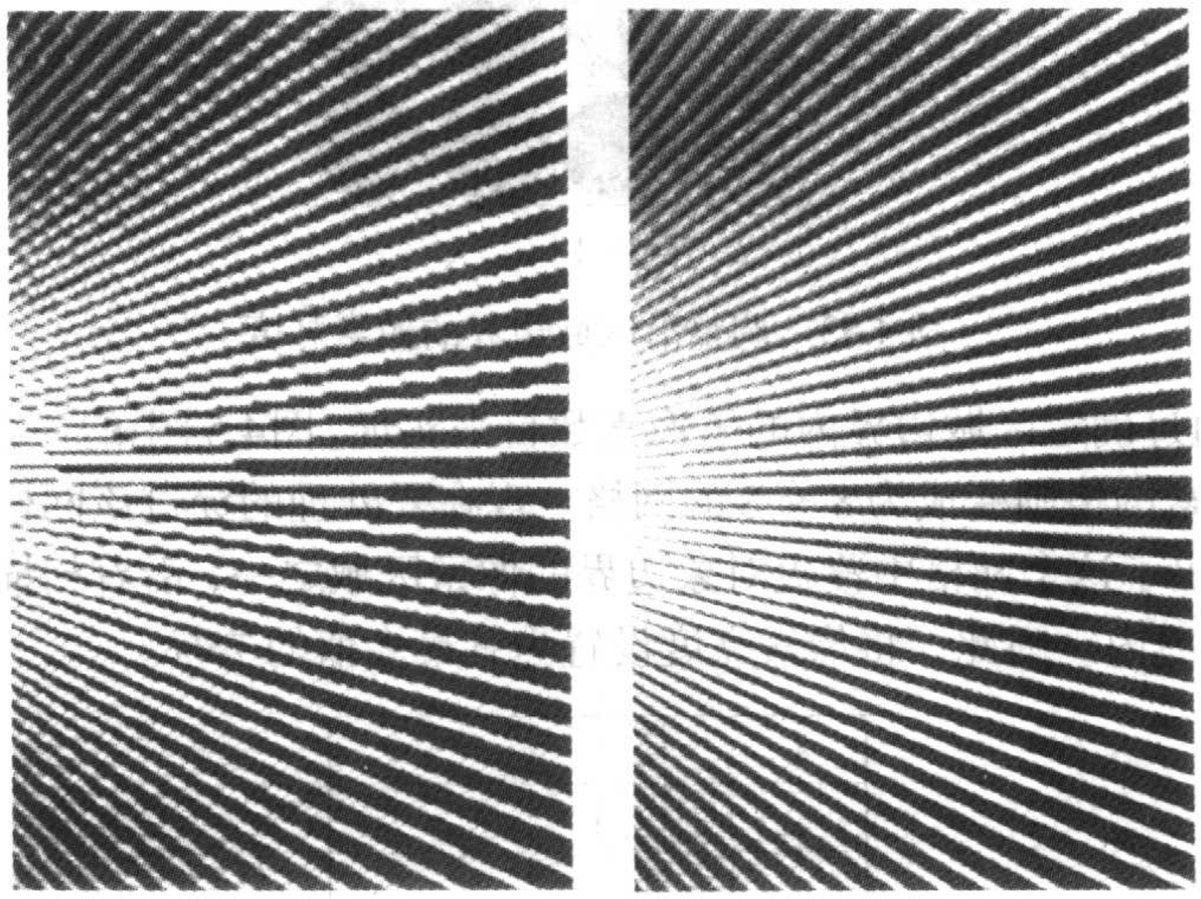
4.17.6 直线亮度差的校正

为了减轻阶梯状效应，对直线进行反走样也为如图4.52所示的另一种光栅效果提供了校正。使用相同数目像素所绘制的两条线，对角线还是比水平线长$\sqrt{2}$倍。例如，当水平线的长度为10 cm

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

时，对角线的长度超过 14 cm。这导致的视觉效果是对角线显得比水平线要暗，因为对角线以更低的单位长度亮度进行显示。画线算法通过按照每条线的斜率来调整其亮度，就可以对这种效果进行校正。水平和垂直线将以最低的亮度显示，而 45° 线则以最高亮度显示。一旦将反走样技术应用于显示，就可以自动校正亮度。当考虑直线段的有限宽度时，可以把整体线段显示亮度调整为正比于其长度。



(a)　(b)

图 4.51　(a)在 Merlin 9200 系统上绘制的锯齿形线；(b)使用像素移相的反走样技术进行平滑后的效果，这种技术增加了系统上可编址点的数目，从 768 × 576 到 3072 × 2304（Peritek 公司提供）

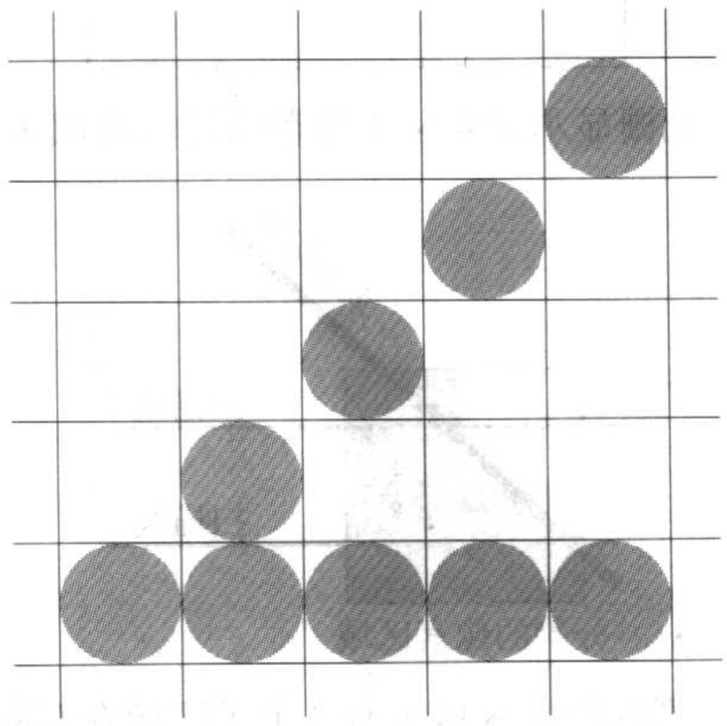


图 4.52　用相同数目像素显示的不等长直线

4.17.7　区域边界的反走样

直线的反走样概念也可以用于区域的边界，从而消除其锯齿形的外貌。我们可以将这种程序加入到扫描线算法中，在生成区域时来平滑区域轮廓。

假如系统具有允许像素重定位的功能，那么就可以将边界像素位置调整到更靠近区域边界来实现对区域边界的平滑处理。其他方法则是根据边界内像素区域的百分比来调整每个边界位置上的像素亮度。在图 4.53 中，位置(x, y)上的像素有大约一半的区域在多边形边界内。因此，该位置处的

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

亮度将调整到其设定值的一半。沿边界的下一个位置$(x+1, y+1)$的亮度则调整到约为其设定值的1/3。以像素区域覆盖率为基础，类似的调整方法可以应用于沿边界的其他亮度值。

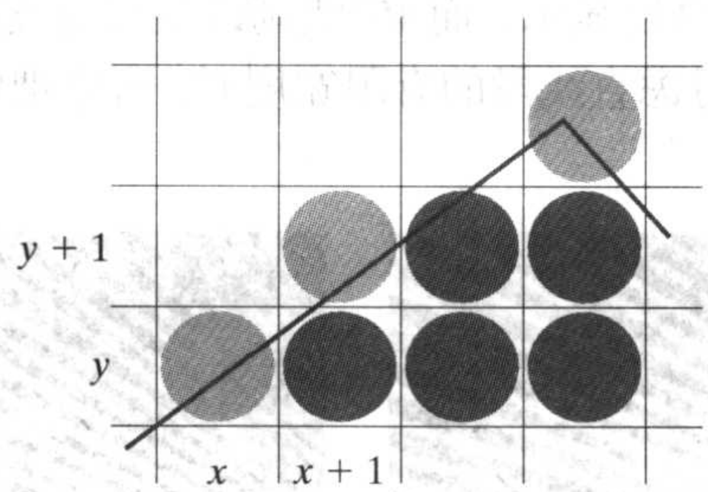


图 4.53　调整沿区域边界的像素亮度

过取样方法可通过确定区域边界内的子像素数目来实现。图4.54中的一个像素被分割成4个子区域，原4 × 4的像素网格则变成为8 × 8的网格。现在要处理的穿过该网格的扫描线是8条，而不是4条。图4.55给出了这个网格中覆盖对象边界的像素区域之一，沿着扫描线，可以确定子像素区域的3个区域在边界内。因此，将像素亮度设置为其最大值的75%。

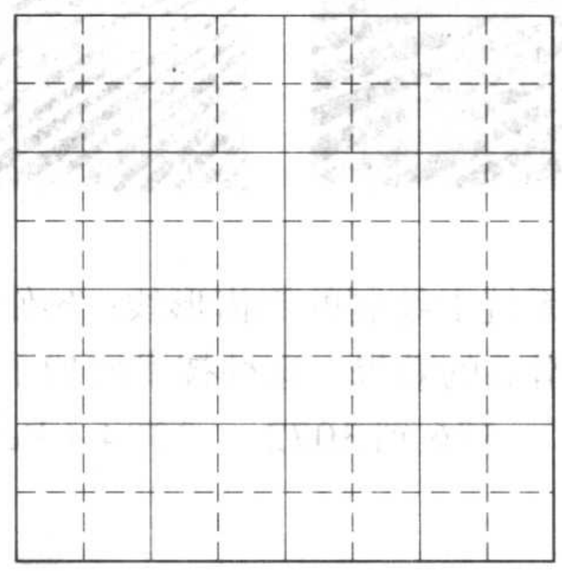


图 4.54　光栅显示的4 × 4像素段将分割成8 × 8网格

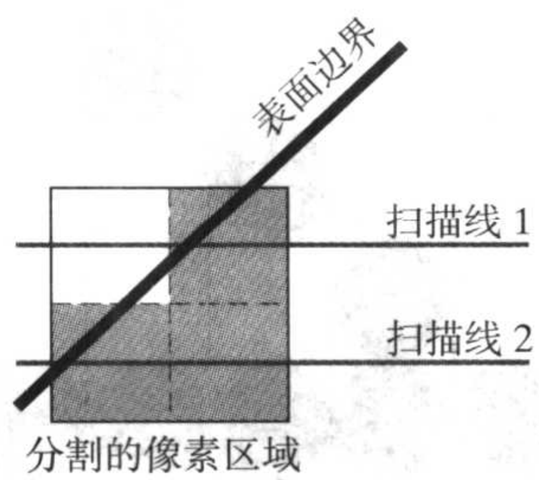


图 4.55　分割的像素区域中有3个子区域在对象边界线内

另一种由Pitteway和Watkinson提出的确定边界内像素区域百分比的方法，是以中点算法为基础的。这个算法通过测试两像素间的中间位置，确定哪个像素更接近于直线而选择沿扫描线的下一个像素。类似于Bresenham算法，可以建立决策参数p，其符号可以表明下面的两个候选像素中哪一个更接近于直线。通过对p形式的略微修改，就可以得到被对象覆盖的当前像素区域的百分比。

我们首先考虑斜率m在0到1之间的画线算法。在图4.56中，线路径显示在像素网格之上。假设已经绘制了(x_k, y_k)上的像素，那么最接近$x = x_k+1$上直线的下一个像素可能是y_k上的像素，或是y_k+1上的像素。我们使用下列计算来确定哪一个更接近直线：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$y - y_{\text{mid}} = [m(x_k + 1) + b] - (y_k + 0.5) \tag{4.14}$$

这给出了从线段上的实际 y 坐标到位置 y_k 和 y_k+1 间的中点的垂直距离。假如差为负，那么 y_k 上的像素更接近直线；假如差为正，则 y_k+1 上的像素更接近。我们可以通过加上 $1-m$ 来调整这个计算，从而使它产生一个 0 到 1 之间的正数：

$$p = [m(x_k + 1) + b] - (y_k + 0.5) + (1 - m) \tag{4.15}$$

假如 $p < 1-m$，则 y_k 上的像素更接近；假如 $p > 1-m$，则 y_k+1 上的像素更接近。

参数 p 也能计算区域覆盖当前像素的实际量。对于图 4.57 中 (x_k, y_k) 处的像素，像素的内部有一个区域可计算为

$$\text{area} = m \cdot x_k + b - y_k + 0.5 \tag{4.16}$$

这个对 (x_k, y_k) 上的覆盖区域表达式，与等式（4.15）中决策参数 p 的表达式是相同的。因此，通过计算 p 值来确定沿多边形边界的下一个位置，也可以确定对当前像素覆盖区域的百分比。

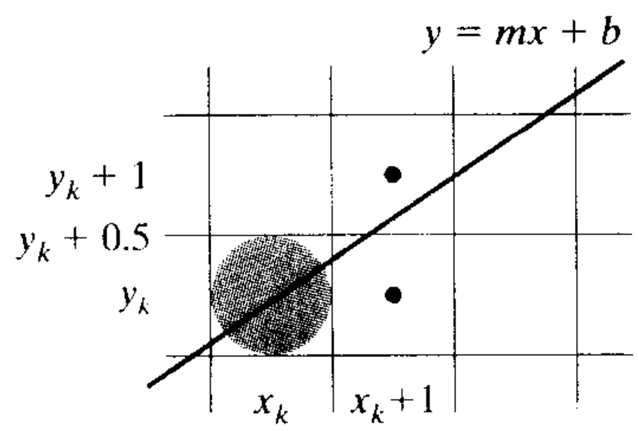


图 4.56　通过像素网格部分的区域边界

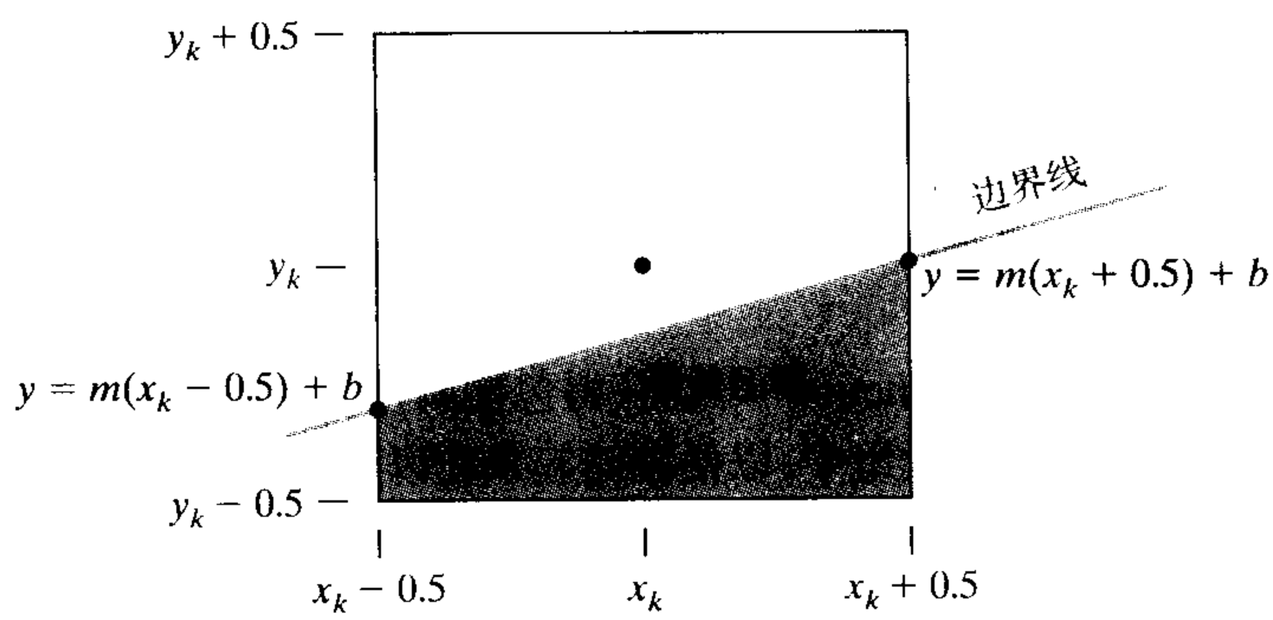


图 4.57　中心在 (x_k, y_k)、有多边形内部区域的像素矩形的覆盖区域

我们可将这个算法一般化，以适应具有负斜率和斜率大于 1 的直线段。对决策参数 p 的这种计算可以加入到中点线算法中，从而沿多边形的边对像素位置进行定位，且并行地调整沿边界线的像素亮度。同样，我们可以调整计算，将像素坐标指向左下角并保持区域比例，如 3.13 节中讨论的那样。

在多边形顶点处和对于很小的多边形（如图 4.58 所示），有多于一条边通过像素区域。对于这些情况，我们需要处理所有通过像素的边并确定正确的内部区域来修改 Pitteway-Watkinson 算法。

对直线的反走样所讨论的过滤技术也能应用于区域边。同样，各种反走样方法可用于多边形区域或具有曲线边界的区域。边界方程则用来估计像素区域与将要显示的区域的覆盖量。而相关性技术则用于沿扫描线之间以简化计算。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

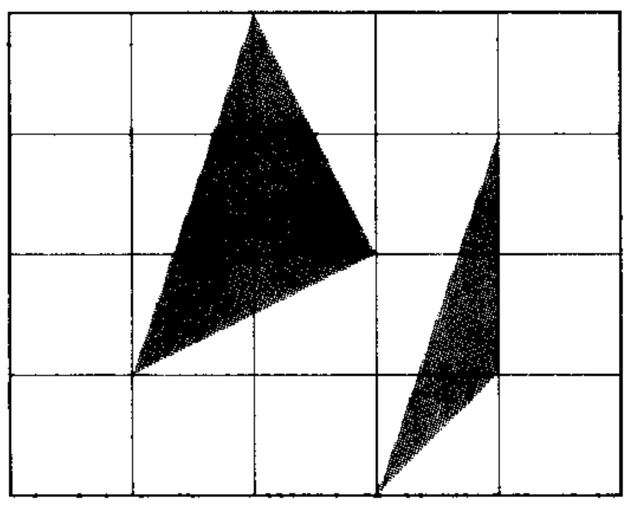


图 4.58 具有多条通过单个像素区域的边界线的多边形

4.18 OpenGL 反走样函数

使用下列函数可激活 OpenGL 的反走样子程序：

```
glEnable (primitiveType);
```

其中参数 primitiveType 被赋以符号常量 GL_POINT_SMOOTH、GL_LINE_SMOOTH 或 GL_POLYGON_SMOOTH。假定我们用 RGBA 模式指定颜色，则同样需要激活 OpenGL 颜色调和操作。

```
glEnable (GL_BLEND);
```

接下来，通过下列函数来使用 4.3 节描述的颜色调和方法：

```
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

如果在对象的颜色描述中使用大的 α 值则平滑操作比较高效。

在使用颜色表时也可以应用反走样。但是，在这种颜色模式中，我们必须创建一个颜色斜坡，即从背景色到对象色逐步变化的一张表。该颜色斜坡用来进行对象边界的反走样。

4.19 OpenGL 查询函数

使用OpenGL**查询函数**(query function)可以获得包括属性设定在内的任一状态参数的当前值。这些函数将指定状态值复制到一个数组中，以便存储起来在以后使用或检查当前的系统状态是否有错误。

对于当前属性值，需要使用一个合适的“glGet”函数，例如

```
glGetBooleanv ( )          glGetFloatv ( )
glGetIntegerv ( )          glGetDoublev ( )
```

在其中的每一个函数中，我们指定两个变量。第一个变量是标识一个属性或状态参数的OpenGL符号常量。第二个变量是一个指针，指向由函数名指出的数据类型的一个数组。例如，使用下列语句可以获取当前 RGBA 浮点颜色设定。

```
glGetFloatv (GL_CURRENT_COLOR, colorValues);
```

当前颜色分量被传递给数组colorValues。要获取整数的当前颜色分量，必须引用glGetIntegerv函数。有些情况下，必须在返回指定数据类型时进行类型转换。

GL_POINT_SIZE、GL_LINE_WIDTH 和 GL_CURRENT_RASTER_POSITION 等其他一些 OpenGL 常量，可用于这些函数以获取当前状态值。常量 GL_POINT_SIZE_RANGE 和 GL_LINE_WIDTH_RANGE 可用来支持对点的大小和线的宽度的检查。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

虽然可以通过 glGet 函数取回和重用单个属性，但 OpenGL 还提供其他一些函数来存储一组属性并重新使用它们的值。这些存储当前属性设定的函数的使用将在下一节中讨论。

还有许多对查询有用的其他状态和系统参数，例如确定在特定系统的帧缓存中所提供的每一像素的位数，了解每一独立的颜色分量的有效位数：

```
glGetIntegerv (GL_RED_BITS, redBitSize);
```

这里，数组 redBitSize 被赋以每一缓存（帧缓存、深度缓存、累计缓存和模板缓存）中红颜色的有效位数。同样，可以使用 GL_GREEN_BITS、GL_BLUE_BITS、GL_ALPHA_BITS 或 GL_INDEX_BITS 来查询其他颜色的位数。

我们还可以了解边标志是否已设定，多边形的面是否已标为前向面或后向面，以及系统是否支持双缓存。还可以询问特定的子程序（如颜色调和、线画或反走样）是否激活或关闭了。

4.20　OpenGL 属性组

属性和其他状态参数按**属性组**（attribute group）进行组织。每一组包括相关的状态参数集合。例如，**点属性组**（point-attribute group）包括了大小和点的平滑（反走样）参数，而**线属性组**（line-attribute group）包括了宽度、模板状态、模板图案、模板重复计数及线段光滑状态等。类似地，**多边形属性组**（polygon-attribute group）包括了 11 种多边形参数，如填充模式、前向面标志及多边形平滑状态。因为颜色是所有图元所共有的一个属性，所以它有单独的属性组。而有些参数被包含在多个组内。

OpenGL中有约20个不同的属性组，可以使用一个函数来保存或重新设定一个或多个组内的所有参数。保存一个指定属性组所有参数的工作由下列命令实现。

```
glPushAttrib (attrGroup);
```

参数 attrGroup 用标识一个属性组的 OpenGL 符号常量来赋值，如 GL_POINT_BIT、GL_LINE_BIT 或 GL_POLYGON_BIT。为了保存颜色参数，使用符号常量 GL_CURRENT_BIT。我们可以用符号常量 GL_ALL_ATTRIB_BITS 来存储所有属性组中的所有状态参数。glPushAttrib 函数将指定组的所有参数放进**属性栈**（attribute stack）。

利用逻辑 OR 操作，可以组合符号常量，从而将参数存储在两个或更多的组中。下面的语句给出了属性栈上用于点、直线和多边形的参数。

```
glPushAttrib (GL_POINT_BIT | GL_LINE_BIT | GL_POLYGON_BIT);
```

将一组状态参数存储起来后，我们可以用下列函数将属性栈的所有值进行重建。

```
glPopAttrib ( );
```

在 glPopAttrib 函数中不使用任何变量，因为它使用栈中的所有值来设定 OpenGL 的当前状态。

这些保存和重建状态参数的命令使用一个服务器属性栈（server attribute stack）。OpenGL 中还有一个客户属性栈（client attribute stack）用来保存和重建客户状态参数。存取这些堆栈的函数是 glPushClientAttrib 和 glPopClientAttrib。客户属性栈只有两个：一个用于像素存储器模式而另一个用于顶点数组。像素存储器参数包括字节对齐信息和用于存储显示中的子图的数组类型。顶点数组参数给出当前顶点数组状态的信息，如各种数组的激活 / 停止状态。

4.21　小结

属性控制图元的显示特征。在许多图形系统中，属性值以状态变量形式存储而图元使用当前状态值生成。当改变一个状态变量的值时，它仅仅影响在改变后定义的那些图元。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

颜色是所有图元的公共属性，它常用RGB（或RGBA）分量来描述。红、绿和蓝颜色值存储在帧缓存中，它们用来控制RGB监视器的三个电子枪。颜色选择也可通过使用颜色表来实现。在这种情况下，帧缓存中的一个颜色只是一张表的一个索引，而该索引位置的表项中存储特定的一组RGB颜色值。颜色表对数据可视化和图像处理应用很有用，也可用来在不要求增加帧缓存容量的前提下提供大的可选颜色范围。计算机图形软件包常提供对使用颜色表或直接在帧缓存中存储颜色的选择。

基本的点属性是颜色和大小。在光栅系统中，各种大小的点用正方形像素阵列来显示。线属性有颜色、宽度和类型。线宽用标准的单像素宽度的倍数来指定。线型属性包括实线、划线和点线以及各种画刷或画笔类型。这些属性可应用于直线段和曲线。

填充区域属性包括实心颜色填充、填充图案填充或仅仅显示区域边界的空心填充。各种图案填充用颜色阵列来描述，然后映射到区域内部。扫描线方法常用于填充多边形、圆和椭圆。内点填充用于扫描线与边界的每一对交点之间的像素位置从左往右进行。对于多边形来说，扫描线与其相交于顶点可导致两个交点。缩短某些边可解决这一问题。如果把填充区限定为凸多边形则扫描线填充算法可得到简化。如果场景中所有填充区域都是三角形则扫描线填充算法可进一步简化。沿每一扫描线的内部像素按照填充属性描述来赋以适当的颜色。绘画程序通常用边界填充方法或泛滥填充方法显示填充区。这两种方法都需要一个起始内点。然后从起始点开始逐个像素绘制直到边界。

区域也可以用颜色调和来填充。这种类型的填充应用于反走样和绘画软件包。软填充程序为区域提供了一个与先前填充颜色有相同变化的新填充颜色。线性软填充算法是这种方法的一个例子，该算法假设先前的填充是前景色和背景色的线性组合，然后从帧缓存设置中确定这种相同的线性关系，并用于以新颜色重新对区域涂色。

字符可以按照不同颜色、大小、间隔和方向进行显示。为了设置字符串的方向，我们选择字符向上向量的方向和文本路径的方向。此外，我们可以设置相对于起始坐标位置的文字串对齐方式。标记符号可以通过选择各种大小和颜色的标准字符和专用符号进行显示。

由于扫描转换是光栅系统的一个数字化过程，因此显示图元有阶梯效应。这是由于将坐标值取整到像素位置造成的信息低取样所形成的。我们可通过应用调整像素强度的反走样过程来改善光栅图元的外貌。过取样是一种实现反走样的方法。即将每一像素看作为子像素的组合并计算每一子像素的强度及所有子像素的平均值。我们还可按子像素的位置来确定其贡献权值，给中心以最高权值。另一种选择是，进行区域取样并确定一个屏幕像素的区域覆盖百分比，然后设定与该百分比成比例的像素亮度。另一种反走样方法是构造能移动像素位置的硬件配置。

OpenGL中，图元的属性值由状态变量来维护。一次属性设定保证在其后定义的图元都有效，直到该属性值再次被改变。改变一个属性值不会影响前面的图元显示。在OpenGL中可以用RGB（RGBA）颜色模式或使用颜色表索引所选颜色的颜色索引模式来指定颜色。我们也可以使用α颜色分量来调和颜色值。我们还可以在颜色数组中指定与顶点数组相邻的值。除颜色外，OpenGL还提供选择点大小、线宽、线型和多边形顶点填充类型的函数，并提供显示用一组边或一组顶点定义的多边形填充区域的函数。我们也可以从显示中消除多边形的指定边，可以颠倒前向面和后向面的描述。在OpenGL中字符串可使用GLUT中的点阵或子程序来生成。为显示GLUT字符而设定的属性包括颜色、字体、大小、间隔、线宽和线型。OpenGL函数库中也提供对输出图元显示反走样的函数。我们可以用查询函数来获得状态变量的当前值，也可以用一个函数获得一组OpenGL属性的所有值。

表4.2总结了本章讨论的OpenGL属性函数。另外，该表列出了某些与属性相关的函数。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

表 4.2　OpenGL 属性函数小结

函数	叙述
glutInitDisplayMode	选择颜色模式，GLUT_RGB 或 GLUT_INDEX
glColor*	指定一个 RGB 或 RGBA 颜色
glIndex*	用颜色表索引指定一个颜色
glutSetColor (index, r, g, b);	将一个颜色装入一个颜色表位置
glEnable (GL_BLEND);	激活颜色调和
glBlendFunc (sFact, dFact);	指定颜色调和因子
glEnableClientState (GL_COLOR_ARRAY);	激活 OpenGL 颜色数组设施
glColorPointer (size, type, stride, array);	指定一个 RGB 颜色数组
glIndexPointer (type, stride, array);	用颜色索引模式指定一个颜色数组
glPointSize (size)	指定点的大小
glLineWidth (width);	指定线宽
glEnable (GL_LINE_STIPPLE);	激活线型
glEnable (GL_POLYGON_STIPPLE);	激活填充模式
glLineStipple (repeat, pattern);	指定线型图案
glPolygonStipple (pattern);	指定填充模式图案
glPolygonMode	以一组边或一组顶点方式显示前向面或后向面
glEdgeFlag	将填充多边形边标志设为 GL_TRUE 或 GL_FALSE 来确定一条边的显示状态
glFrontFace	将前向面顶点次序指定为 GL_CCW 或 GL_CW
glEnable	用 GL_POINT_SMOOTH、GL_LINE_SMOOTH 或 GL_POLYGON_SMOOTH 激活反走样（也需激活颜色调和）
glGet**	多种查询函数，要求指定数据类型、状态参数的符号常量和数组指针
glPushAttrib	将所有状态参数存入指定的属性组
glPopAttrib ( );	将最后存储的所有状态参数进行重建

参考文献

软填充技术请参见 Fishkin and Barsky(1984)。反走样技术参见 Pitteway and Watinson (1980)、Crow (1981)、Turkowski (1982)、Fujimoto and Iwata(1983)、Korein and Badler (1983)、Kirk and Arvo (1991)以及 Wu (1991)。有关颜色和灰度的讨论请参见 Crow (1978)。其他关于属性和状态参数的讨论参见 Glassner(1990)、Arvo(1991)、Kirk(1992)、Heckbert(1994)和 Paeth(1995)。

使用 OpenGL 属性函数的编程示例请参见 Woo, Neider, Davis, and Shreiner(1999)。有关 OpenGL 属性函数的完整清单可以在Shreiner(2000)中找到，并且在Kilgard(1996)中讨论了GLUT的字符属性。

练习题

4.1　使用 glutSetColor 函数为一组输入的颜色值建立一个颜色表。

4.2　使用顶点和颜色数组建立一个至少包括六个二维对象的场景描述。

4.3　编写显示上题中的二维场景描述的程序。

4.4　使用顶点和颜色数组建立一个至少包括四个三维对象的场景描述。

4.5　编写显示二维、灰度“云”场景的程序，其中云的形状用蓝色天空背景的点图案描述。云的亮和暗的区域用可变尺寸及间隔的点来建模。（例如，很亮的区域用小的宽间隔、浅灰色点。类似地，暗的区域用大的、更靠近的深灰色点建模。）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

4.6 修改上题中的程序，将云显示成在日出或日落时可能看到的红色和黄色。为了实现真实感，对这些点使用不同的红色、黄色（也许绿色）浓淡。

4.7 修改 Bresenham 画线算法，以实现一个通用的显示实线、划线和点线的线型函数。

4.8 使用中点线算法实现显示实线、划线和点线的线型函数。

4.9 设计实现线型函数的并行方法。

4.10 设计实现线宽函数的并行方法。

4.11 由两个端点和线宽指定的线段可以转换成有四个顶点的矩形，然后用扫描线算法显示。请给出一个有效的算法，使用作为输入参数的线段端点和线宽来计算，以定义这个矩形的四个顶点。

4.12 实现画线程序中的线宽功能，从而使三种线宽中的任何一个都能显示。

4.13 编写一个程序，输出在相同 x 坐标范围上定义的三个数据组的线状图。对程序的输入包括三组数据值和图的标签。需缩放数据组以适合指定的显示窗口区域。每一数据集用不同的线型绘图。

4.14 修改上题中的程序，使用不同的颜色及不同的线型来绘制三个数据集。

4.15 建立一个用方帽、圆帽或突方帽粗线的显示算法，这些选择以菜单选项的形式提供。

4.16 设计一个算法来显示具有斜角连接、圆角连接或斜切连接的粗折线，这种选择以菜单选项的形式提供。

4.17 修改 4.8 节显示数据折线图的程序段，使线宽参数传递给过程 linePlot。

4.18 修改 4.8 节显示数据折线图的程序段，使线型参数传递给过程 linePlot。

4.19 修改 4.8 节显示数据折线图的程序段，使之从数据文件中输入数据。

4.20 修改4.8 节显示数据折线图的程序段，使之从数据文件中输入数据。并让程序提供坐标轴的标签和屏幕显示区的坐标。缩放数据集使其适合显示窗口的坐标范围，且每一条绘制的线使用不同的线型、线宽和颜色。

4.21 实现画线过程的画笔和画刷的菜单选项，至少包括两个选项：圆帽和方帽。

4.22 修改画线算法，使输出的亮度按其斜率设置，即按照斜率值调整像素亮度，使直线以每单位长度的相同亮度进行显示。

4.23 定义并实现控制显示椭圆线型（实线、划线、点线）的函数。

4.24 定义并实现设置显示椭圆线宽的函数。

4.25 编写一个程序，可以在任意指定屏幕区域内显示条状图形。其中的输入包括数据组、坐标轴的标签和屏幕区域的坐标。数据组要进行缩放以适合所指定的屏幕区域，直方图要以指定的颜色或图案进行显示。

4.26 编写一个程序，显示在相同 x 坐标范围上定义的两个数据组，数据值缩放到适合显示屏幕的指定区域。其中的一个数据组的直方图将水平放置，以产生重叠的直方图图案，这样可以很容易地对两组数据进行比较。两组直方图使用不同的颜色或不同的填充图案。

4.27 设计一个实现颜色查找表的算法。

4.28 假如读者拥有每英寸能显示 100 个像素的 8 × 10 英寸的显示屏幕系统，这个系统的一个颜色查找表有 64 个位置，那么对于帧缓存，其最小的可能尺寸（以字节计）是多少？

4.29 考虑有 512 × 512 每像素 20 位的帧缓存和每像素 24 位的颜色查找表的 RGB 光栅系统，(a)使用这样的系统可以显示多少种不同的灰度等级？(b)能显示多少种不同的颜色（包括灰度等级）？(c)在任何一个时刻能显示多少种颜色？(d)总的存储容量是多少？(e)说明两种减少容量的同时保持相同的颜色能力的方法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

4.30 修改扫描线算法，从而将任何指定的矩形填充图案应用到多边形内部，从所指定的图案位置开始填充。

4.31 编写一个程序，使用扫描转换法对给定椭圆的内部进行单色填充。

4.32 编写一个程序，使用指定的图案对给定椭圆的内部进行单色填充。

4.33 编写一个程序，对任意指定的包括有自相交边的一组填充区顶点的内部进行填充，使用非零环绕数规则识别内部区域。

4.34 修改 4- 连通区域的边界填充算法，使之通过结合扫描线方法避免过多使用堆栈。

4.35 编写一个 8- 连通区域的边界填充程序。

4.36 解释使用中点法显示的椭圆怎样能用边界填充算法进行填充。

4.37 开发并实现一个对任意指定区域的内部填充的泛滥算法。

4.38 定义和实现一个修改已有矩形填充图案尺寸的程序。

4.39 编写一个实现软填充算法的程序。仔细定义所要完成的软填充算法以及如何组合颜色。

4.40 设计一个算法，调整定义为矩形网格图案的字符高度和宽度。

4.41 实现设置字符向上向量和文本路径以控制字符串显示的程序。

4.42 编写一个程序，按照对齐参数的输入值指定的位置将文本对齐。

4.43 开发一个实现标记属性（尺寸和颜色）的程序。

4.44 扩展 Bresenham 画线算法，实现一个反走样程序，从而调整线路径附近的像素亮度。

4.45 实现一个对于中点线算法的反走样程序。

4.46 开发一个对于椭圆边界的反走样算法。

4.47 修改区域填充的扫描线算法，在其中加入反走样。使用连贯性来减少连续扫描线上的计算量。

4.48 编写一个实现用扫描线过程、OpenGL 画点函数填充多边形内部时的 Pitteway-Watkinson 反走样算法的程序。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第5章 几何变换

到目前为止，我们已经看到如何使用线段和填充区等图元来描述场景，并利用属性来辅助这些图元。我们给出了扫描线算法，可以将图元显示在光栅设备上。现在，再看看可用于对象重定位或改变大小的变换操作。这些操作也用于将世界坐标系中的场景描述转换为输出设备上显示的观察子程序中。另外，它们用于各种其他的应用中，如计算机辅助设计和计算机动画。例如，一个建筑设计师通过安排组成部分的方向和大小来创建一个设计布局图，而计算机动画师通过沿指定路径移动“照相机”位置或场景中的对象来开发一个视频序列。应用于对象几何描述并改变它的位置、方向或大小的操作称为**几何变换**（geometric transformation）。

几何变换有时也称为建模变换（modeling transformation），但有些图形系统将两者区分开来。建模变换一般用于构造场景或给出由多个部分组合而成的复杂对象的层次式描述等。例如，一架飞机由机翼、机尾、机身、发动机和其他部分组成，每一部分又可描述成第二级组合体，如此等等，在层次结构中逐层往下描述。因此，该飞机可用那些部件及附属于每个部件的“建模”变换来描述，这些变换指出那些部件如何适用于整个飞机的设计。另一方面，几何变换能用来描述动画序列中对象在场景中可以怎样移动或简单地从另一角度来观察它们。因此，有些图形系统提供两套变换子程序，而其他一些软件包则提供一套能同时用于几何变换和建模变换的函数。

5.1 基本的二维几何变换

平移、旋转和缩放是所有图形软件包中都含有的几何变换函数。其他可能包括在图形软件包中的变换函数有反射和错切操作。为了介绍几何变换的一般概念，我们首先考虑二维操作，然后讨论这些基本的思想怎样扩充到三维场景中。在理解基本的概念后，我们可以很容易地编写执行二维场景对象几何变换的程序。

5.1.1 二维变换

通过将位移量加到一个点的坐标上来生成一个新的坐标位置，可以实现一次**平移**（translation）。实际上，我们将该点从原始位置沿一直线路径移动到新位置。类似地，对于一个用多个坐标位置定义的对象（如四边形），可以通过对所有坐标位置使用相同的位移量沿平行路径重定位来实现平移。然后在新位置显示完整的对象。

将**平移距离**（translation distance）t_x和t_y加到原始坐标(x, y)上获得一个新的坐标位置(x', y')，可以实现一个二维位置的平移，如图5.1所示。

$$x' = x + t_x, \qquad y' = y + t_y \tag{5.1}$$

一对平移距离(t_x, t_y)称为**平移向量**（translation vector）或**位移向量**（shift vector）。

我们可以使用下面的列向量来表示坐标位置和平移向量，然后将方程（5.1）表示成单个矩阵等式。

$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.2}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

这样就可以使用矩阵形式来表示二维平移方程：

$$\mathbf{P}' = \mathbf{P} + \mathbf{T} \tag{5.3}$$

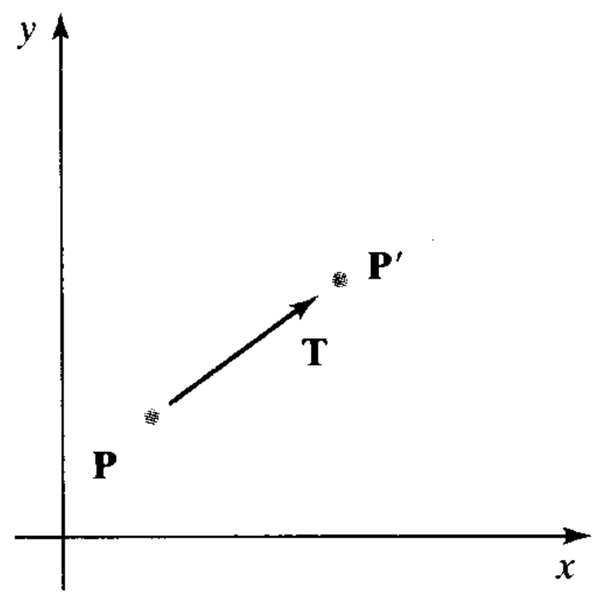


图 5.1　使用平移向量 **T**，从位置 **P** 平移一个点到位置**P′**

平移是一种移动对象而不改变其形状的刚体变换。即对象上的每一点移动了同样的距离。一条直线段的平移通过使用方程（5.3）对其两端点进行移动并重画两个新端点位置间的线段来实现。多边形也以类似的方法平移。我们将平移向量加到多边形的每一顶点坐标位置，然后使用这组新端点来重新生成多边形。图 5.2 给出了使用指定的平移向量将一个对象从一个位置移动到另一位置的例子。

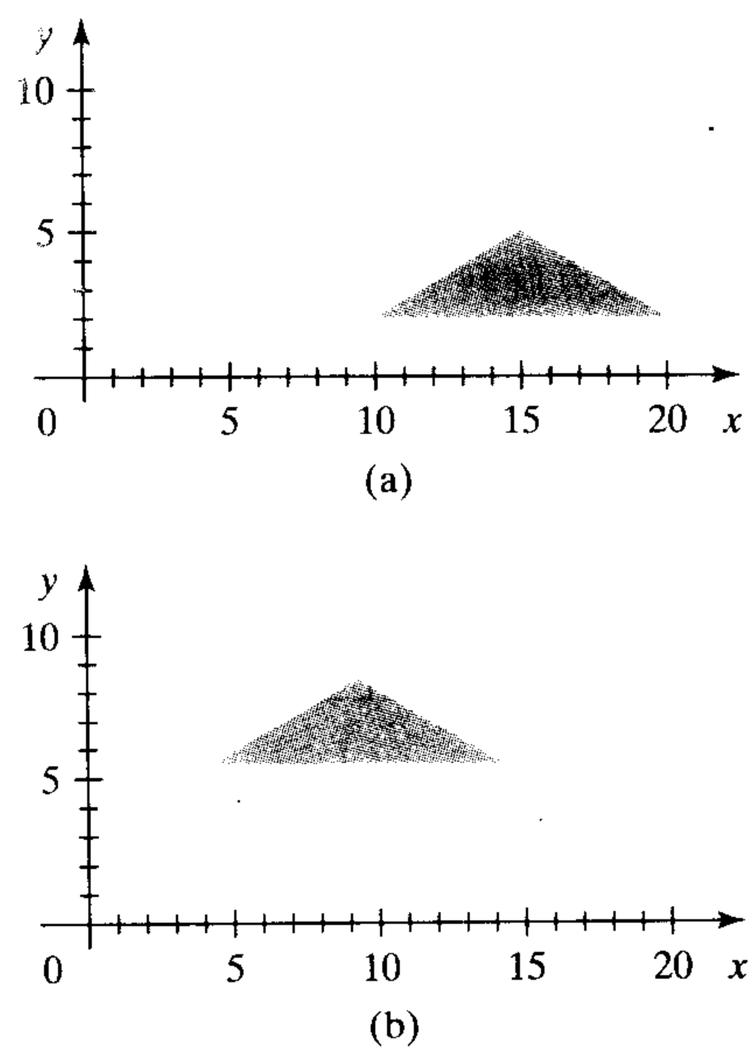


图 5.2　使用平移向量(−5.50, 3.75)将多边形从位置(a)移到位置(b)

下面的程序演示了平移操作。输入的平移向量用来将一个多边形的*n*个顶点从一个世界坐标系移动到另一个，而 OpenGL 子程序用来重新生成平移后的多边形。

```
class wcPt2D {
   public:
      GLfloat x, y;
};

void translatePolygon (wcPt2D * verts, GLint nVerts, GLfloat tx, GLfloat ty)
{
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    GLint k;

    for (k = 0; k < nVerts; k++) {
       verts [k].x = verts [k].x + tx;
       verts [k].y = verts [k].y + ty;
    }
    glBegin (GL_POLYGON);
       for (k = 0; k < nVerts; k++)
          glVertex2f (verts [k].x, verts [k].y);
    glEnd ( );
}
```

如果我们要删除原来的多边形，则可以在平移前用背景色显示它。有些图形软件包中有另外的删除图形部分的方法。同样，如果我们要保存原来的多边形位置，可以将平移后的多边形存入不同的数组。

可以使用同样的方法来平移其他对象。为了改变圆或椭圆的位置，我们可以平移中心坐标并在新的中心位置上重画图形。对于一个样条曲线，通过平移定义该曲线路径的点，然后使用平移过的坐标位置来重构曲线。

5.1.2 二维旋转

通过指定一个**旋转轴**（rotation axis）和一个**旋转角度**（rotation angle），可以进行一次**旋转**（rotation）变换。在将对象的所有顶点按指定角度绕指定旋转轴旋转后，该对象的所有点都旋转到新位置。

一个对象的二维旋转通过在 xy 平面上沿圆路径将对象重定位来实现。此时，我们将对象绕与 xy 平面垂直的旋转轴（与 z 轴平行）旋转。三维旋转的参数有旋转角 θ 和称为**旋转点**（rotation point 或 pivot point）的位置 (x_r, y_r)，对象绕该点旋转（参见图 5.3）。基准点是旋转轴与 xy 平面的交点。正角度 θ 定义绕基准点的逆时针旋转（参见图 5.3），而负角度将对象沿顺时针方向旋转。

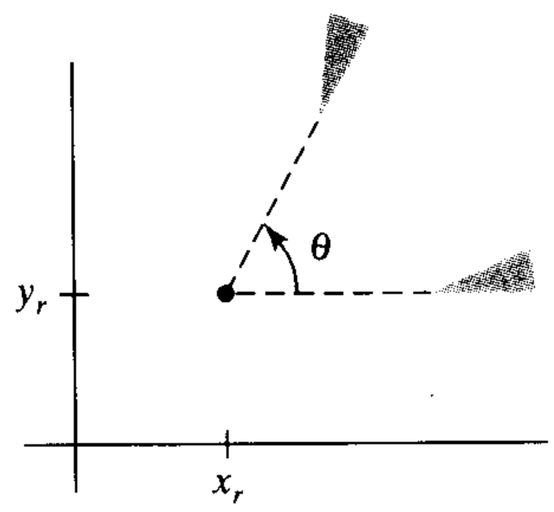


图 5.3 绕基准点 (x_r, y_r) 将对象旋转 θ 角

为了简化该基本方法的叙述，我们首先确定当基准点为坐标原点时点位置**P**进行旋转的变换方程。原始点和变换后点位置的角度和坐标关系如图 5.4 所示。其中，r 是点到原点的固定距离，角 ϕ 是点的原始角度位置与水平线的夹角，θ 是旋转角。应用标准的三角等式，我们可以利用角度 θ 和 ϕ 将转换后的坐标表示为

$$\begin{aligned} x' &= r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y' &= r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \end{aligned} \tag{5.4}$$

在极坐标系中，点的原始坐标为

$$x = r\cos\phi, \qquad y = r\sin\phi \tag{5.5}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

将表达式（5.5）代入方程（5.4）中，我们就得到相对于原点，将位置(x, y)的点旋转 θ 角的变换方程：

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned} \tag{5.6}$$

使用列向量表达式（5.2）表示坐标位置，那么旋转方程的矩阵形式为

$$\mathbf{P}' = \mathbf{R}\cdot\mathbf{P} \tag{5.7}$$

其中，旋转矩阵为

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.8}$$

表达式（5.2）中坐标位置 **P** 的列向量是标准的数学表示。然而，早期的图形系统有时用行向量表示坐标位置，这会改变执行旋转时矩阵相乘的次序。但现在，OpenGL、Java、PHIGS 和 GKS 都按标准列向量方式表示。

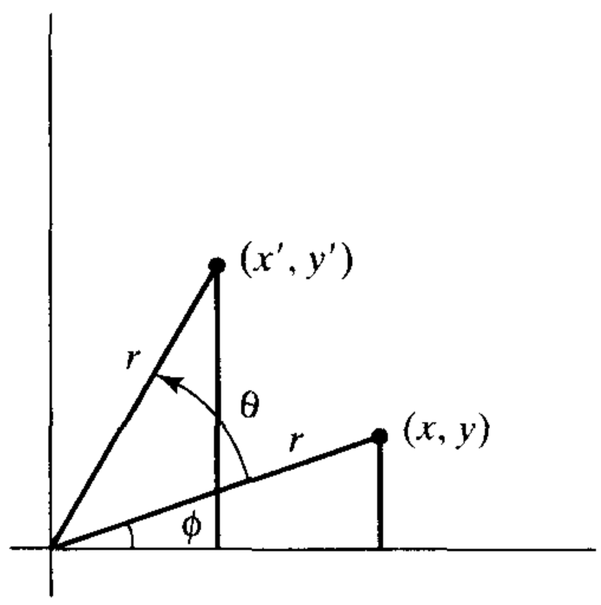


图 5.4　相对于原点将点从位置(x, y)旋转 θ 角到(x', y')点。原点对 x 轴的角位移为 ϕ

图 5.5 给出了绕任意基准点旋转一个点的例子。利用图中的三角关系，可以将方程（5.6）规范化为绕任意指定的旋转位置(x_r, y_r)旋转的点的变换方程：

$$\begin{aligned} x' &= x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta \\ y' &= y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \end{aligned} \tag{5.9}$$

这个通用旋转方程不同于方程（5.6），该方程包含了一个加项以及在坐标值上的多重系数。因此，通过其中的元素包含方程（5.9）中的加项（平移项）的列向量矩阵加法，就可以修改矩阵表达式（5.7），使其包括基准点坐标。然而，还有更好的方法可以形成这样的矩阵公式，在 5.2 节我们将讨论表达变换公式的更一致的方法。

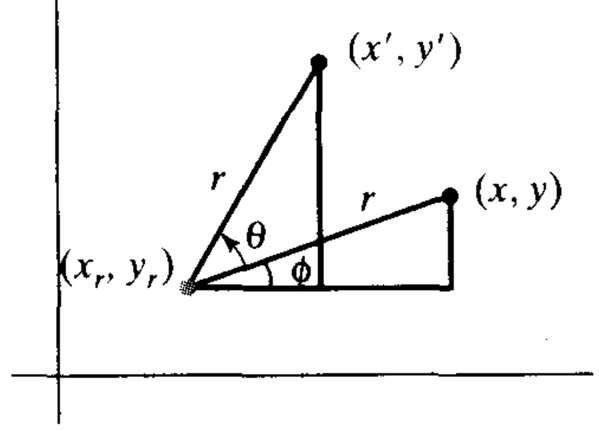


图 5.5　相对旋转点(x_r, y_r)将点从位置(x, y)旋转 θ 角到位置(x', y')

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

类似于平移，旋转是一种不变形地移动对象的刚体变换，对象上的所有点旋转相同的角度。线段的旋转可以通过将旋转方程（5.9）用于每个线段端点，并重新绘制新端点间的线段而得到。多边形的旋转则是将每个顶点旋转指定的旋转角，并使用新的顶点来生成多边形而实现旋转。曲线的旋转通过定位定义的点并重新绘制曲线而完成。例如圆或椭圆，可以通过将中心位置沿指定旋转角对着的弧移动而绕非中心轴旋转。椭圆可通过旋转其长轴和短轴来实现绕其中心位置的旋转。

在下列程序示例中，一个多边形绕指定的世界坐标系中的基准点旋转。旋转过程的输入参数是原始的多边形顶点、基准点坐标和用弧度表示的旋转角 θ，多边形用 OpenGL 子程序重新生成。

```
class wcPt2D {
   public:
      GLfloat x, y;
};

void rotatePolygon (wcPt2D * verts, GLint nVerts, wcPt2D pivPt,
                    GLdouble theta)
{
   wcPt2D * vertsRot;
   GLint k;

   for (k = 0; k < nVerts; k++) {
      vertsRot [k].x = pivPt.x + (verts [k].x - pivPt.x) * cos (theta)
                          - (verts [k].y - pivPt.y) * sin (theta);
      vertsRot [k].y = pivPt.y + (verts [k].x - pivPt.x) * sin (theta)
                          + (verts [k].y - pivPt.y) * cos (theta);
   }
   glBegin {GL_POLYGON};
      for (k = 0; k < nVerts; k++)
         glVertex2f (vertsRot [k].x, vertsRot [k].y);
   glEnd ( );
}
```

5.1.3 二维缩放

改变一个对象的大小，可使用**缩放**（scaling）变换。一个简单的二维缩放操作可通过将**缩放系数**（scaling factor）s_x 和 s_y 与对象坐标位置(x, y)相乘而得：

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y \tag{5.10}$$

缩放系数 s_x 在 x 方向缩放对象，而 s_y 在 y 方向进行缩放。基本的二维缩放方程（5.10）也可以写成矩阵形式：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.11}$$

或

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P} \tag{5.12}$$

其中，**S** 是等式（5.11）中的 2 × 2 缩放矩阵。

可以赋给缩放系数 s_x 和 s_y 任何正数值。值小于 1 将缩小对象的尺寸，值大于 1 则放大对象。如果将 s_x 和 s_y 都指定为 1，那么对象尺寸就不会改变。当赋给 s_x 和 s_y 相同的值时，就会产生保持对象相对比例的**一致缩放**（uniform scaling）。s_x 和 s_y 值不等时将产生设计应用中常见的

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

差值缩放（differential scaling），其中的图形由少数形状经缩放和定位变换来构造（参见图 5.6）。在有些系统中，也可为缩放参数指定负值。这不仅改变对象的尺寸，还相对于一个或多个坐标轴反射。

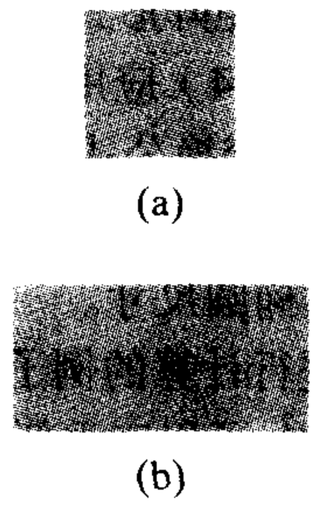


图 5.6　使用缩放系数 $s_x = 2$ 和 $s_y = 1$ 将正方形(a)变换成矩形(b)

利用方程（5.11）变换的对象既被缩放，又被重定位。当缩放系数的绝对值小于 1 时，缩放后的对象向原点靠近；而缩放系数绝对值大于 1 时，缩放后的坐标位置远离原点。图 5.7 给出了将值 0.5 赋给方程（5.11）中的 s_x 和 s_y 时对线段的缩放。线段的长度和到原点的距离都减少到 1/2。

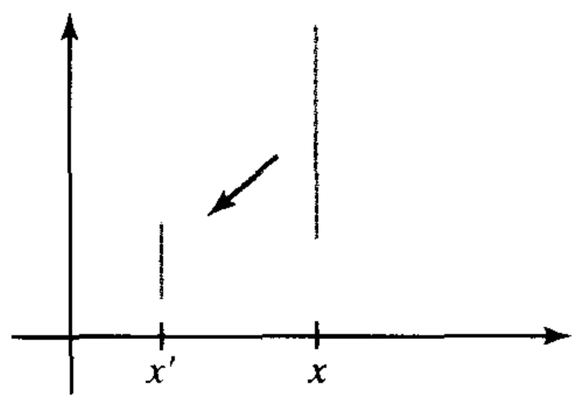


图 5.7　$s_x = s_y = 0.5$ 时使用等式（5.12）缩放的线段，尺寸减少并向坐标原点移动

我们可以选择一个在缩放变换后不改变位置的点，称为**固定点**（fixed point），以控制缩放后对象的位置。固定点的坐标(x_f, y_f)可以选择对象的中点（参见附录 A）等位置或任何其他空间位置。这样，多边形通过缩放每个顶点到固定点的距离而相对于固定点进行缩放（参见图 5.8）。对于坐标为(x, y)的顶点，缩放后的坐标(x', y')可计算为

$$x' - x_f = (x - x_f)s_x, \qquad y' - y_f = (y - y_f)s_y \tag{5.13}$$

我们可以将乘积项和加法项分开而重写上述缩放变换公式：

$$\begin{aligned} x' &= x \cdot s_x + x_f(1 - s_x) \\ y' &= y \cdot s_y + y_f(1 - s_y) \end{aligned} \tag{5.14}$$

其中，加法项 $x_f(1-s_x)$和 $y_f(1-s_y)$ 对于对象中的任何点都是常数。

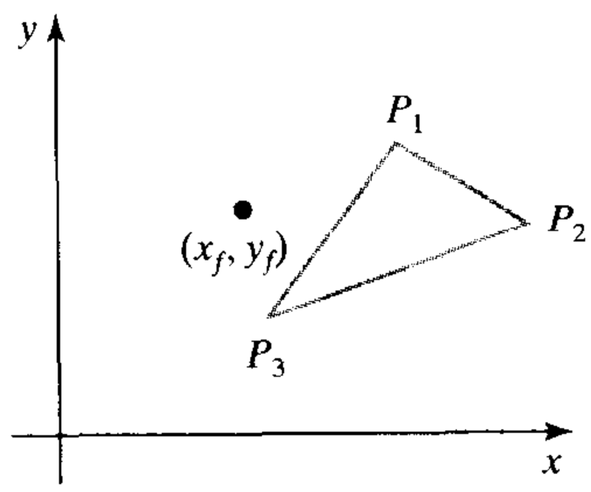


图 5.8　相对于所选择的固定点(x_f, y_f)的缩放，从每个多边形顶点到固定点的距离由变换方程（5.13）进行缩放

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在缩放公式中包含固定点的坐标，类似于在旋转公式中包含基准点的坐标。我们可以建立一个其元素为方程（5.14）中常数项的列向量，然后将这个列向量加到等式（5.12）中的乘积 **S** · **P** 上。下一节将讨论仅包含矩阵乘法的变换方程的矩阵形式。

多边形的缩放可以通过将变换方程（5.14）应用于每个顶点，然后利用变换后的顶点重新生成多边形而实现。其他对象的变换则将缩放变换公式应用到定义对象的参数上。要改变圆的大小，可通过缩放其半径并计算圆上坐标点的新坐标位置来实现。标准位置中的椭圆通过缩放两个轴并且按其中心坐标重新绘制椭圆而实现缩放椭圆尺寸。

下列程序给出了对一个多边形缩放进行计算的例子。多边形顶点和固定点的坐标以及缩放系数是输入参数。坐标变换后，使用 OpenGL 子程序重新生成缩放后的多边形。

```
class wcPt2D {
   public:
      GLfloat x, y;
};

void scalePolygon (wcPt2D * verts, GLint nVerts, wcPt2D fixedPt,
                                GLfloat sx, GLfloat sy)
{
   wcPt2D vertsNew;
   GLint k;

   for (k = 0; k < nVerts; k++) {
      vertsNew [k].x = verts [k].x * sx + fixedPt.x * (1 - sx);
      vertsNew [k].y = verts [k].y * sy + fixedPt.y * (1 - sy);
   }
   glBegin {GL_POLYGON};
      for (k = 0; k < nVerts; k++)
         glVertex2f (vertsNew [k].x, vertsNew [k].y);
   glEnd ( );
}
```

5.2 矩阵表示和齐次坐标

许多图形应用涉及到几何变换的顺序。例如，动画需要将对象在运动的每个增量处进行平移和旋转。在设计和图形构造的应用中，我们要完成平移、旋转和缩放，从而将图形组成部分安排到合适的位置。观察变换涉及一系列平移和旋转，从而将原始的场景描述变成输出设备上的显示。这里我们考虑怎样重组上一节所讨论的矩阵表达式，从而可以有效地处理这种变换顺序。

在 5.1 节中我们已经看到，每个基本变换（平移、旋转和缩放）都可以表示为普通矩阵形式：

$$\mathbf{P}' = \mathbf{M}_1 \cdot \mathbf{P} + \mathbf{M}_2 \tag{5.15}$$

坐标位置 **P′** 和 **P** 表示为列向量，矩阵 $\mathbf{M}_1$ 是一个包含乘法系数的 2 × 2 矩阵，$\mathbf{M}_2$ 是包含平移项的两元素列矩阵。对于平移，$\mathbf{M}_1$ 是单位矩阵。对于旋转或缩放，$\mathbf{M}_2$ 包含与基准点或缩放固定点相关的平移项。为了利用这个公式产生先缩放、再旋转、后平移这样的变换顺序，必须一步一步地计算变换的坐标。首先将坐标位置缩放，然后将缩放后的坐标旋转，最后将旋转后的坐标平移。更有效的方法是将变换组合，从而直接从初始坐标得到最后的坐标位置，这样就消除了中间坐标值的计算。因此，需要重组等式（5.15）以消除 $\mathbf{M}_2$ 中与平移项相关的矩阵加法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

5.2.1　齐次坐标

如果将 2 × 2 矩阵表达式扩充为 3 × 3 矩阵，就可以把二维几何变换的乘法和平移项组合成单一矩阵表示。这时将变换矩阵的第三列用于平移项，而所有的变换公式可表达为矩阵乘法。但为了这样操作，必须解释二维坐标位置到三元列向量的矩阵表示。标准的实现技术是将二维坐标位置表示(x, y)扩充到三维表示(x_h, y_h, h)，称为**齐次坐标**（homogeneous coordinate），这里的**齐次参数**（homogeneous parameter）h 是一个非零值，因此

$$x = \frac{x_h}{h}, \qquad y = \frac{y_h}{h} \tag{5.16}$$

这样，普通的二维齐次坐标表示可写为$(h \cdot x, h \cdot y, h)$。对于二维几何变换，可以把齐次参数 h 取为任何非零值。因而，对于每个坐标点(x, y)，可以有无数个等价的齐次表达式。最方便的选择是简单地设置 $h = 1$。因此每个二维位置都可用齐次坐标$(x, y, 1)$来表示。h 的其他值也是需要的，例如在三维观察变换的矩阵公式中。

齐次坐标这一术语在数学中用来指出笛卡儿方程的表达效果。当笛卡儿点(x, y)转换成齐次坐标（x_h, y_h, h）时，包含 x 和 y 的方程 $f(x, y) = 0$ 变成了三个参数 x_h、y_h 和 h 的齐次方程。这恰好说明，假如三个参数均被各自乘上值 v 后的值替换，那么 v 可以从方程中作为因子提取出来。

利用齐次坐标表示位置，使我们可以用矩阵相乘的形式来表示所有的几何变换公式，而这是图形系统中使用的标准方法。二维坐标位置用三元素列向量表示，而二维变换操作用一个 3 × 3 矩阵表示。

5.2.2　二维平移矩阵

使用齐次坐标方法，坐标位置的二维平移可表示为下面的矩阵乘法。

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.17}$$

该平移操作可简写为

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P} \tag{5.18}$$

其中 $\mathbf{T}(t_x, t_y)$是等式（5.17）中的 3 × 3 矩阵。在平移参数没有混淆的情况下，我们可以简单地使用 **T** 来表示平移矩阵。

5.2.3　二维旋转矩阵

类似地，绕坐标系原点的二维旋转变换方程可以表示成矩阵形式：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.19}$$

或

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P} \tag{5.20}$$

旋转变换操作 $\mathbf{R}(\theta)$是方程（5.19）中旋转参数为 θ 的 3 × 3 矩阵。我们可以简单地把旋转矩阵写成 **R**。

有些图形软件包中只支持如方程（5.19）所示的绕坐标系原点的二维旋转函数。绕任意基准点的旋转要经过一系列的变换操作来完成。图形软件中的一种替代方法是在旋转子程序中为基准点坐标提供另外的参数。然后，包含基准点参数的旋转子程序建立一个不需要引入一系列变换函数的通用旋转矩阵。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

5.2.4 二维缩放矩阵

最后，相对于坐标原点的缩放变换现在可以表示为矩阵乘法：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.21}$$

或

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P} \tag{5.22}$$

缩放操作 $\mathbf{S}(s_x, s_y)$是方程（5.21）中以 s_x 和 s_y 为参数的 3×3 矩阵。多数情况下，我们可以将缩放矩阵表示成 $\mathbf{S}$。

有些软件包仅提供如方程（5.21）所示的以坐标系原点为中心的缩放函数。在这种情况下，以另一参考点为中心的缩放变换通过一系列变换操作来处理。然而，有些系统也包括通用的缩放子程序，可为以指定点为中心的缩放构造齐次坐标。

5.3 逆变换

对于平移变换，我们通过对平移距离取负值而得到逆矩阵。因此，如果二维平移距离是 t_x 和 t_y，则其逆平移矩阵是

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.23}$$

这产生相反方向的平移，而平移矩阵和其逆矩阵的乘积是一个单位矩阵。

逆旋转通过用旋转角度的负角取代该旋转角来实现。例如，绕坐标系原点的角度为 θ 的二维旋转有如下的逆变换矩阵：

$$\mathbf{R}^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.24}$$

旋转角的负值生成顺时针方向的旋转，因而当任何旋转矩阵和其逆旋转矩阵相乘时生成单位矩阵。由于旋转角符号的变化仅影响 sin 函数，因此该逆矩阵可以通过交换行与列来获得。即我们可以做任何旋转矩阵 $\mathbf{R}$ 的转置矩阵来得到它的逆矩阵（$\mathbf{R}^{-1} = \mathbf{R}^T$）。

将缩放系数用其倒数取代就得到了缩放变换的逆矩阵。对以坐标系原点为中心、缩放参数为 s_x 和 s_y 的二维缩放，其逆变换矩阵为

$$\mathbf{S}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.25}$$

该逆矩阵生成相反的缩放变换，因此任何缩放矩阵与其逆矩阵的乘积生成了单位矩阵。

5.4 二维复合变换

利用矩阵表达式，可以通过计算单个变换的矩阵乘积，将任意的变换序列组成**复合变换矩阵**（composite transformation matrix）。形成变换矩阵的乘积经常称为矩阵的**合并**（concatenation）或

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

复合（composition）。由于一个坐标位置用齐次列矩阵表示，我们必须用表达任一变换次序的矩阵前乘该列矩阵。由于场景中许多位置用相同的顺序变换，先将所有变换矩阵相乘形成一个复合矩阵将是高效率的方法。因此，如果我们要对点位置 **P** 进行两次变换，变换后的位置将用下式计算：

$$\begin{aligned}\mathbf{P}' &= \mathbf{M}_2 \cdot \mathbf{M}_1 \cdot \mathbf{P} \\ &= \mathbf{M} \cdot \mathbf{P}\end{aligned} \tag{5.26}$$

该坐标位置使用矩阵 **M** 来变换，而不是单独地先用 $\mathbf{M}_1$ 然后 $\mathbf{M}_2$ 来变换。

5.4.1　复合二维平移

假如将两个连续的平移向量(t_{1x}, t_{1y})和(t_{2x}, t_{2y})用于坐标位置 **P**，那么最后的变换位置 **P′** 可以计算为

$$\begin{aligned}\mathbf{P}' &= \mathbf{T}(t_{2x}, t_{2y}) \cdot \{\mathbf{T}(t_{1x}, t_{1y}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y})\} \cdot \mathbf{P}\end{aligned} \tag{5.27}$$

其中，**P** 和 **P′** 表示为三元素、齐次坐标的列向量。我们可以计算两个相关的矩阵乘积来检验这个结果。同样，这个平移序列的复合变换矩阵为

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \tag{5.28}$$

或

$$\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1y} + t_{2y}) \tag{5.29}$$

这表示两个连续平移是相加的。

5.4.2　复合二维旋转

应用于 **P** 的两个连续旋转产生的变换为

$$\begin{aligned}\mathbf{P}' &= \mathbf{R}(\theta_2) \cdot \{\mathbf{R}(\theta_1) \cdot \mathbf{P}\} \\ &= \{\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1)\} \cdot \mathbf{P}\end{aligned} \tag{5.30}$$

通过两个旋转矩阵相乘，我们可以证明两个连续旋转是相加的

$$\mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1) = \mathbf{R}(\theta_1 + \theta_2) \tag{5.31}$$

因此，点旋转的最后坐标可以使用复合变换矩阵计算为

$$\mathbf{P}' = \mathbf{R}(\theta_1 + \theta_2) \cdot \mathbf{P} \tag{5.32}$$

5.4.3　复合二维缩放

合并两个连续的二维缩放操作的变换矩阵，生成如下的复合缩放矩阵：

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.33}$$

或

$$\mathbf{S}(s_{2x}, s_{2y}) \cdot \mathbf{S}(s_{1x}, s_{1y}) = \mathbf{S}(s_{1x} \cdot s_{2x},\ \ s_{1y} \cdot s_{2y}) \tag{5.34}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

这种情况下的结果矩阵表明，连续缩放操作是相乘的，假如我们要连续两次将对象尺寸放大3倍，那么其最后的尺寸将是原始尺寸的9倍。

5.4.4 通用二维基准点旋转

当图形软件包仅提供绕坐标系原点旋转函数时，我们可通过完成下列平移-旋转-平移操作序列来实现绕任意选定的基准点(x_r, y_r)的旋转。

1. 平移对象使基准点位置移动到坐标原点；
2. 绕坐标原点旋转；
3. 平移对象使基准点回到其原始位置。

这个变换序列如图5.9所示。利用矩阵合并可以得到该序列的复合变换矩阵：

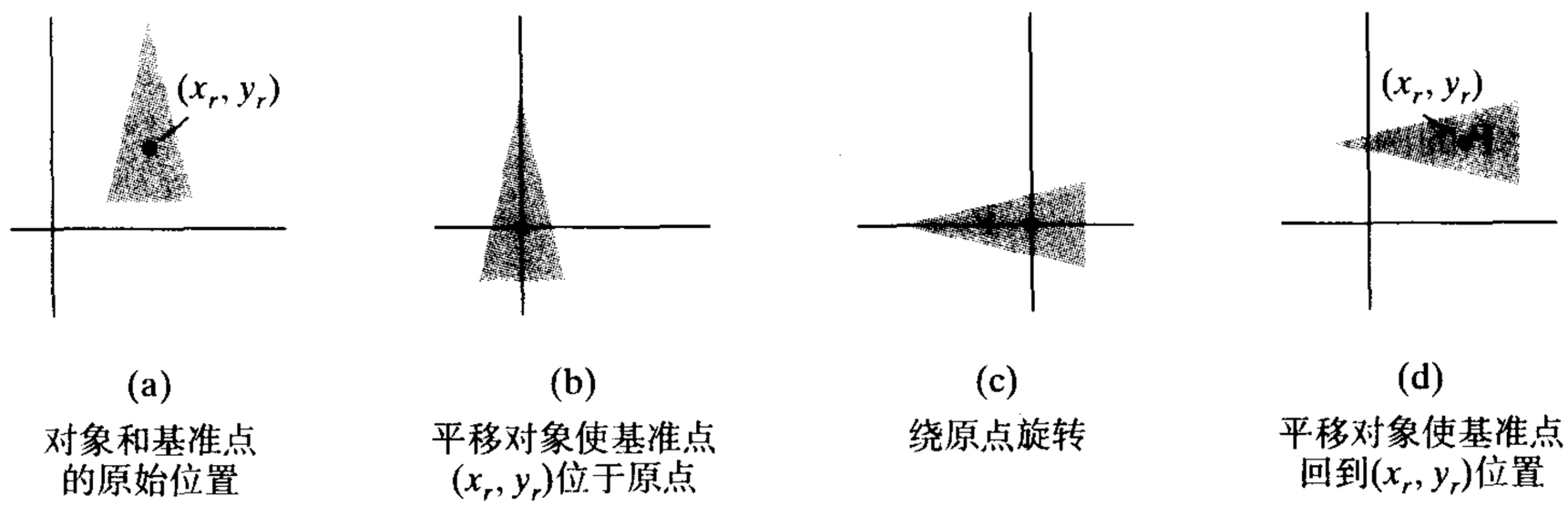
$$\begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{5.35}$$

可以使用下列形式表示：

$$\mathbf{T}(x_r, y_r) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-x_r, -y_r) = \mathbf{R}(x_r, y_r, \theta) \tag{5.36}$$

其中，$\mathbf{T}(-x_r, -y_r) = \mathbf{T}^{-1}(x_r, y_r)$。通常，可以将图形库中的旋转函数设计成先接收基准点坐标参数及旋转角，然后自动生成方程（5.35）的旋转矩阵。



(a) 对象和基准点的原始位置

(b) 平移对象使基准点(x_r, y_r)位于原点

(c) 绕原点旋转

(d) 平移对象使基准点回到(x_r, y_r)位置

图5.9 使用方程（5.19）的旋转矩阵$\mathbf{R}(\theta)$绕指定基准点旋转一对象的变换顺序

5.4.5 通用二维基准点缩放

只有相对于坐标原点缩放的缩放函数时，图5.10给出了关于任意选择的基准位置(x_f, y_f)缩放的变换序列：

1. 平移对象使固定点与坐标原点重合；
2. 对于坐标原点进行缩放；
3. 使用步骤1的反向平移将对象返回到原始位置。

将这三个操作的矩阵合并，就可以产生所需的缩放矩阵：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\begin{bmatrix}1 & 0 & x_f\\0 & 1 & y_f\\0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}s_x & 0 & 0\\0 & s_y & 0\\0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}1 & 0 & -x_f\\0 & 1 & -y_f\\0 & 0 & 1\end{bmatrix}=\begin{bmatrix}s_x & 0 & x_f(1-s_x)\\0 & s_y & y_f(1-s_y)\\0 & 0 & 1\end{bmatrix} \tag{5.37}$$

或

$$\mathbf{T}(x_f, y_f)\cdot\mathbf{S}(s_x, s_y)\cdot\mathbf{T}(-x_f, -y_f)=\mathbf{S}(x_f, y_f, s_x, s_y) \tag{5.38}$$

该变换在提供接收基准点坐标的缩放函数的系统中自动生成。

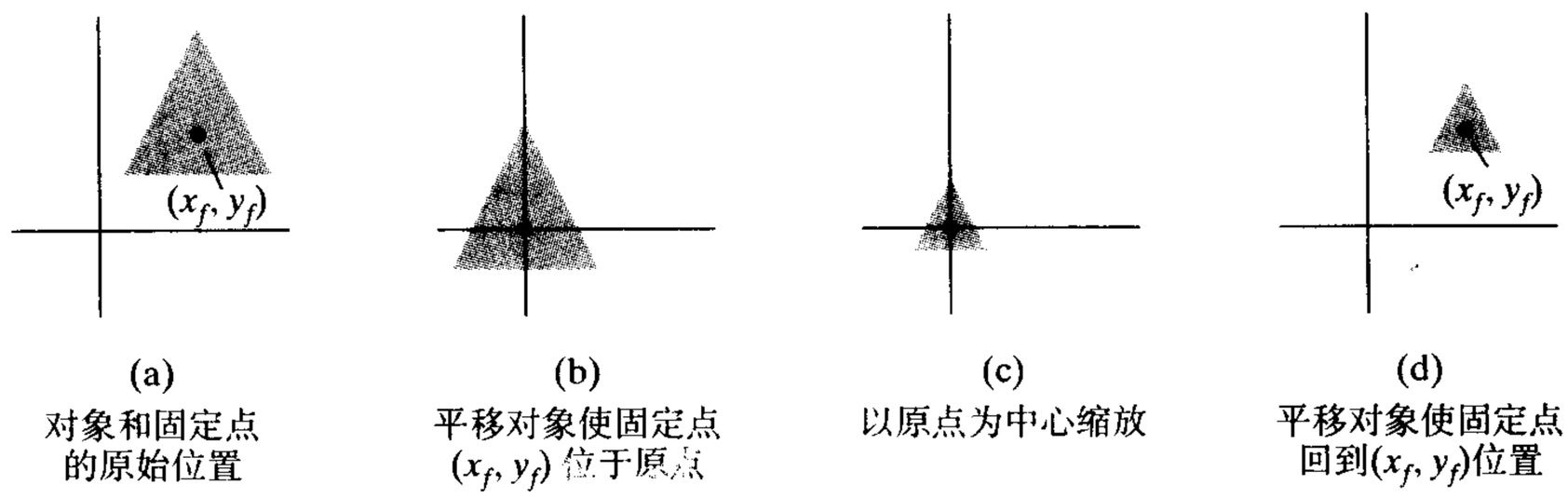


图 5.10 使用变换方程（5.21）的缩放矩阵 $\mathbf{S}(s_x, s_y)$对以指定的固定位置为中心进行对象缩放的变换顺序

5.4.6 通用二维定向缩放

参数 s_x 和 s_y 沿 x 和 y 方向缩放对象，可以通过在应用缩放变换之前，将对象所希望的缩放方向旋转到与坐标轴一致而在其他方向上缩放对象。

假如我们要在图 5.11 所示的方向上，使用参数 s_1 和 s_2 指定的值作为缩放系数。为了完成这种缩放而不改变对象方向，我们首先完成旋转操作，使 s_1 和 s_2 的方向分别与 x 和 y 轴重合。然后应用缩放变换 $\mathbf{S}(s_1, s_2)$，再进行反向旋转以回到其原始位置。从这三个变换的乘积得到的复合矩阵为

$$\mathbf{R}^{-1}(\theta)\cdot\mathbf{S}(s_1, s_2)\cdot\mathbf{R}(\theta)=\begin{bmatrix}s_1\cos^2\theta+s_2\sin^2\theta & (s_2-s_1)\cos\theta\sin\theta & 0\\(s_2-s_1)\cos\theta\sin\theta & s_1\sin^2\theta+s_2\cos^2\theta & 0\\0 & 0 & 1\end{bmatrix} \tag{5.39}$$

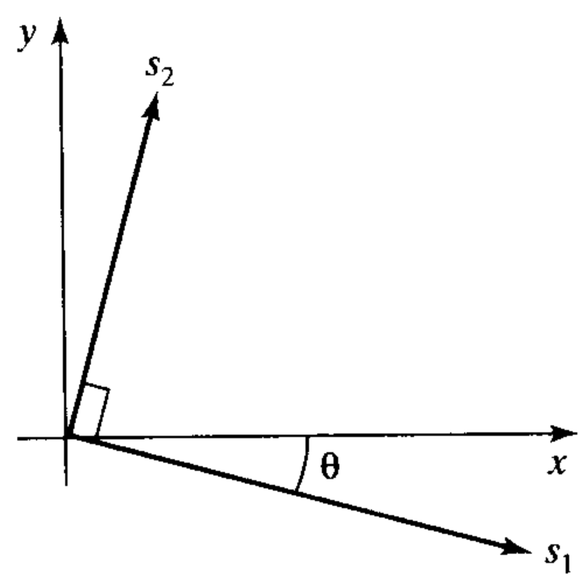


图 5.11 以正交方向由角位移 θ 定义的缩放参数 s_1 和 s_2

作为缩放变换的一个例子，通过沿(0, 0)到(1, 1)的对角线将单位正方形拉长，使其转换成平行四边形（参见图 5.12）。我们使用参数 $\theta = 45°$ 将对角线旋转到 y 轴，并按 $s_1 = 1$ 和 $s_2 = 2$ 将其长度加倍，然后再旋转使对角线回到原来的位置。

在矩阵（5.39）中，假设缩放是相对原点完成的，可以将这个缩放操作推进一步并与平移操作合并，从而使复合矩阵包含为指定的固定位置进行缩放的参数。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

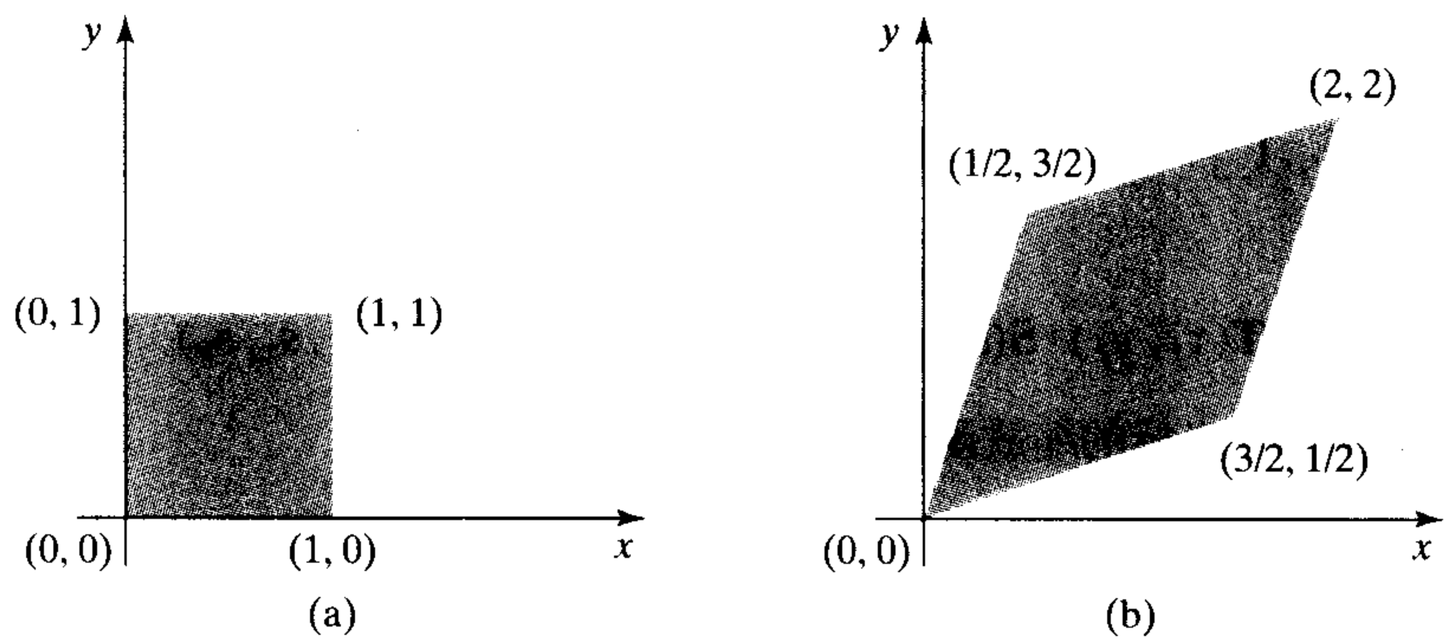


图 5.12 使用复合变换矩阵（5.39），并根据 $s_1 = 1$、$s_2 = 2$ 和 $\theta = 45°$，将单位正方形(a)变换成平行四边形(b)

5.4.7 矩阵合并特性

矩阵相乘符合结合律。对于任何三个矩阵 $\mathbf{M}_1$、$\mathbf{M}_2$ 和 $\mathbf{M}_3$，矩阵积 $\mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1$ 可先将 $\mathbf{M}_3$ 和 $\mathbf{M}_2$ 相乘或先将 $\mathbf{M}_2$ 和 $\mathbf{M}_1$ 相乘：

$$\mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1 = (\mathbf{M}_3 \cdot \mathbf{M}_2) \cdot \mathbf{M}_1 = \mathbf{M}_3 \cdot (\mathbf{M}_2 \cdot \mathbf{M}_1) \tag{5.40}$$

因此，依靠变换的描述次序，我们既可以使用从左到右（前乘）、也可以使用从右到左（后乘）的结合分组来求矩阵乘积。有些图形软件包要求变换按应用的次序描述。在这种情况下，我们先引入变换 $\mathbf{M}_1$，然后 $\mathbf{M}_2$，最后 $\mathbf{M}_3$。在每一个连续的变换子程序被调用时，其矩阵从左边与前面的矩阵乘积合并。而另一些图形系统是后乘矩阵，因此该变换序列按相反次序引入：最后引入的变换（本例中是 $\mathbf{M}_1$）是最先应用的，而第一个被调用的变换（此时是 $\mathbf{M}_3$）是最后应用的。

另一方面，变换积一般不可交换，矩阵积 $\mathbf{M}_2 \cdot \mathbf{M}_1$ 不等于 $\mathbf{M}_1 \cdot \mathbf{M}_2$。这说明如果要平移和旋转对象，必须注意复合矩阵求值的顺序（参见图 5.13）。对于变换序列中每一个类型都相同的特殊情况，变换矩阵的多重相乘是可交换的。例如，两个连续的旋转可以按两种顺序完成，但其最后位置是相同的。这种交换特性对两个连续的平移或两个连续缩放也同样适用。另一对可交换操作是旋转和一致缩放（$s_x = s_y$）。

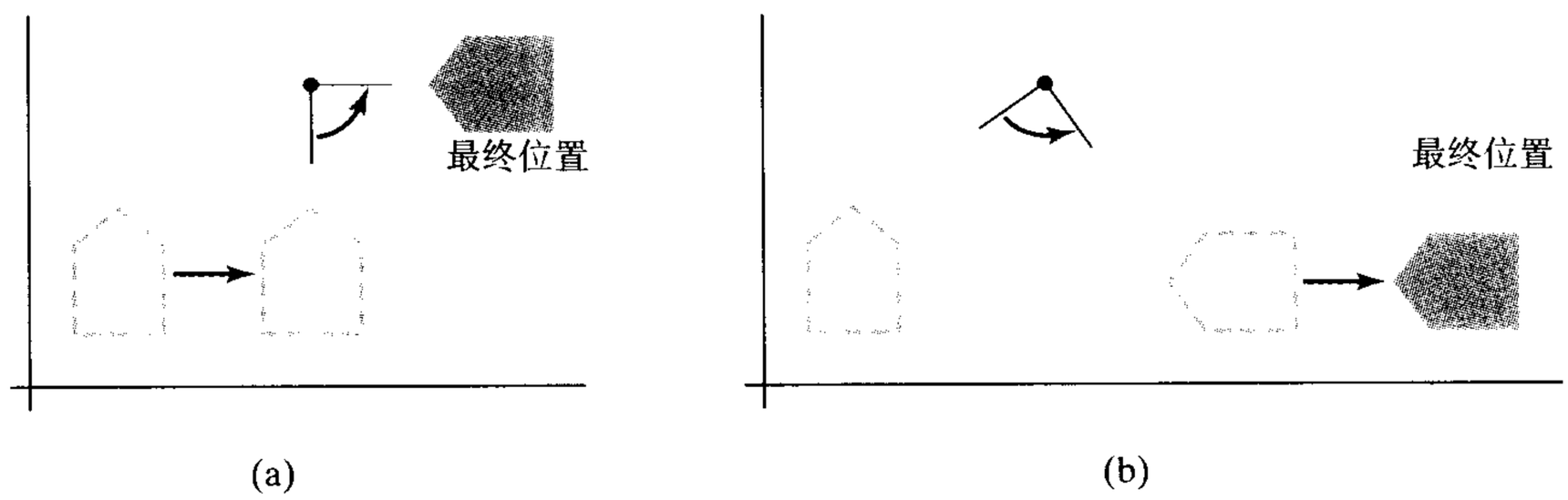


图 5.13 改变变换序列的顺序将影响对象的变换位置。在(a)中对象先平移后旋转，在(b)中对象先旋转后平移

5.4.8 通用二维复合变换和计算效率

表示平移、旋转和缩放组合的通用二维变换可以表示为

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} rs_{xx} & rs_{xy} & trs_x \\ rs_{yx} & rs_{yy} & trs_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5.41}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

4 个元素 rs_{jk} 是变换中（仅包含旋转角和缩放系数）的多重旋转－缩放项。元素 trs_x 和 trs_y 是包含平移距离、基准点和固定点坐标以及旋转角和缩放参数组合的平移项。例如，如果一个对象要对于其中心坐标(x_c, y_c)进行缩放、旋转和平移，那么复合变换矩阵的元素值为

$$\mathbf{T}(t_x, t_y) \cdot \mathbf{R}(x_c, y_c, \theta) \cdot \mathbf{S}(x_c, y_c, s_x, s_y)$$

$$= \begin{bmatrix} s_x \cos\theta & -s_y \sin\theta & x_c(1 - s_x \cos\theta) + y_c s_y \sin\theta + t_x \\ s_x \sin\theta & s_y \cos\theta & y_c(1 - s_y \cos\theta) - x_c s_x \sin\theta + t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.42}$$

尽管矩阵等式（5.41）需要 9 次乘法和 6 次加法，但变换后坐标的显式计算为

$$x' = x \cdot rs_{xx} + y \cdot rs_{xy} + trs_x, \qquad y' = x \cdot rs_{yx} + y \cdot rs_{yy} + trs_y \tag{5.43}$$

因此，实际上变换坐标位置仅需完成4次乘法和4次加法。一旦把单个矩阵连接起来计算出复合矩阵的元素值，这就是任何变换序列所需计算的最大数目。假如没有合并，那么每次都要使用一个单独的变换，则计算的数目将大大增加。因此，变换操作的有效实现是先形成变换矩阵，合并所有变换序列，然后用等式（5.43）计算变换的坐标。在并行系统上，使用等式（5.41）的复合变换矩阵而直接进行矩阵相乘也可以有相同的效果。

由于旋转计算需要对每个变换点进行三角求值和多次乘法，因而在旋转变换中的计算效率就成为十分重要问题。在动画及其他包含许多重复变换和小旋转角的应用中，我们可用近似和循环计算来减少复合变换方程中的计算量。当旋转角较小时，三角函数可用其幂级数展开式的前几项的近似值来代替，对于足够小的角度（小于 10°），$\cos\theta$近似为 1，而 $\sin\theta$的值非常接近于 θ的弧度值。例如，假如以小角度步长绕原点旋转，那么可以将$\cos\theta$设置为 1.0，并在每一步中将变换计算减少为两次乘法和两次加法：

$$x' = x - y \sin\theta, \qquad y' = x \sin\theta + y \tag{5.44}$$

其中，只要旋转角不变化，$\sin\theta$对所有的步长只需求值一次。在每一步中，由这种近似所引起的误差随旋转角的减少而减少。但即使是使用较小的旋转角，很多步之后的积累误差也会变得很大。如果要通过消除每一步中x'和y'的误差来控制积累误差，则必须在误差积累变得太大时重新设置对象位置。有些动画应用自动地在固定间隔处重设对象位置，如每 360° 或 180°。

复合变换经常包括逆矩阵计算。例如，对于通用缩放方向及对于反射和错切（参见 5.5 节）的变换顺序，可以使用逆旋转分量进行描述。我们已经注意到，基本几何变换的逆矩阵表示可以使用简单程序生成，逆平移矩阵可以通过改变平移距离的符号而得到，逆旋转矩阵通过完成矩阵转置（或改变 sin 项的符号）而得到，这些操作比直接逆矩阵计算要简单得多。

5.4.9　二维刚体变换

如果一个变换矩阵仅包含平移和旋转参数，则它是一个**刚体变换矩阵**（rigid-body transformation matrix）。二维刚体变换矩阵的一般形式为

$$\begin{bmatrix} r_{xx} & r_{xy} & tr_x \\ r_{yx} & r_{yy} & tr_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.45}$$

其中，4 个元素 r_{jk} 是多重旋转项，元素 tr_x 和 tr_y 是平移项。坐标位置的刚体变化有时也称为**刚体运动**（rigid -motion）变换。变换后坐标位置间的所有角度和距离都不变化。此外，矩阵（5.45）具有

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

其左上角的 2 × 2 矩阵是一正交矩阵（orthogonal matrix）的特性。这说明，假如将子矩阵的每一行（或每一列）作为一个向量，那么两个行向量(r_{xx}, r_{xy})和(r_{yx}, r_{yy})（或两个列向量）形成单位向量的正交组。这样一组向量也称为正交向量组。每个向量具有单位长度：

$$r_{xx}^2 + r_{xy}^2 = r_{yx}^2 + r_{yy}^2 = 1 \tag{5.46}$$

并且向量相互垂直（它们的点积为零）：

$$r_{xx}r_{yx} + r_{xy}r_{yy} = 0 \tag{5.47}$$

因此，假如这些单位向量通过旋转子矩阵进行变换，那么(r_{xx}, r_{xy})就转换成沿x轴的单位向量，(r_{yx}, r_{yy})转换成沿坐标系统y轴的单位向量：

$$\begin{bmatrix} r_{xx} & r_{xy} & 0 \\ r_{yx} & r_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{xx} \\ r_{xy} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{5.48}$$

$$\begin{bmatrix} r_{xx} & r_{xy} & 0 \\ r_{yx} & r_{yy} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{yx} \\ r_{yy} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \tag{5.49}$$

作为一个例子，下列刚体变换先将对象对于基准点(x_r, y_r)旋转 θ 角，然后平移：

$$\mathbf{T}(t_x, t_y) \cdot \mathbf{R}(x_r, y_r, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta + t_x \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta + t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.50}$$

这里，左上角 2 × 2 子矩阵中的对角单位向量为$(\cos\theta, -\sin\theta)$和$(\sin\theta, \cos\theta)$，并且

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta \\ -\sin\theta \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{5.51}$$

同样，单位向量$(\sin\theta, \cos\theta)$也由前面的变换矩阵转换成 y 方向的单位向量(0, 1)。

5.4.10 构造二维旋转矩阵

在只知道对象的最后方向而不知道将对象放到这个位置所需的旋转角度时，旋转矩阵的正交特性可用于构造矩阵。该方向信息可以根据与场景中某一对象对齐或由场景中所选定的位置来确定。例如，我们可能要将一对象旋转使其与观察（照相机）方向对称的轴对齐，或将一对象转到另一对象之上。图 5.14 给出了将要与单位方向向量 $\mathbf{u}'$ 和 $\mathbf{v}'$ 对齐的对象。假设原始对象方向是与坐标轴对齐的，如图 5.14(a)所示，那么我们可以将 $\mathbf{u}'$ 的元素位置设置给旋转矩阵的第一行，将 $\mathbf{v}'$ 的元素设置给旋转矩阵的第二行，从而构造所期望的变换。例如在建模应用中，当我们知道在整个世界坐标系中的方向时，我们可以利用这种方法从对象局部坐标系得到变换矩阵。类似的变换是将对象描述从一个坐标系转换到另一个坐标系。在 5.8 节和 5.15 节，我们将会详细调整这些方法。

5.4.11 二维复合矩阵编程示例

下面的程序给出了一系列几何变换的实现示例。开始时，复合矩阵 compMatrix 是一个单位矩阵。在本例中，使用从左往右合并的次序来构造复合变换矩阵，且按其执行顺序调用变换子程序。在每一变换函数（缩放、旋转和平移）被调用时，为该变换建立矩阵并从左边去和复合矩阵相结合。在指定完所有变换后，使用复合矩阵对三角形进行变换。该三角形先对其中心位置（参见附录 A）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

进行缩放，然后绕其中心旋转，最后进行平移。图5.15给出了使用该序列变换的三角形的原始位置和最后位置。

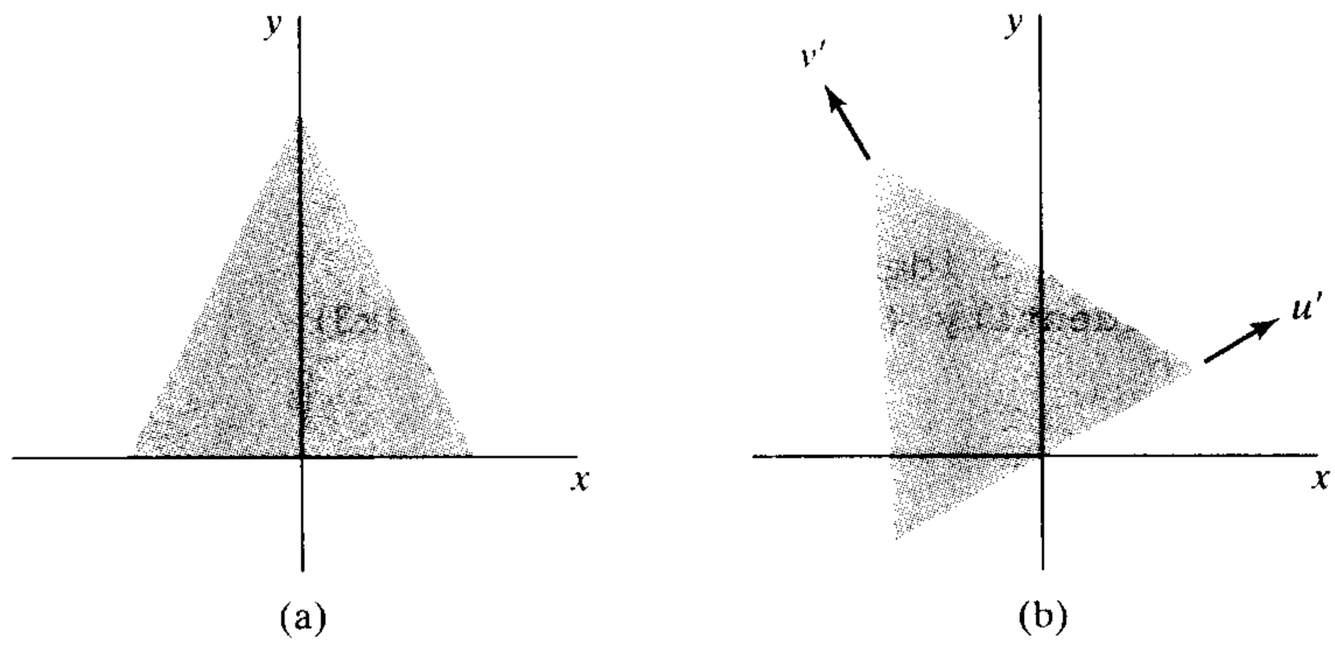


图 5.14　将一个对象从位置(a)旋转到位置(b)的旋转矩阵，可以使用单位方向向量 **u′** 和 **v′** 相对于原始方向的值来构造

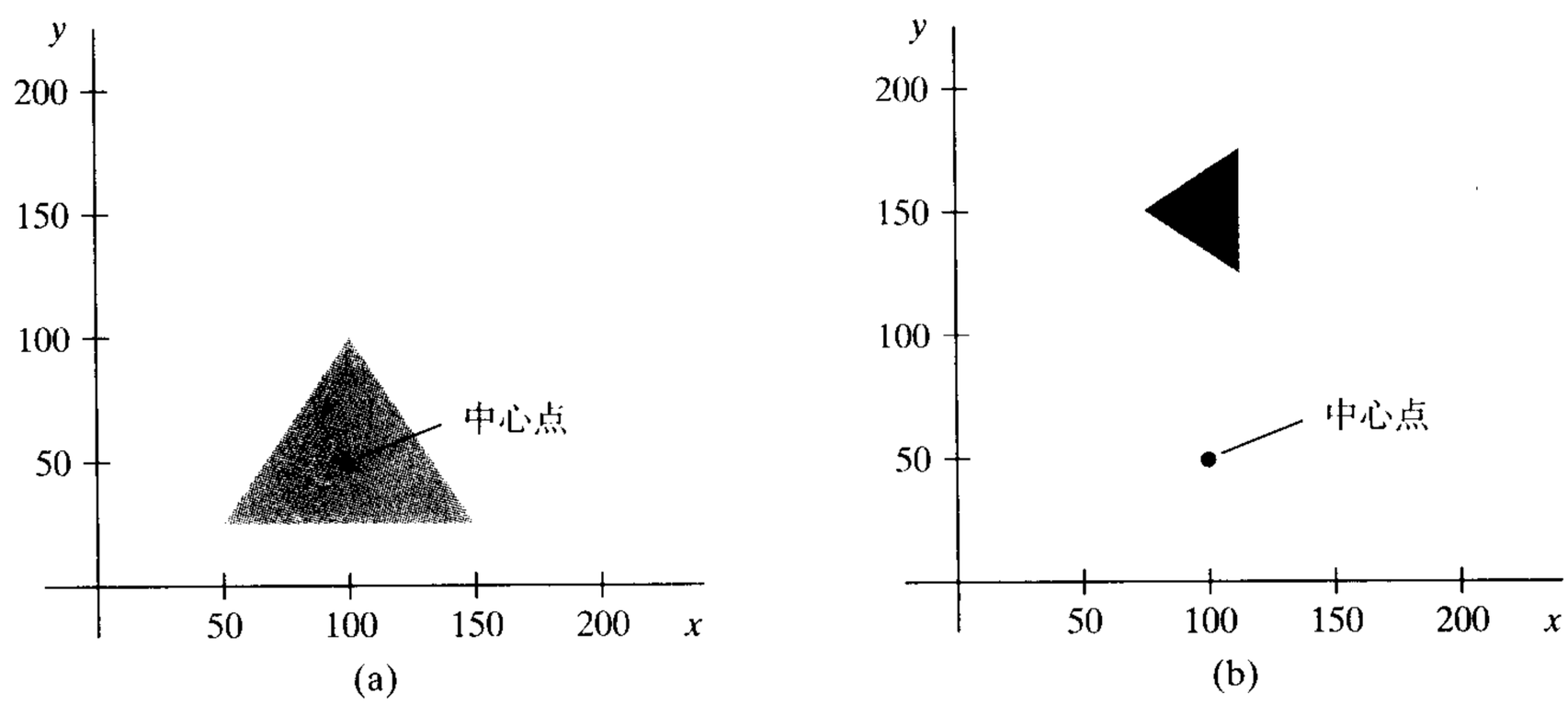


图 5.15　使用过程 transformVerts2D 中的复合矩阵计算将(a)中的三角形变换到(b)中的位置

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

/*  Set initial display-window size.  */
GLsizei winWidth = 600, winHeight = 600;

/*  Set range for world coordinates.  */
GLfloat xwcMin = 0.0, xwcMax = 225.0;
GLfloat ywcMin = 0.0, ywcMax = 225.0;

class wcPt2D {
   public:
      GLfloat x, y;
};

typedef GLfloat Matrix3x3 [3][3];

Matrix3x3 matComposite;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
const GLdouble pi = 3.14159;

void init (void)
{
   /*  Set color of display window to white.  */
   glClearColor (1.0, 1.0, 1.0, 0.0);
}

/* Construct the 3 by 3 identity matrix. */
void matrix3x3SetIdentity (Matrix3x3 matIdent3x3)
{
   GLint row, col;

   for (row = 0; row < 3; row++)
      for (col = 0; col < 3; col++)
         matIdent3x3 [row][col] = (row == col);
}

/* Premultiply matrix m1 times matrix m2, store result in m2. */
void matrix3x3PreMultiply (Matrix3x3 m1, Matrix3x3 m2)
{
   GLint row, col;
   Matrix3x3 matTemp;

   for (row = 0; row < 3; row++)
      for (col = 0; col < 3 ; col++)
         matTemp [row][col] = m1 [row][0] * m2 [0][col] + m1 [row][1] *
                              m2 [1][col] + m1 [row][2] * m2 [2][col];

   for (row = 0; row < 3; row++)
      for (col = 0; col < 3; col++)
         m2 [row][col] = matTemp [row][col];
}

void translate2D (GLfloat tx, GLfloat ty)
{
   Matrix3x3 matTransl;

   /*  Initialize translation matrix to identity.  */
   matrix3x3SetIdentity (matTransl);

   matTransl [0][2] = tx;
   matTransl [1][2] = ty;

   /*  Concatenate matTransl with the composite matrix.  */
   matrix3x3PreMultiply (matTransl, matComposite);
}

void rotate2D (wcPt2D pivotPt, GLfloat theta)
{
   Matrix3x3 matRot;

   /*  Initialize rotation matrix to identity.  */
   matrix3x3SetIdentity (matRot);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
   matRot [0][0] = cos (theta);
   matRot [0][1] = -sin (theta);
   matRot [0][2] = pivotPt.x * (1 - cos (theta)) +
                        pivotPt.y * sin (theta);
   matRot [1][0] = sin (theta);
   matRot [1][1] = cos (theta);
   matRot [1][2] = pivotPt.y * (1 - cos (theta)) -
                        pivotPt.x * sin (theta);

   /*  Concatenate matRot with the composite matrix.  */
   matrix3x3PreMultiply (matRot, matComposite);
}

void scale2D (GLfloat sx, GLfloat sy, wcPt2D fixedPt)
{
   Matrix3x3 matScale;

   /*  Initialize scaling matrix to identity.  */
   matrix3x3SetIdentity (matScale);

   matScale [0][0] = sx;
   matScale [0][2] = (1 - sx) * fixedPt.x;
   matScale [1][1] = sy;
   matScale [1][2] = (1 - sy) * fixedPt.y;

   /*  Concatenate matScale with the composite matrix.  */
   matrix3x3PreMultiply (matScale, matComposite);
}

/* Using the composite matrix, calculate transformed coordinates. */
void transformVerts2D (GLint nVerts, wcPt2D * verts)
{
   GLint k;
   GLfloat temp;

   for (k = 0; k < nVerts; k++) {
      temp = matComposite [0][0] * verts [k].x + matComposite [0][1] *
             verts [k].y + matComposite [0][2];
      verts [k].y = matComposite [1][0] * verts [k].x + matComposite [1][1] *
                    verts [k].y + matComposite [1][2];
         verts [k].x = temp;
   }
}

void triangle (wcPt2D *verts)
{
    GLint k;

    glBegin (GL_TRIANGLES);
       for (k = 0; k < 3; k++)
          glVertex2f (verts [k].x, verts [k].y);
    glEnd ( );
}

void displayFcn (void)
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
{
   /*  Define initial position for triangle.  */
   GLint nVerts = 3;
   wcPt2D verts [3] = { {50.0, 25.0}, {150.0, 25.0}, {100.0, 100.0} };

   /*  Calculate position of triangle centroid.  */
   wcPt2D centroidPt;

   GLint k, xSum = 0, ySum = 0;
   for (k = 0; k < nVerts;  k++) {
      xSum += verts [k].x;
      ySum += verts [k].y;
   }
   centroidPt.x = GLfloat (xSum) / GLfloat (nVerts);
   centroidPt.y = GLfloat (ySum) / GLfloat (nVerts);

   /*  Set geometric transformation parameters.  */
   wcPt2D pivPt, fixedPt;
   pivPt = centroidPt;
   fixedPt = centroidPt;

   GLfloat tx = 0.0, ty = 100.0;
   GLfloat sx = 0.5, sy = 0.5;
   GLdouble theta = pi/2.0;

   glClear (GL_COLOR_BUFFER_BIT);   //  Clear display window.

   glColor3f (0.0, 0.0, 1.0);       //  Set initial fill color to blue.
   triangle (verts);                //  Display blue triangle.

   /*  Initialize composite matrix to identity.  */
   matrix3x3SetIdentity (matComposite);

   /*  Construct composite matrix for transformation sequence.  */
   scale2D (sx, sy, fixedPt);    //  First transformation: Scale.
   rotate2D (pivPt, theta);      //  Second transformation: Rotate
   translate2D (tx, ty);         //  Final transformation: Translate.

   /*  Apply composite matrix to triangle vertices.  */
   transformVerts2D (nVerts, verts);

   glColor3f (1.0, 0.0, 0.0);  // Set color for transformed triangle.
   triangle (verts);           // Display red transformed triangle.

   glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );
   gluOrtho2D (xwcMin, xwcMax, ywcMin, ywcMax);

   glClear (GL_COLOR_BUFFER_BIT);
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void main (int argc, char ** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (50, 50);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Geometric Transformation Sequence");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);

   glutMainLoop ( );
}
```

5.5 其他二维变换

大多数图形软件包中包含了类似平移、旋转和缩放这些基本变换。有些软件包还提供一些对某些应用有用的其他变换。反射和错切是这样的两个变换。

5.5.1 反射

产生对象镜像的变换称为**反射**（reflection）。对于一个二维反射而言，其反射镜像通过将对象绕反射轴旋转180°而生成。我们可以在xy平面内或垂直于xy平面选择**反射轴**（axis of reflection）。当反射轴是xy平面内的一条直线时，绕这个轴的旋转路径在垂直于xy平面的平面中；而对于垂直于xy平面的反射轴，旋转路径在xy平面内。下面举出一些普通反射的例子。

关于直线$y=0$（x轴）的反射，可以由下列变换矩阵来完成：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.52}$$

这个变换保持x值相同，但“翻动”y坐标位置的值。对象对于x轴反射后的方位示于图5.16中。为了想像这种反射的旋转变换，我们可以认为：平面上的对象移出xy平面，通过三维空间绕x轴旋转180°再回到x轴另一侧的xy平面。

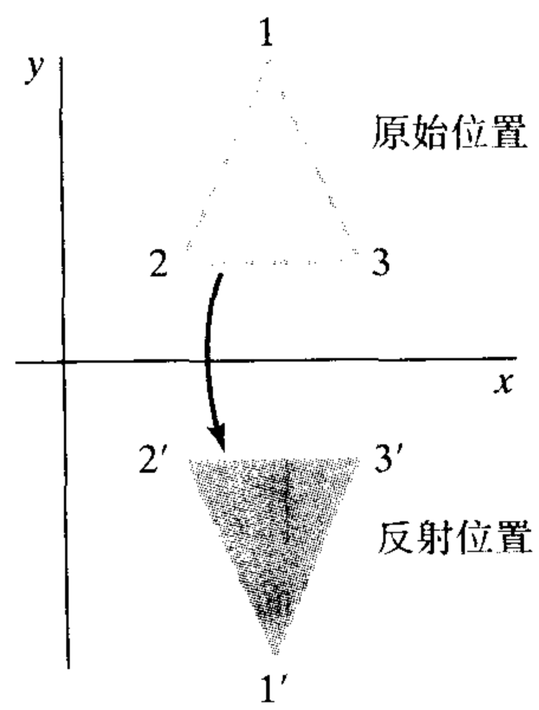


图5.16 相对于x轴的对象反射

对于$x=0$（y轴）的反射，翻动x的坐标而保持y坐标不变，这种变换的矩阵是

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.53}$$

图 5.17 给出了关于直线 $x=0$ 反射的对象的位置变化。这时的等量旋转是通过三维空间绕 y 轴旋转 180°。

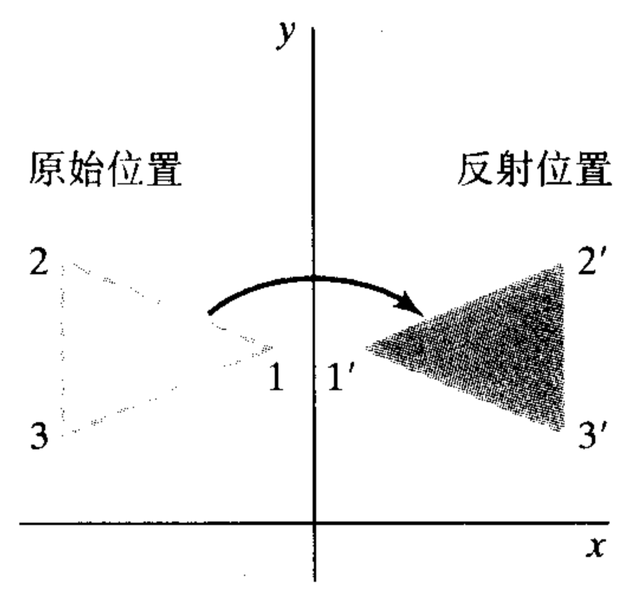


图 5.17 相对于 y 轴的对象反射

我们可以通过相对于垂直 xy 平面而又经过坐标原点的轴进行反射，并且同时翻动点的 x 和 y 坐标，这种反射称为相对于坐标原点的反射，它与同时相对于两个坐标轴的反射等价。这种反射的变换矩阵表示为

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.54}$$

关于原点反射的例子如图 5.18 所示。反射矩阵（5.54）是 $\theta=180°$ 的旋转矩阵 $\mathbf{R}(\theta)$，也就是将 xy 平面内的对象绕原点旋转半圈。

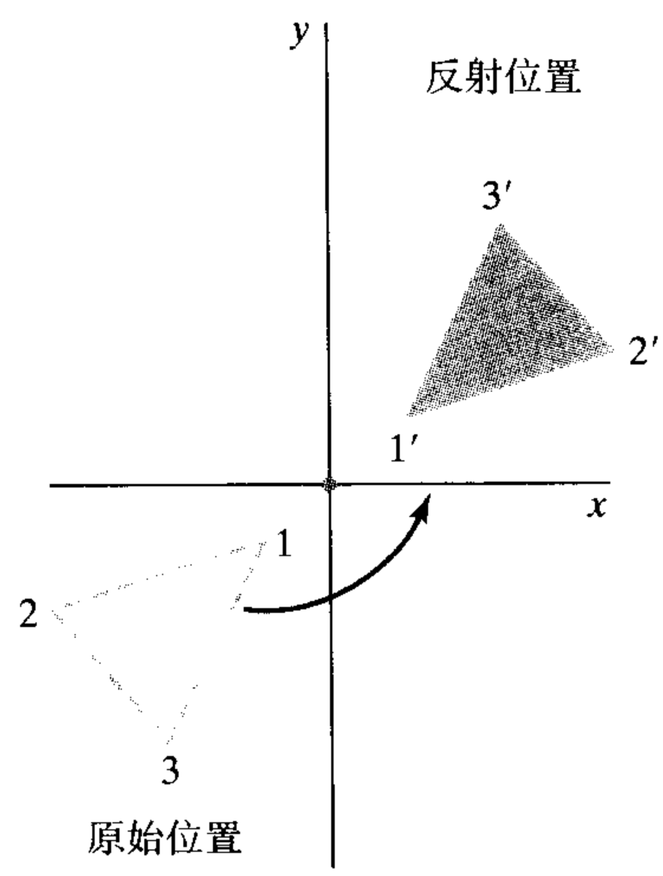


图 5.18 对象相对于垂直 xy 平面并通过坐标原点的轴进行反射

反射矩阵（5.54）可以一般化为 xy 平面内的任何反射点（参见图 5.19），这种反射与将反射点作为基准点并在 xy 平面内旋转 180° 是相同的。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

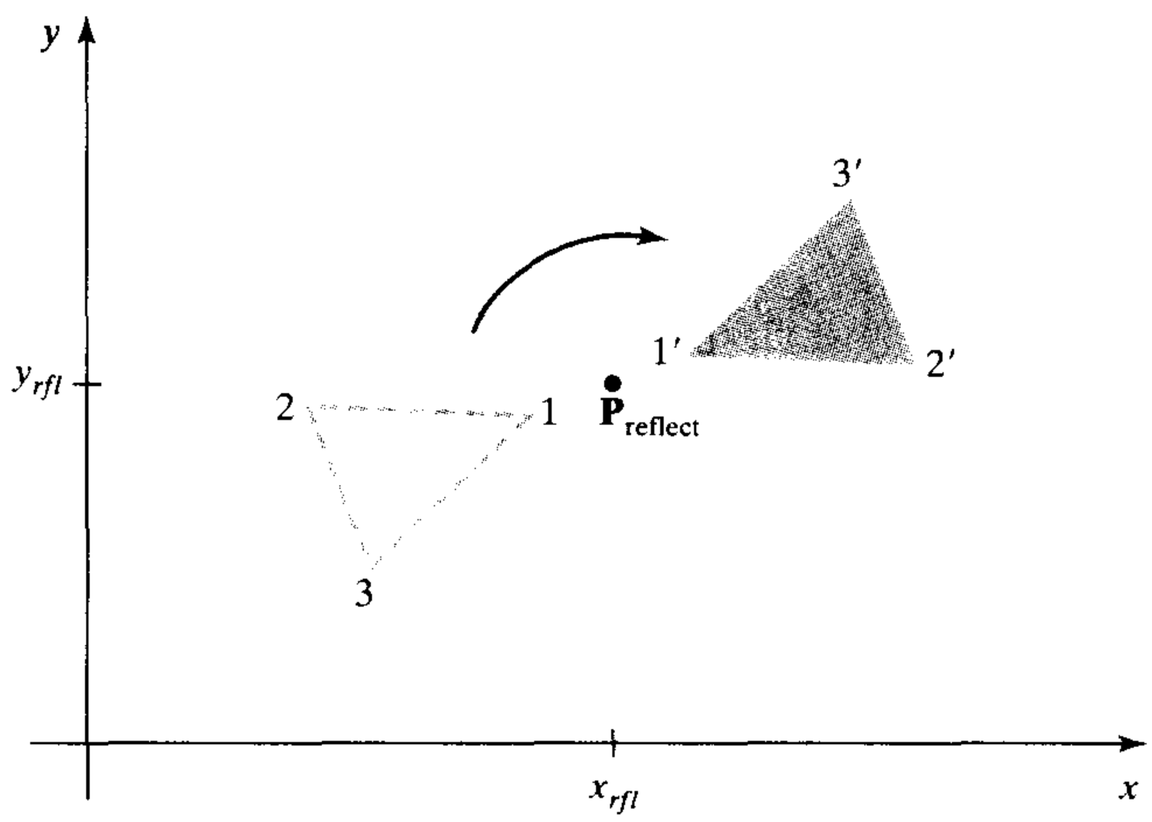


图 5.19　对象相对于垂直 xy 平面并通过点 $\mathbf{P}_{reflect}$ 的轴的反射

假如我们将对角线 $y = x$ 选为反射轴（参见图 5.20），那么反射矩阵为

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.55}$$

可以通过将一系列的旋转和坐标轴反射矩阵合并来推导出上述矩阵。图 5.21 给出了一个可能的顺序。这里首先完成顺时针 45° 旋转，将直线 $y = x$ 旋转到 x 轴上；接着完成对于 x 轴的反射；最后逆时针旋转 45°，将直线 $y = x$ 旋转回到其原始位置。另一个等价的变换是先将对象关于 x 轴反射，然后逆时针旋转 90°。

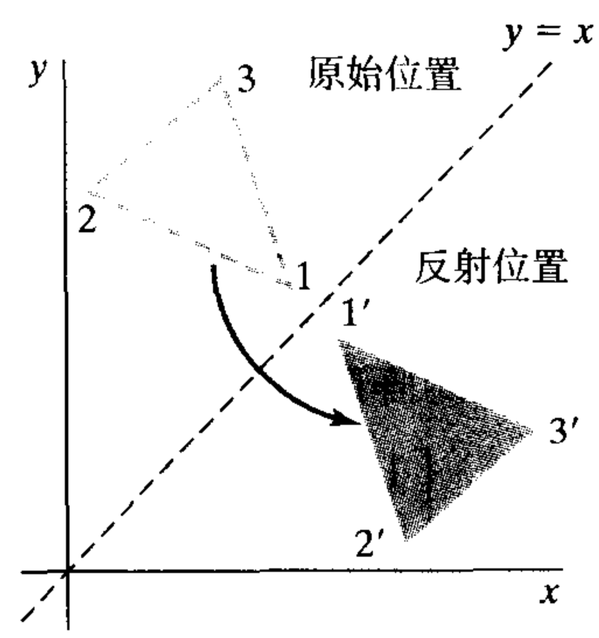


图 5.20　对象对于直线 $y = x$ 的反射

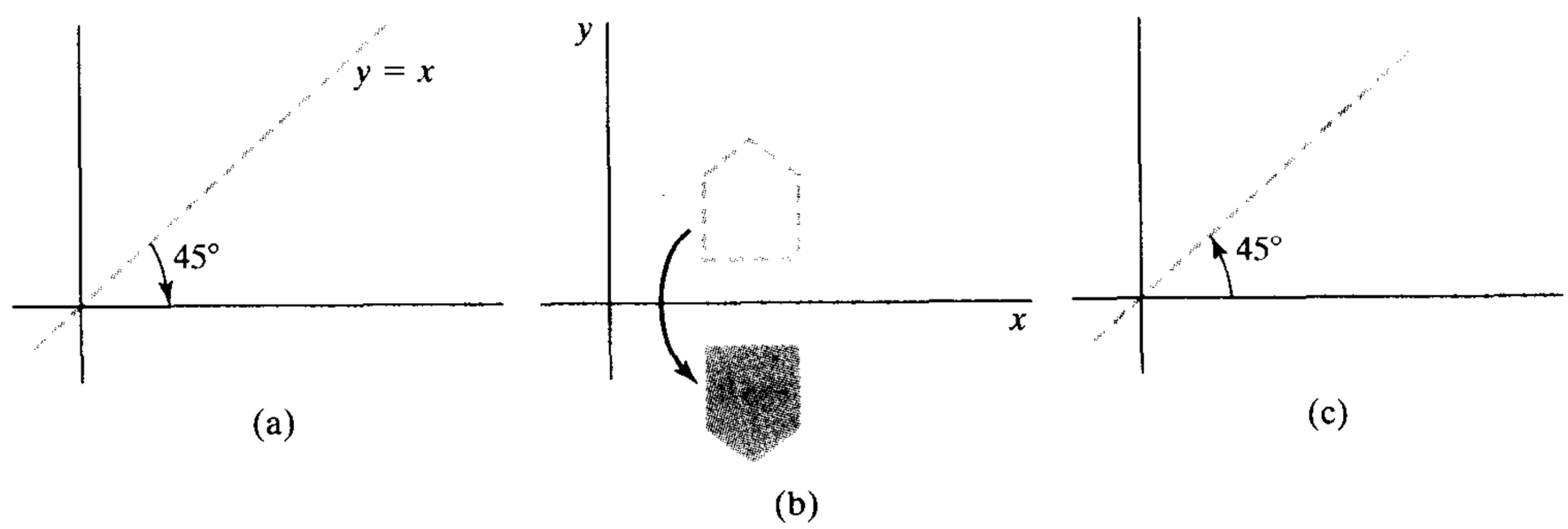


图 5.21　对于直线 $y = x$ 产生反射的变换序列：(a)顺时针旋转 45°；(b)对于 x 轴反射；(c)逆时针旋转 45°

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

为了得到关于对角线 $y=-x$ 反射的变换矩阵，我们按下列变换顺序合并矩阵：(1)顺时针旋转 45°，(2)对于 y 轴反射，(3)逆时针旋转 45°，导致的变换矩阵为

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.56}$$

图 5.22 给出了使用这个反射矩阵变换对象的原始和最终位置。

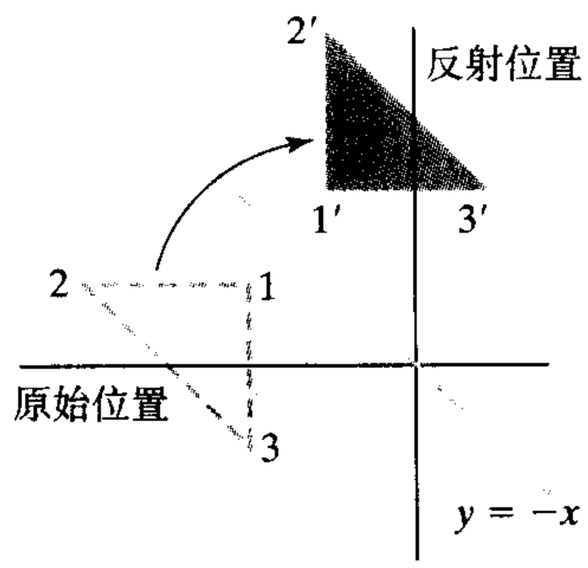


图 5.22 相对于直线 $y=-x$ 的反射

关于 xy 平面内任意直线 $y=mx+b$ 的反射，可以使用平移－旋转－反射变换的组合而完成。通常，我们先平移直线使其经过原点。然后将直线旋转到坐标轴之一，并关于坐标轴反射。最后利用逆旋转和逆平移变换将直线还原到原来位置。

我们可以将关于坐标轴或坐标原点的反射处理为缩放系数为负值的缩放变换。同样，反射矩阵的元素也可设置为 ±1 以外的其他值。大于 1 的值将镜像移至远离反射轴的位置，小于 1 的值将镜像移至接近反射轴的位置。因此，反射后的对象也可能放大、缩小或变形。

5.5.2 错切

错切（shear）是一种使对象形状发生变化的变换，经过错切的对象好像是由已经相互滑动的内部夹层组成。两种常用的错切变换是移动 x 坐标值的错切和移动 y 坐标值的错切。

相对于 x 轴的 x 方向错切由下列变换矩阵产生：

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.57}$$

该矩阵将坐标位置转换成

$$x' = x + sh_x \cdot y, \qquad y' = y \tag{5.58}$$

可以将任意实数赋给错切参数 sh_x。然后将坐标位置 (x, y) 水平地移动与其到 x 轴（$y=0$）的距离（y 值）成正比的量。例如，设置 sh_x 为 2，将图 5.23 中的正方形变为平行四边形。sh_x 为负值则将坐标位置向左移动。

我们可以使用下列矩阵生成相对于其他参考线的 x 方向的错切：

$$\begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{\text{ref}} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.59}$$

现在，坐标位置将变换为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$x' = x + sh_x(y - y_{\text{ref}}), \qquad y' = y \tag{5.60}$$

图 5.24 中给出了错切参数为 1/2、相对于直线 $y_{\text{ref}} = -1$ 的错切变换例子。

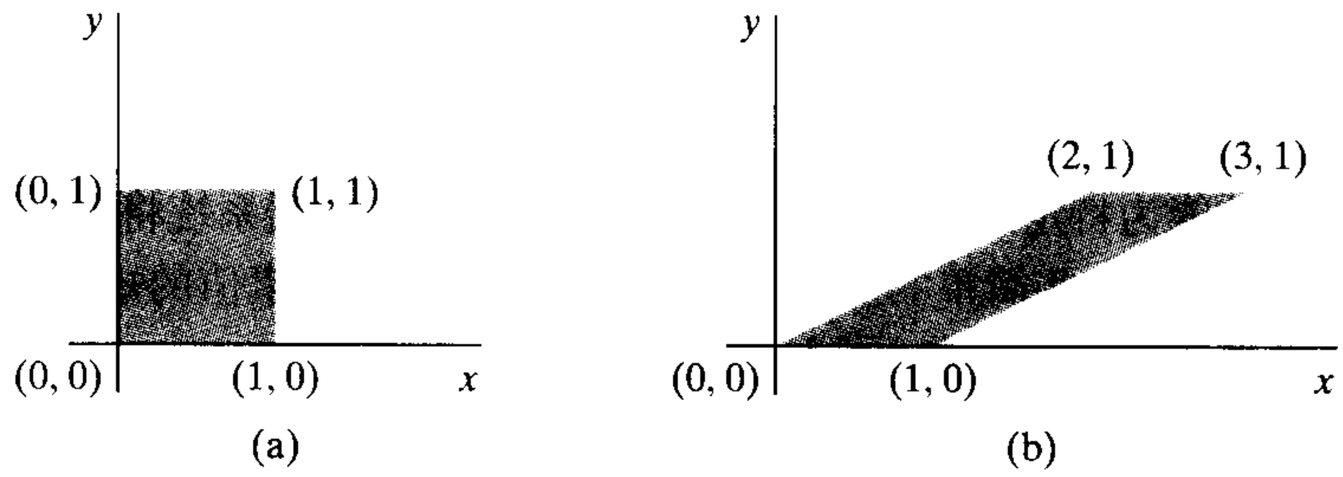


图 5.23　使用 $sh_x = 2$ 的 x 方向错切矩阵（5.57），将单位正方形(a)转换为平行四边形(b)

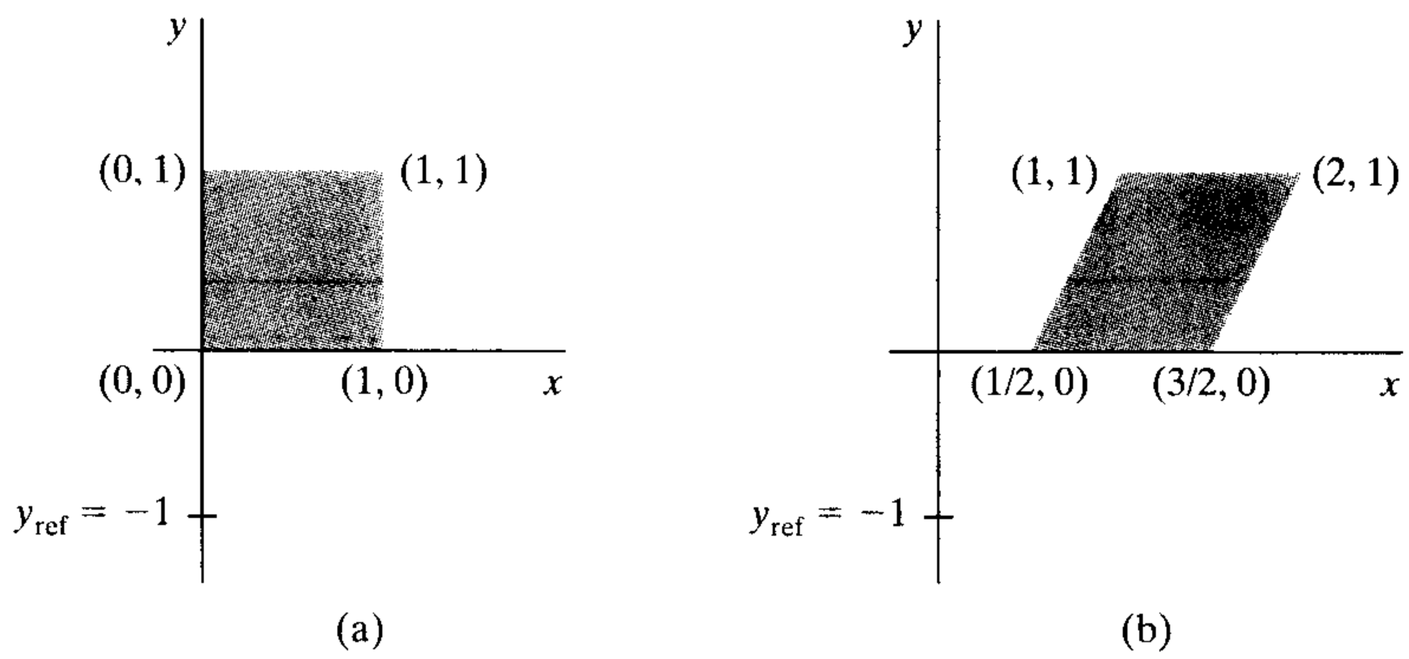


图 5.24　根据错切矩阵（5.59）中的 $sh_x = 0.5$ 和 $y_{\text{ref}} = -1$，将单位正方形(a)变换成移位的平行四边形(b)

使用下列变换矩阵生成相对于线 $x = x_{\text{ref}}$ 的 y 方向错切：

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{\text{ref}} \\ 0 & 0 & 1 \end{bmatrix} \tag{5.61}$$

该矩阵生成变换的坐标位置：

$$x' = x, \qquad y' = y + sh_y(x - x_{\text{ref}}) \tag{5.62}$$

这种变换根据正比于其到参考线 $x = x_{\text{ref}}$ 距离的量而垂直地改变坐标位置。图 5.25 给出了使用 $sh_y = 0.5$ 和 $x_{\text{ref}} = -1$ 将正方形转换成平行四边形。

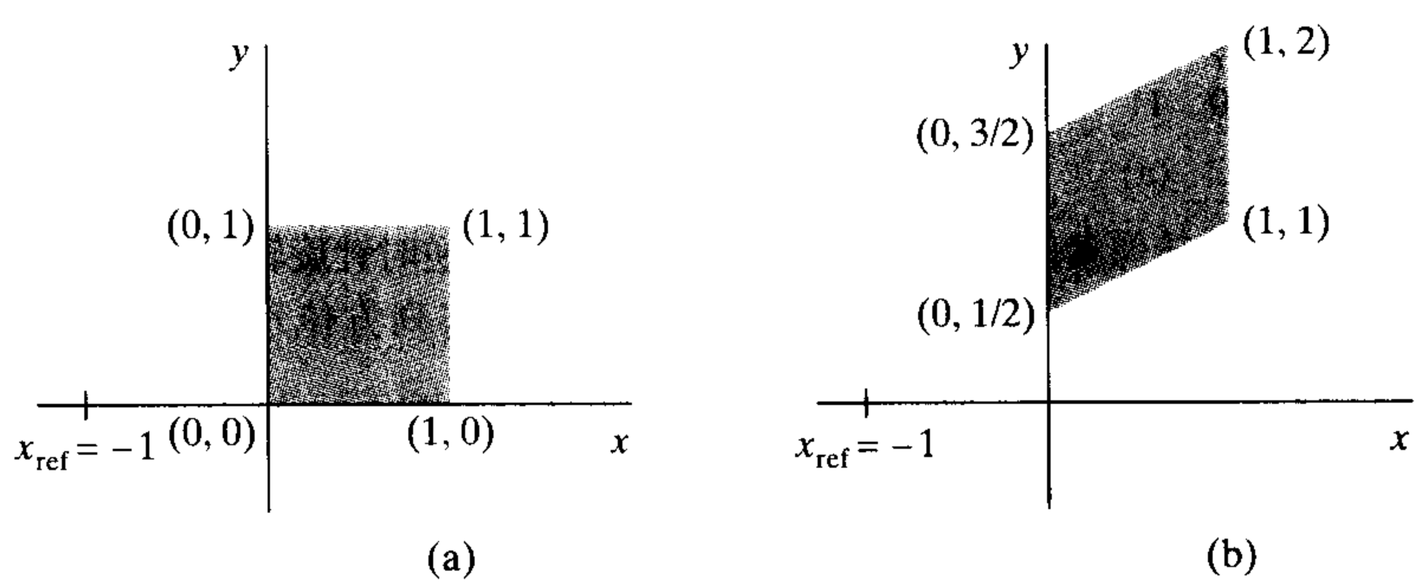


图 5.25　利用错切变换矩阵（5.61），用参数值 $sh_y = 0.5$ 和 $x_{\text{ref}} = -1$ 在 y 方向将单位正方形(a)变换成移位的平行四边形(b)

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

错切操作可以表示为基本变换的序列。例如，x方向的错切矩阵（5.57）可以写为复合变换矩阵，包含一系列旋转和沿对角线缩放图5.23中的单位正方形的缩放矩阵，同时保持了边的原始长度以及与x轴的平行性。在相对于错切参考线的对象位置处的移位等价于平移。

5.6 几何变换的光栅方法

光栅系统的特殊功能为特定的二维变换提供了另一种方法。光栅系统将图像信息作为颜色图案存储在帧缓冲器中。因此，一些简单的变换可以通过操纵储存的像素值的矩形数组而快速地执行。而且只需很少的算术操作，因此像素变换特别有效。

正如3.19节指出的，控制矩形像素数组的光栅功能通常称为光栅操作（raster operation），将一块像素从一个位置移到另一位置的过程也称为像素值的块移动（block transfer）。图形软件中通常包含完成某些光栅操作的子程序。

图5.26给出了作为光栅缓存区域的块移动而完成的二维平移。所有在矩形区域显示的位作为一个块而复制到光栅的另一部分。可以通过使用背景亮度来填充该块的矩形区域来删除原始对象（假设场景中要删除的图案不覆盖其他对象）。

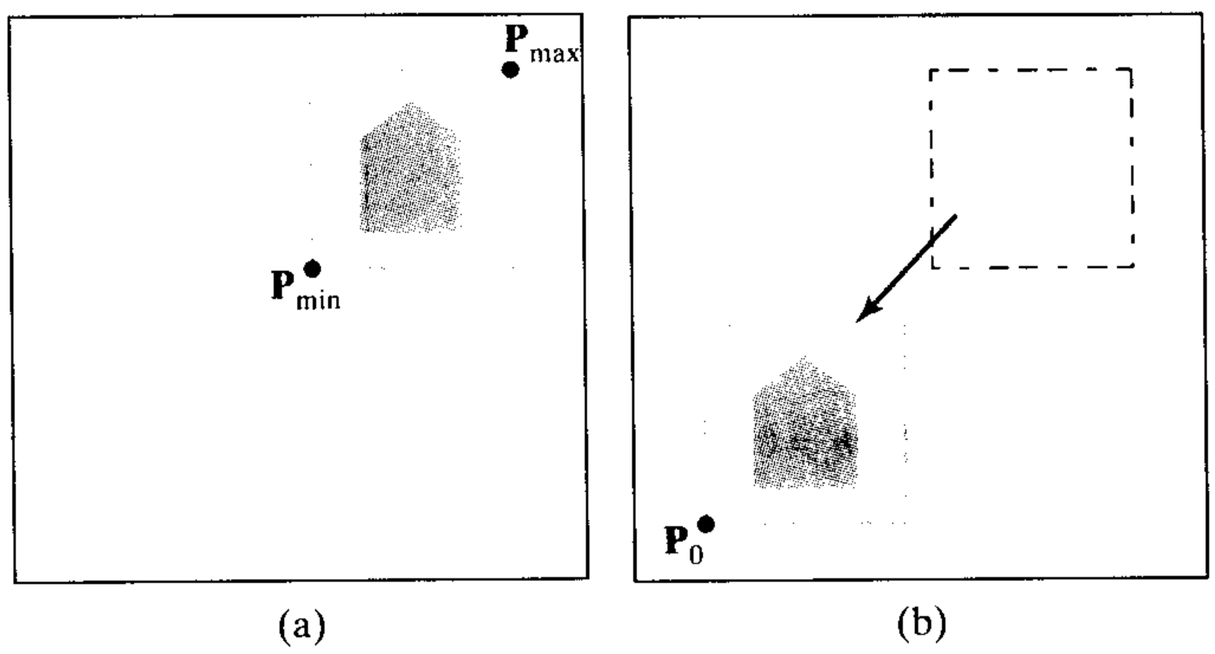


图5.26 通过移动像素值的矩形块，将一个对象从屏幕位置(a)移到位置(b)，坐标位置$\mathbf{P}_{min}$和$\mathbf{P}_{max}$指定了将要移动的矩形块的界限，$\mathbf{P}_0$则是目标参考位置

90°倍数的旋转可以很容易地利用重新安排像素矩阵的元素而实现。通过首先将阵列的每一行的像素值颠倒，然后交换其行和列来将对象逆时针旋转90°；通过颠倒阵列的每一行中元素的顺序，然后将行的顺序颠倒来得到180°的旋转。图5.27给出了将像素块旋转90°和180°所需的数组管理。

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \quad \begin{bmatrix} 3 & 6 & 9 & 12 \\ 2 & 5 & 8 & 11 \\ 1 & 4 & 7 & 10 \end{bmatrix} \quad \begin{bmatrix} 12 & 11 & 10 \\ 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

(a) (b) (c)

图5.27 旋转像素值阵列，原始阵列方位示于(a)，逆时针旋转90°后的网格方位示于(b)，180°旋转后的阵列方位示于(c)

对于不是90°倍数的阵列旋转，则需要完成更多的计算。图5.28给出了这一通用过程。将每个目标区域映射到旋转的网格中，并计算其与旋转的像素区域的重叠量。然后，通过对覆盖的源像素亮度求得平均值，并通过区域重叠的百分比加权来计算目标像素的亮度。或使用像反走样那样的近似方法来确定目标像素的颜色。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

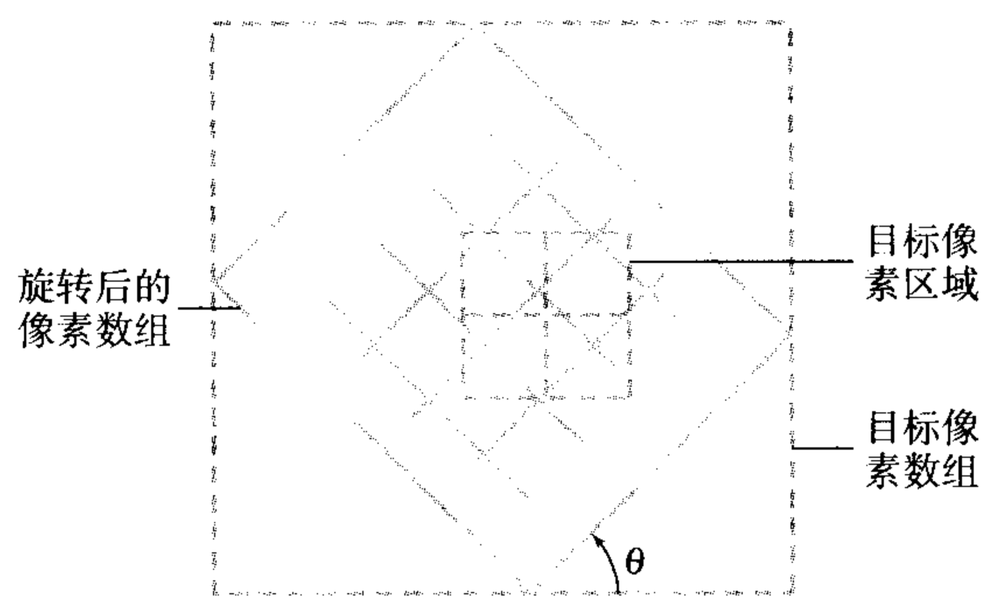


图 5.28　像素的矩形块的旋转，通过将目标像素区域映射到旋转后的像素块上而完成

像素块的光栅缩放采用类似方法实现。我们用指定的 s_x 和 s_y 值对原始块中的像素区域进行缩放，并将缩放的矩形映射到一组目标像素上，然后按照其与缩放像素区域的重叠区域，设置每个目标像素的亮度（参见图 5.29）。

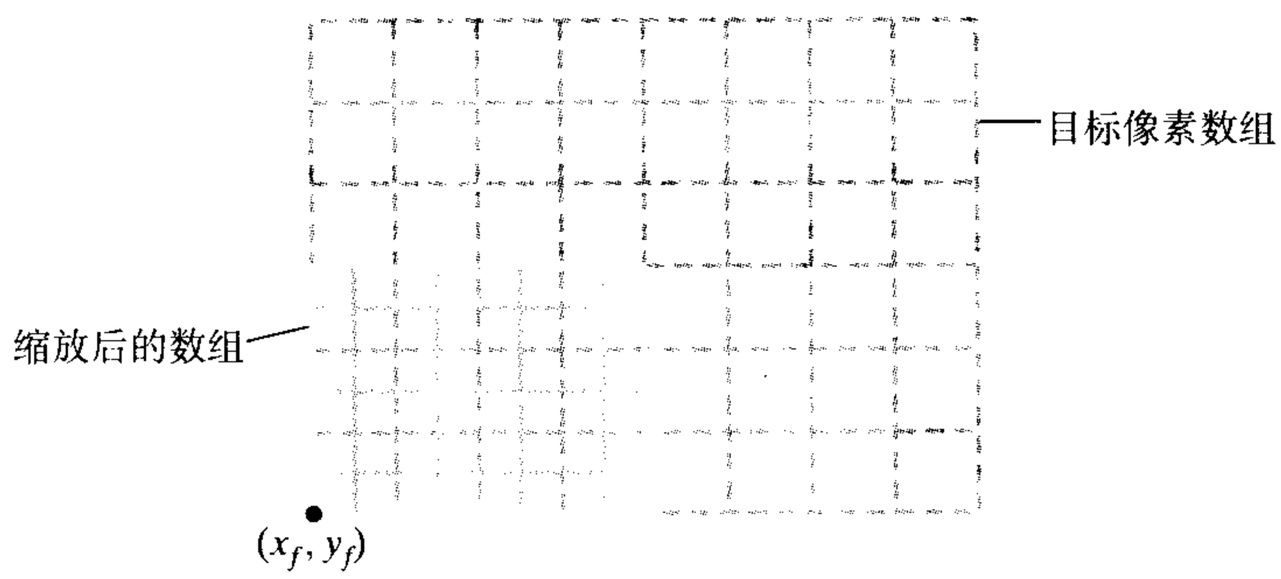


图 5.29　将目标像素区域映射到像素值的缩放后的数组中，
相对于固定点(x_f, y_f)使用缩放系数 $s_x=s_y=0.5$

使用在像素块中颠倒行或列的变换，并与平移结合，可以完成光栅对象的反射。沿行或列移动像素位置则可实现错切。

5.7　OpenGL 光栅变换

在3.19节中，我们介绍了OpenGL中完成光栅操作的大部分函数。像素颜色值的矩形数组从一个缓存到另一个的平移可以作为 OpenGL 的复制操作来完成：

```
glCopyPixels (xmin, ymin, width, height, GL_COLOR);
```

前面4个参数给出了像素块的位置和尺寸。而OpenGL符号常量GL_COLOR指定要复制的颜色值。该像素数组复制到刷新缓存中由当前光栅位置指定的左下角的一个矩形区域内。像素颜色值依赖于颜色模式的当前设定，按 RGBA 或颜色表进行复制 。提供复制的区域（源）和复制目标区域均应位于屏幕坐标边界内。该平移可作用于任何刷新缓存或在不同缓存之间。glCopyPixels 函数的源缓存用 glReadBuffer 子程序选择，而目标缓存用 glDrawBuffer 子程序选择。

通过先将块存储于一个数组，再重新安排数组的元素并将其放回刷新缓存，可以实现像素颜色块的90°倍数的旋转。如同在3.19节所见，缓存中的一个RGB颜色块可以用下列函数存入一个数组：

```
glReadPixels (xmin, ymin, width, height, GL_RGB,
                GL_UNSIGNED_BYTE, colorArray);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如果颜色表索引存在像素位置，则将 GL_COLOR_INDEX 取代 GL_RGB。为了旋转颜色值，必须如前一节所述重新安排颜色数组的行与列。然后使用下列语句将旋转后的数组放回缓存：

```
glDrawPixels (width, height, GL_RGB, GL_UNSIGNED_BYTE, colorArray);
```

该数组的左下角放到当前光栅位置。我们用 glReadBuffer 选择包含原来的像素值块的源缓存，用 glDrawBuffer 指定目标缓存。

二维缩放变换通过指定缩放因子然后引用 glCopyPixels 或 glDrawPixels 按 OpenGL 中的光栅操作来完成。对于光栅操作，使用下列函数来设定缩放因子：

```
glPixelZoom (sx,sy);
```

这里，参数 sx 和 sy 可赋以任何非零浮点值。大于 1.0 的正值增大源数组元素的尺寸，而小于 1.0 的正值减少元素尺寸。sx 或 sy 中有负值或两个都为负值则生成该数组元素的反射以及缩放。因此，如果 sx = sy = −3.0，则源数组相对于当前光栅位置反射且数组的每一颜色元素映射到目标缓存中的 3 × 3 像素块。如果目标像素的中心位于一数组缩放颜色元素的矩形区域，则用该数组元素给它赋值。中心在缩放数组元素左边界或上边界的目标像素也赋以该元素的颜色。sx 和 sy 的默认值均为 1.0。

也可以使用 3.19 节讨论的逻辑操作组合光栅变换来生成各种效果。例如，一个像素数组用异或操作连续两次复制到同一缓存区域，以恢复在该区域的原始值。该技术可用于将一对象平移一场景而不改变背景像素的动画应用。

5.8 二维坐标系间的变换

计算机图形应用经常需要在场景处理的各阶段将对象的描述从一个坐标系变换到另一个坐标系。观察子程序将对象描述从世界坐标系变换到设备坐标系。对于建模和设计应用，每个对象在各自的局部笛卡儿坐标系中设计。这些局部坐标描述必须接着变换到整个场景坐标系的相应位置和方向。例如，办公室布局的设备管理程序具有单独描述椅子、桌子和其他家具的坐标参照系，椅子和其他家具可以放置到楼层平面上并在不同位置有多个副本。

同样，有时场景在利用对象对称性的非笛卡儿参考系中描述。在这些系统中的对象描述必须转换到笛卡儿世界坐标系中进行处理。非笛卡儿系统的例子有极坐标系统、球面坐标系统、椭圆坐标系统和抛物线坐标系统。附录 A 给出了笛卡儿坐标系统与某些常见的非笛卡儿坐标系统之间的关系。这里，我们仅考虑包括从一个二维笛卡儿坐标系到另一个的转换中的变换。

图5.30给出了一个在笛卡儿坐标系统 xy 中用坐标原点 (x_0, y_0) 及方向角 θ 指定的笛卡儿坐标系统 $x'y'$。为了将对象描述从 xy 坐标变换到 $x'y'$ 坐标，必须建立把 $x'y'$ 轴叠加到 xy 轴的变换，这需要分两步进行：

1. 将 $x'y'$ 系统的坐标原点 (x_0, y_0) 平移到 xy 系统的原点 $(0, 0)$；
2. 将 x' 轴旋转到 x 轴上。

坐标原点的平移可以使用下列矩阵操作表示：

$$\mathbf{T}(-x_0, -y_0) = \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.63}$$

平移操作后两个系统的方位如图 5.31 中所示。为了将两个系统的轴重合，可以顺时针旋转：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{R}(-\theta)=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \tag{5.64}$$

将这两个变换矩阵合并起来，就给出了将对象描述从 xy 系统转换到$x'y'$系统的完整复合矩阵：

$$\mathbf{M}_{xy,x'y'}=\mathbf{R}(-\theta)\cdot\mathbf{T}(-x_0,-y_0) \tag{5.65}$$

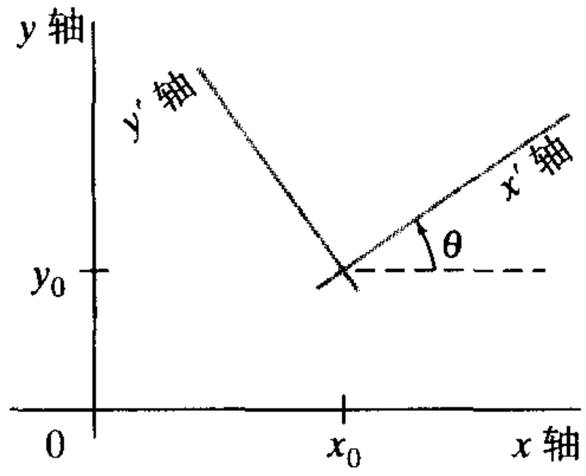


图 5.30　xy 笛卡儿系统中定位在(x_0, y_0)、具有方位 θ 的笛卡儿 $x'y'$ 系统

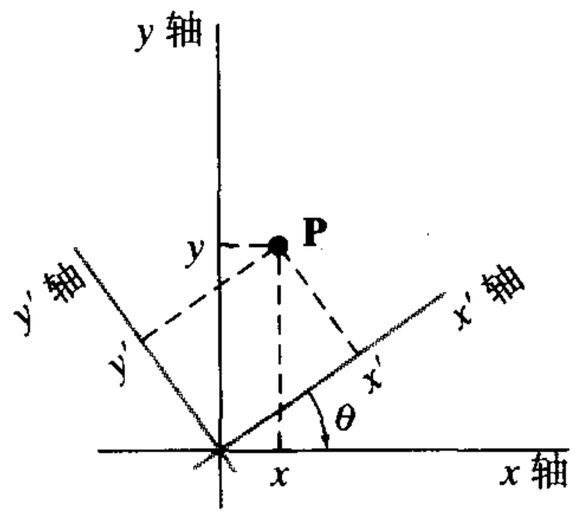


图 5.31　将 $x'y'$ 系统的原点平移到 xy 系统原点后，图 5.30 中的参照系的位置

给出第二个坐标系方向的另一种方法是像图 5.32 所示的那样，指定一个表明正 y' 轴方向的向量 **V**。将向量 **V** 指定为 xy 参照系中相对于 xy 坐标系原点的一个点。那么，在 y' 方向上的单位向量可以计算为

$$\mathbf{v}=\frac{\mathbf{V}}{|\mathbf{V}|}=(v_x, v_y) \tag{5.66}$$

通过将 **v** 顺时针旋转 90°，我们可以得到沿 x' 轴的单位向量 **u**：

$$\begin{aligned}\mathbf{u}&=(v_y, -v_x)\\ &=(u_x, u_y)\end{aligned} \tag{5.67}$$

在5.4节我们指出，任何旋转矩阵的元素可以表示为一组正交单位向量的元素。因此，将$x'y'$系统旋转到与 xy 系统重合的矩阵可以写为

$$R=\begin{bmatrix}u_x & u_y & 0\\ v_x & v_y & 0\\ 0 & 0 & 1\end{bmatrix} \tag{5.68}$$

例如，如果选择 y' 轴的方向为 $\mathbf{V}=(-1,0)$，那么 x' 轴在正 y 方向，且旋转变换矩阵为

$$\begin{bmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}$$

同样地，我们将等式（5.64）中的方位角 θ 设置为 $\theta=90°$ 而得到上述旋转矩阵。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

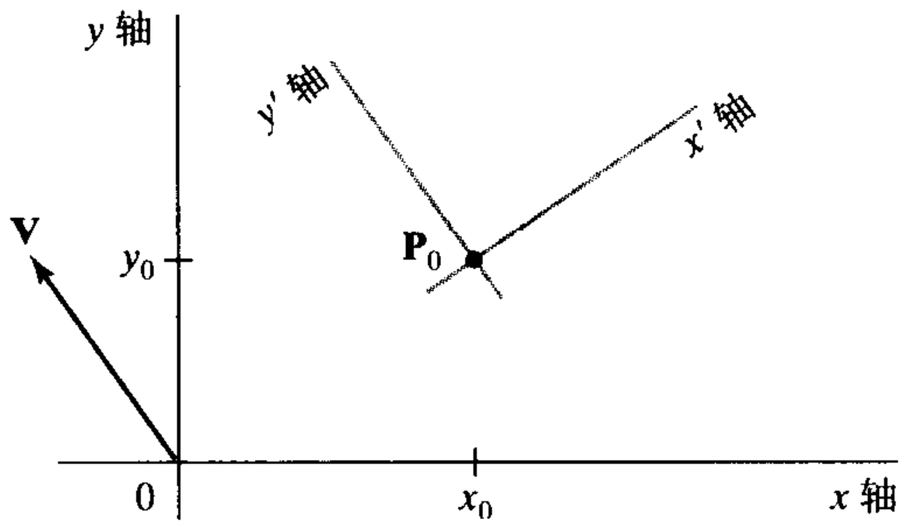


图 5.32 原点在 $\mathbf{P}_0=(x_0, y_0)$、y' 轴平行于向量 $\mathbf{V}$ 的笛卡儿坐标系统 $x'y'$

在交互式应用中，为 $\mathbf{V}$ 选择相对于位置 $\mathbf{P}_0$ 的方位，要比相对于 xy 坐标原点指定其方向更方便。然后，可像图 5.33 所示的那样确定单位向量 $\mathbf{u}$ 和 $\mathbf{v}$ 的方向，此时 $\mathbf{v}$ 的分量计算为

$$\mathbf{v}=\frac{\mathbf{P}_1-\mathbf{P}_0}{|\mathbf{P}_1-\mathbf{P}_0|} \tag{5.69}$$

$\mathbf{u}$ 则垂直于 $\mathbf{v}$ 并形成右手笛卡儿坐标系统。

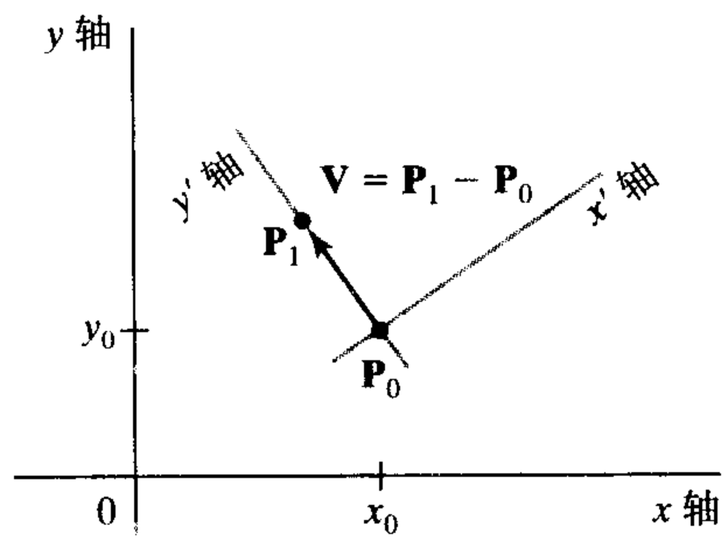


图 5.33 在 xy 参考系内，使用两个坐标位置 $\mathbf{P}_0$ 和 $\mathbf{P}_1$ 定义的笛卡儿 $x'y'$ 坐标系

5.9 三维空间的几何变换

三维几何变换的方法是在二维方法的基础上考虑了 z 坐标而得到的。可以通过指定一个表示对象在三个坐标方向移动距离的三维变换向量来对对象进行平移变换。类似地，也可以利用三个坐标上的缩放因子来缩放对象。然而，三维旋转的扩展则不是那么简单。

当我们讨论 xy 平面上的二维旋转时，只需考虑沿着垂直于 xy 平面的坐标轴进行旋转；而在三维空间中，可能选择空间的任意方向作为旋转轴方向。大多数图形软件将三维旋转作为绕三个坐标轴的二维旋转的复合而进行处理。另一种解决方案则是用户根据给定轴的方向和旋转角度建立一个总的旋转矩阵。

一个三维位置在齐次坐标中表示为4元列向量。因此，现在的每一几何变换操作是一个从左边去乘坐标向量的 4×4 矩阵。和二维中一样，任意变换序列通过依序合并单个变换矩阵而得的一个矩阵表示。变换序列中每一后继矩阵从左边去和以前的变换矩阵合并。

5.10 三维平移

在三维齐次坐标表示中，任意点 $\mathbf{P}=(x, y, z)$ 通过将平移距离 t_x、t_y 和 t_z 加到 $\mathbf{P}$ 的坐标上而平移到位置 $\mathbf{P}'=(x', y', z')$：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$x' = x + t_x, \qquad y' = y + t_y, \qquad z' = z + t_z \tag{5.70}$$

图 5.34 给出了三维点的平移。

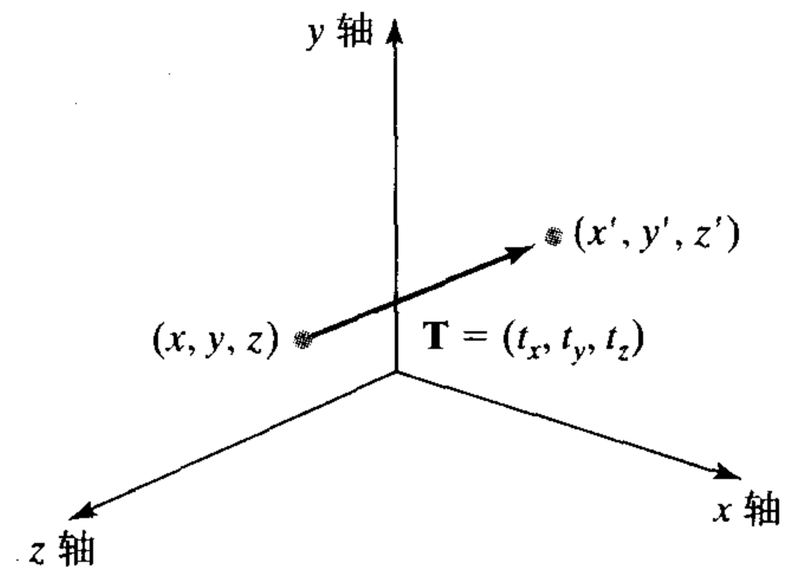


图 5.34　利用变换向量 $\mathbf{T} = (t_x, t_y, t_z)$对某点进行平移变换

我们可以用等式（5.17）中的矩阵形式来表达三维平移操作。但现在坐标位置 **P** 和 **P′** 用 4 元列向量的齐次坐标表示，且变换操作 **T** 是 4 × 4 矩阵：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5.71}$$

或

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P} \tag{5.72}$$

在三维空间中，对象的平移通过平移定义该对象的各个点然后在新位置重建该对象而实现。对于由一组多边形表面表示的对象，可以将各个表面的顶点进行平移（参见图 5.35），然后重新显示新位置的面。

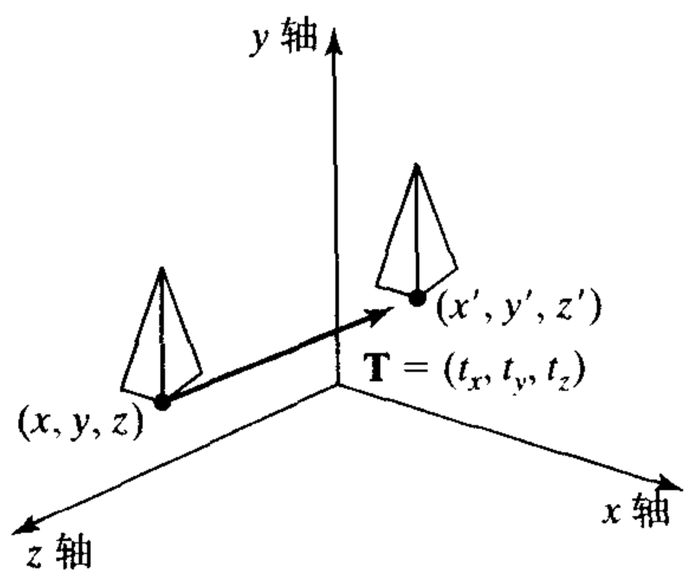


图 5.35　利用变换向量 **T** 对对象进行平移变换

下面的程序段给出了输入一组平移参数后对平移矩阵的构造。为了在这些过程中构造该矩阵，使用与 5.4 节中例子程序相类似的方法。

```
typedef GLfloat Matrix4x4 [4][4];

/* Construct the 4 by 4 identity matrix. */
void matrix4x4SetIdentity (Matrix4x4 matIdent4x4)
{
   GLint row, col;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    for (row = 0; row < 4; row++)
       for (col = 0; col < 4 ; col++)
          matIdent4x4 [row][col] = (row == col);
}

void translate3D (GLfloat tx, GLfloat ty, GLfloat tz)
{
   Matrix4x4 matTransl3D;

   /*  Initialize translation matrix to identity.  */
   matrix4x4SetIdentity (matTransl3D);

   matTransl3D [0][3] = tx;
   matTransl3D [1][3] = ty;
   matTransl3D [2][3] = tz;
}
```

三维平移变换的逆变换使用二维中同样的方法获得，即取平移距离 t_x、t_y 和 t_z 的负值。这将生成相反的平移，而平移矩阵和其逆矩阵之积是单位矩阵。

5.11 三维旋转

我们可以围绕空间的任意轴旋转一个对象，但绕平行于坐标轴的轴的旋转是最容易处理的。我们同样可以利用围绕坐标轴旋转（结合适当的平移）的复合结果来表示任意的一种旋转。因此，我们先考虑坐标轴旋转的操作，然后讨论其他旋转轴所需的计算。

通常，如果沿着坐标轴的正半轴观察原点，那么绕坐标轴的逆时针旋转为正向旋转（参见图5.36）。这与以前在二维中讨论的旋转是一致的。在二维观察中，xy 平面上的正向旋转方向是绕基准点（平行于 z 坐标轴的轴）进行逆时针旋转。

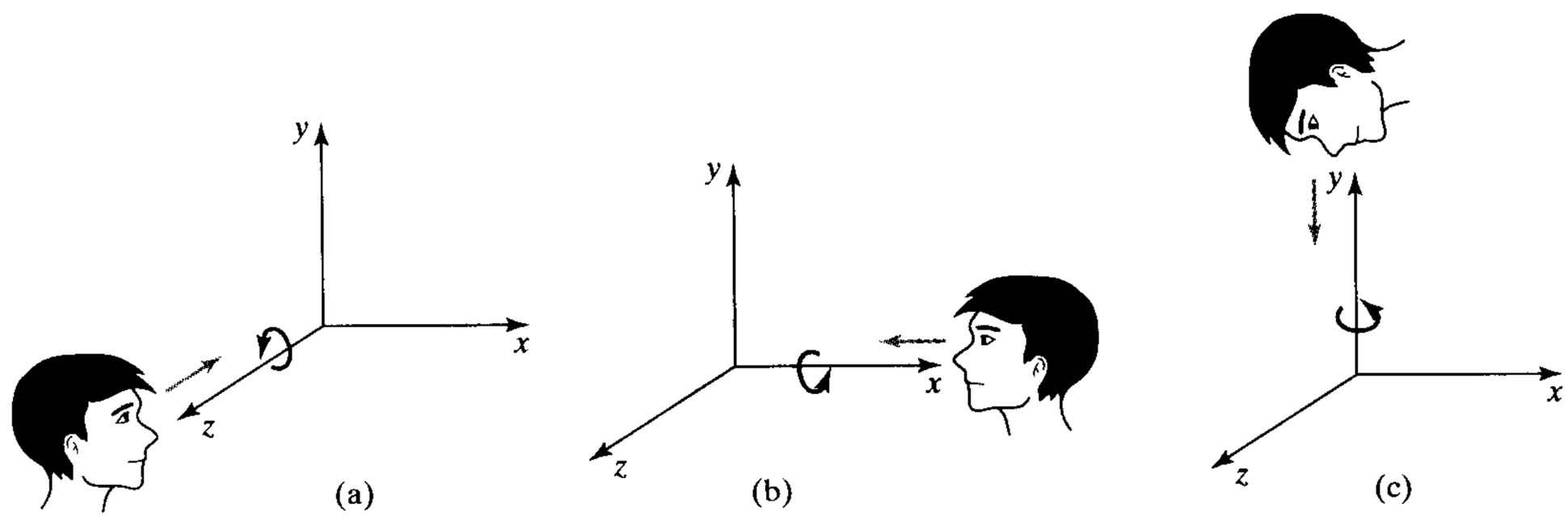


图 5.36 当沿着某坐标轴正半轴观察原点时，绕坐标轴的正向旋转方向是逆时针方向

5.11.1 三维坐标轴旋转

绕 z 轴的二维旋转很容易推广到三维：

$$
\begin{aligned}
x' &= x\cos\theta - y\sin\theta \\
y' &= x\sin\theta + y\cos\theta \\
z' &= z
\end{aligned}
\tag{5.73}
$$

参数 θ 表示指定的绕 z 轴旋转的角度，而 z 坐标值在该变换中不改变。三维 z 轴旋转方程可以用齐次坐标形式表示如下：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5.74}$$

更简洁的形式是

$$\mathbf{P}' = \mathbf{R}_z(\theta) \cdot \mathbf{P} \tag{5.75}$$

图 5.37 给出了一对象绕 z 轴的旋转。

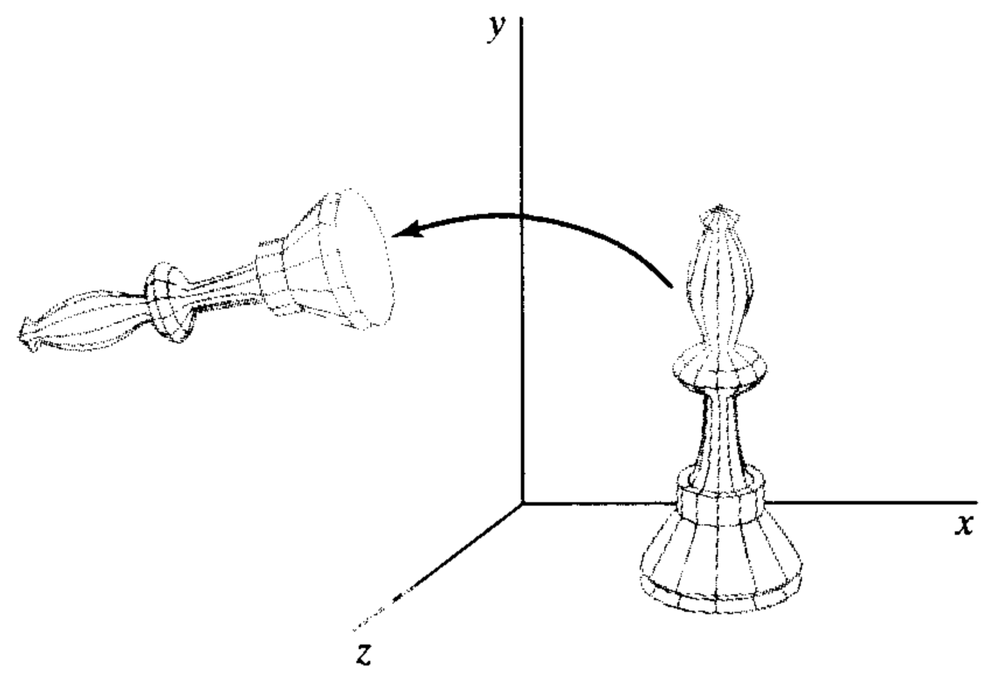


图 5.37　一对象绕 z 轴旋转

绕另外两个坐标轴的旋转变换公式，可以由方程（5.73）中的坐标参数 x、y、z 循环替换而得到：

$$x \to y \to z \to x \tag{5.76}$$

因此，为了得到 x 轴和 y 轴的旋转变换，我们用 y 替代 x、用 z 替代 y、用 x 替代 z，图 5.38 给出了这种循环替换。

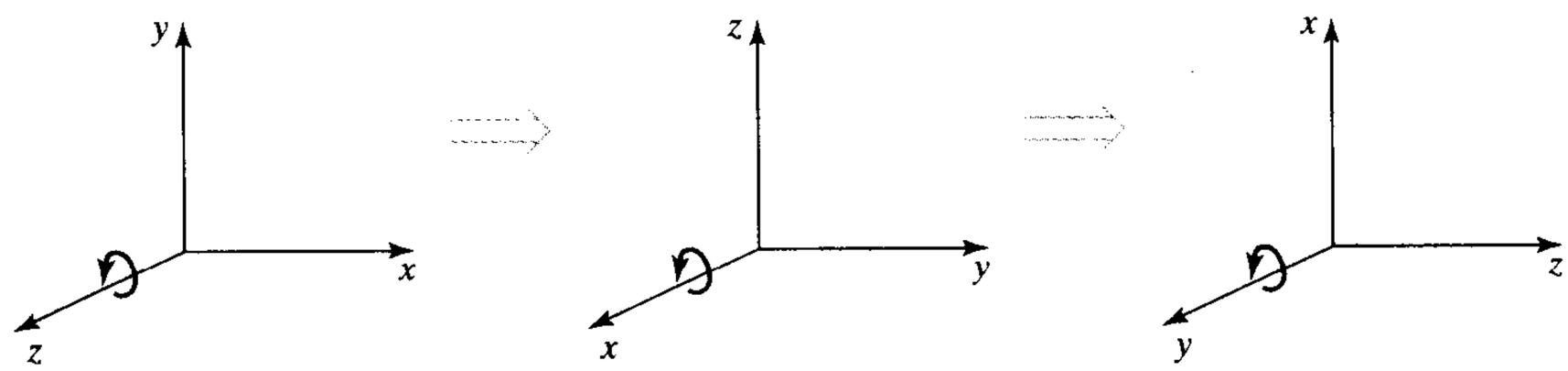


图 5.38　用循环笛卡儿坐标替代生成三组坐标轴旋转公式

在方程（5.73）中利用表达式（5.76）进行替换，可以得到 **x 轴旋转**公式：

$$\begin{aligned} y' &= y\cos\theta - z\sin\theta \\ z' &= y\sin\theta + z\cos\theta \\ x' &= x \end{aligned} \tag{5.77}$$

图 5.39 给出了绕 x 轴旋转一个对象的例子。

对方程（5.77）进行坐标循环替换，可以得到 **y 轴旋转**公式：

$$\begin{aligned} z' &= z\cos\theta - x\sin\theta \\ x' &= z\sin\theta + x\cos\theta \\ y' &= y \end{aligned} \tag{5.78}$$

图 5.40 给出了绕 y 轴旋转一个对象的例子。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

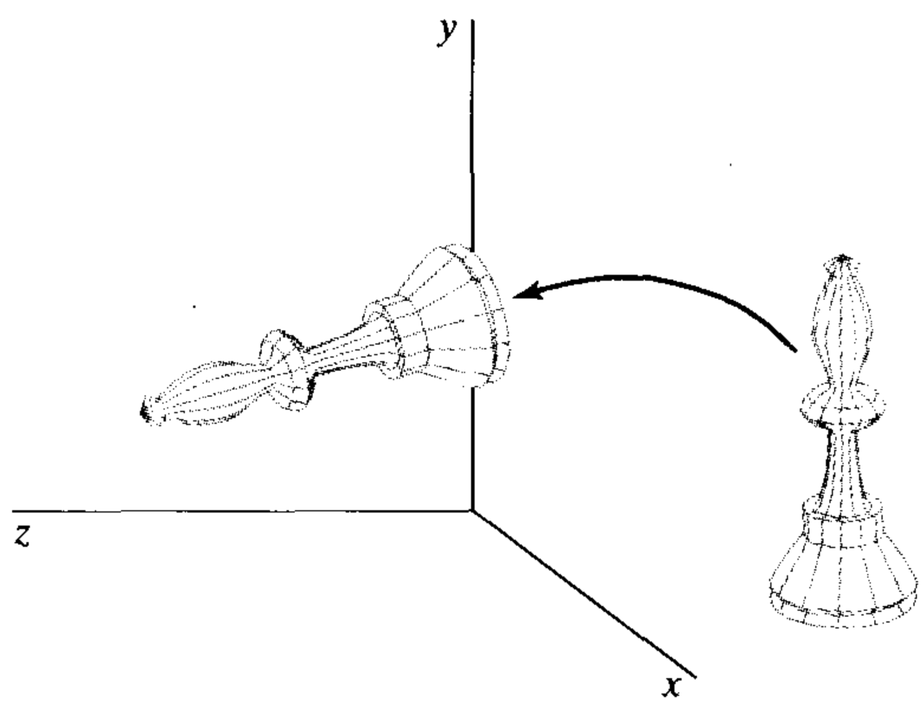


图 5.39 一对象绕 x 轴进行旋转

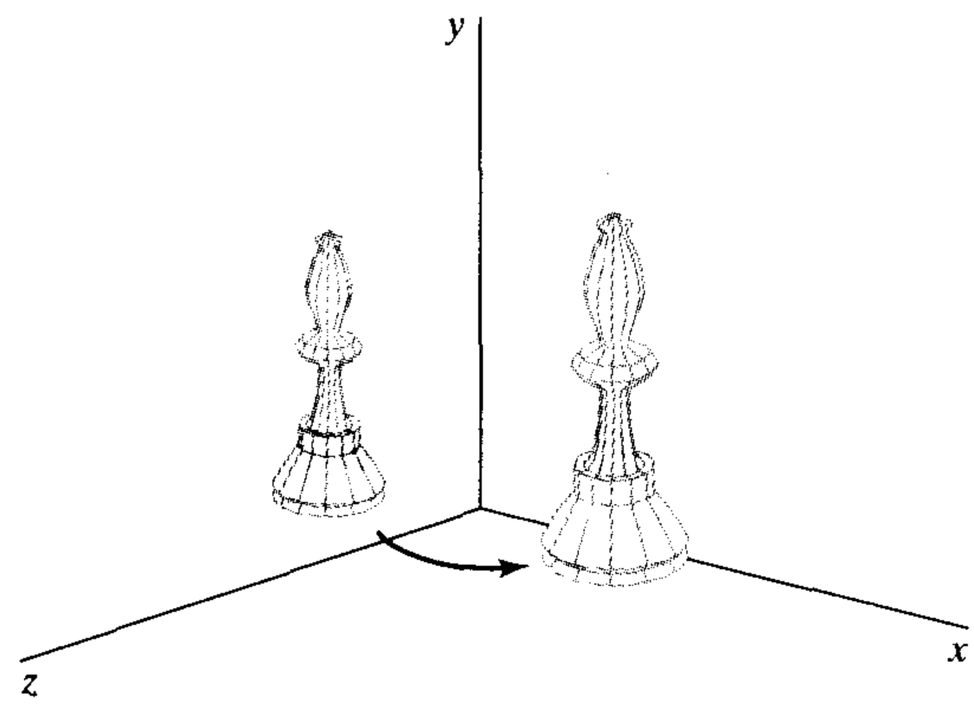


图 5.40 一对象绕 y 轴进行旋转

三维旋转矩阵的逆矩阵可按二维中求逆的方法获得。我们用 $-\theta$ 替代旋转角 θ 可以得到逆旋转矩阵。旋转角取负值将导致对象沿顺时针方向进行旋转，因而当旋转矩阵乘以其逆矩阵时将得到单位矩阵。由于只有正弦函数受到旋转角符号变化的影响，因此逆矩阵也可以通过交换原矩阵中的行和列来得到，即可以利用旋转矩阵 $\mathbf{R}$ 的转置来取代逆矩阵计算（$\mathbf{R}^{-1}=\mathbf{R}^{T}$）。

5.11.2 一般三维旋转

对于绕与坐标轴不重合的轴进行旋转的变换矩阵，可以利用平移与坐标轴旋转的复合而得到。首先将指定旋转轴经移动和旋转变换到坐标轴之一，然后对该坐标轴应用适当的旋转矩阵。最后将旋转轴变回到原来位置。

在某些特殊情况下，例如将对象绕平行于某坐标轴的轴旋转，可以通过下列变换序列来得到所需的旋转矩阵：

1. 平移对象使其旋转轴与平行于该轴的一个坐标轴重合；
2. 对于该轴完成指定的旋转；
3. 平移对象将其旋转轴移回到原来的位置。

此序列的步骤如图 5.41 中所示。在此图中，对象上任意坐标点 $\mathbf{P}$ 经过一系列变换而成为

$$\mathbf{P}' = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \cdot \mathbf{P} \tag{5.79}$$

其中，旋转变换的复合矩阵是

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \tag{5.80}$$

其形式与绕平行于 z 轴的任意轴旋转的二维变换序列（非原点的任意点）的形式相类似。

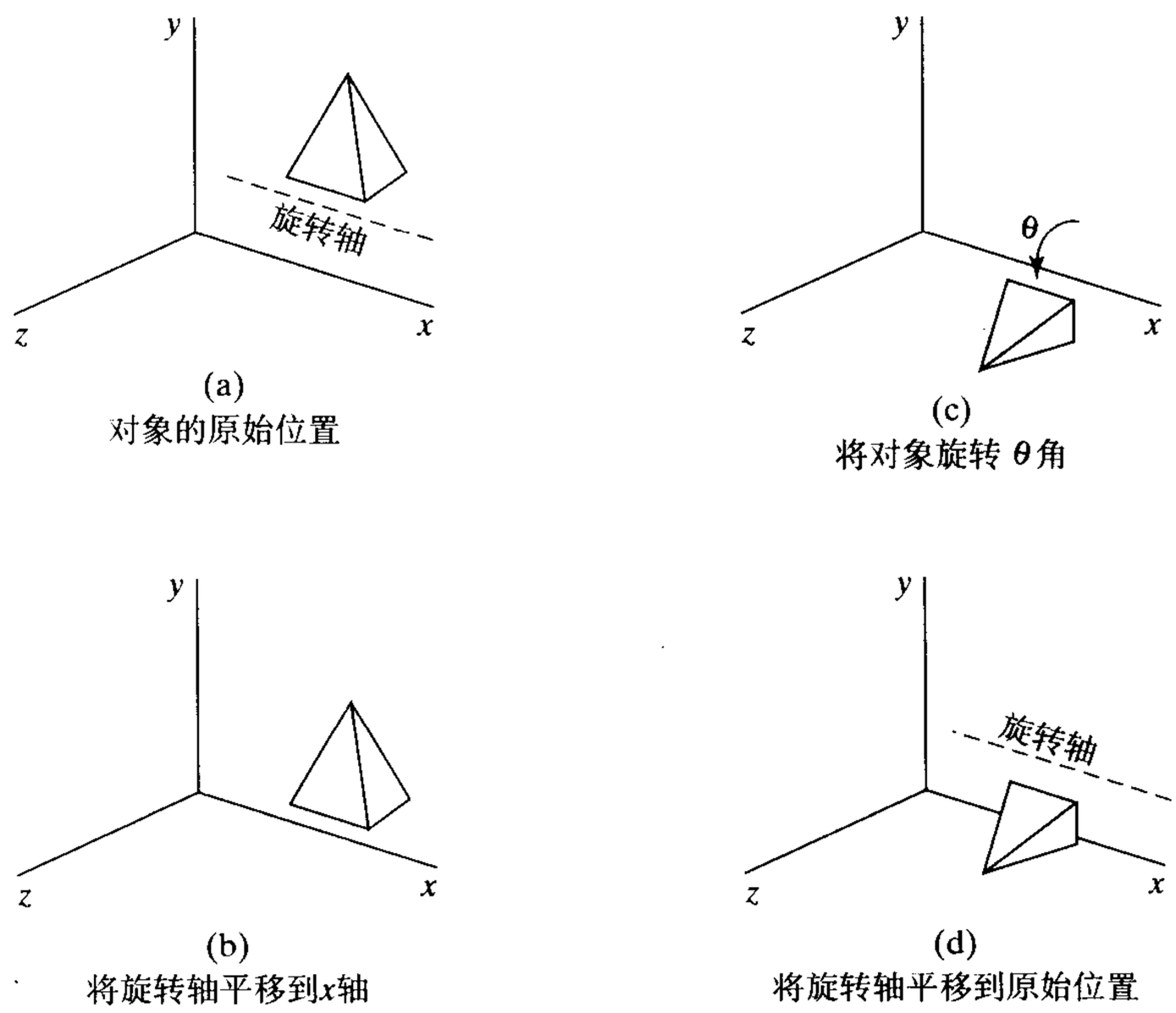
如果对象绕与每个坐标轴均不平行的轴旋转，则需要进行额外的变换。此时，仍需要进行旋转以使旋转轴与某一选定的坐标轴对齐，然后将此轴变回到原始位置。若给定旋转轴和旋转角，我们可以按照 5 个步骤来完成所需的旋转：

1. 平移对象，使得旋转轴通过坐标原点；
2. 旋转对象使得旋转轴与某一坐标轴重合；

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

3. 绕坐标轴完成指定的旋转；
4. 利用逆旋转使旋转轴回到其原始方向；
5. 利用逆平移使旋转轴回到其原始位置。



(a) 对象的原始位置

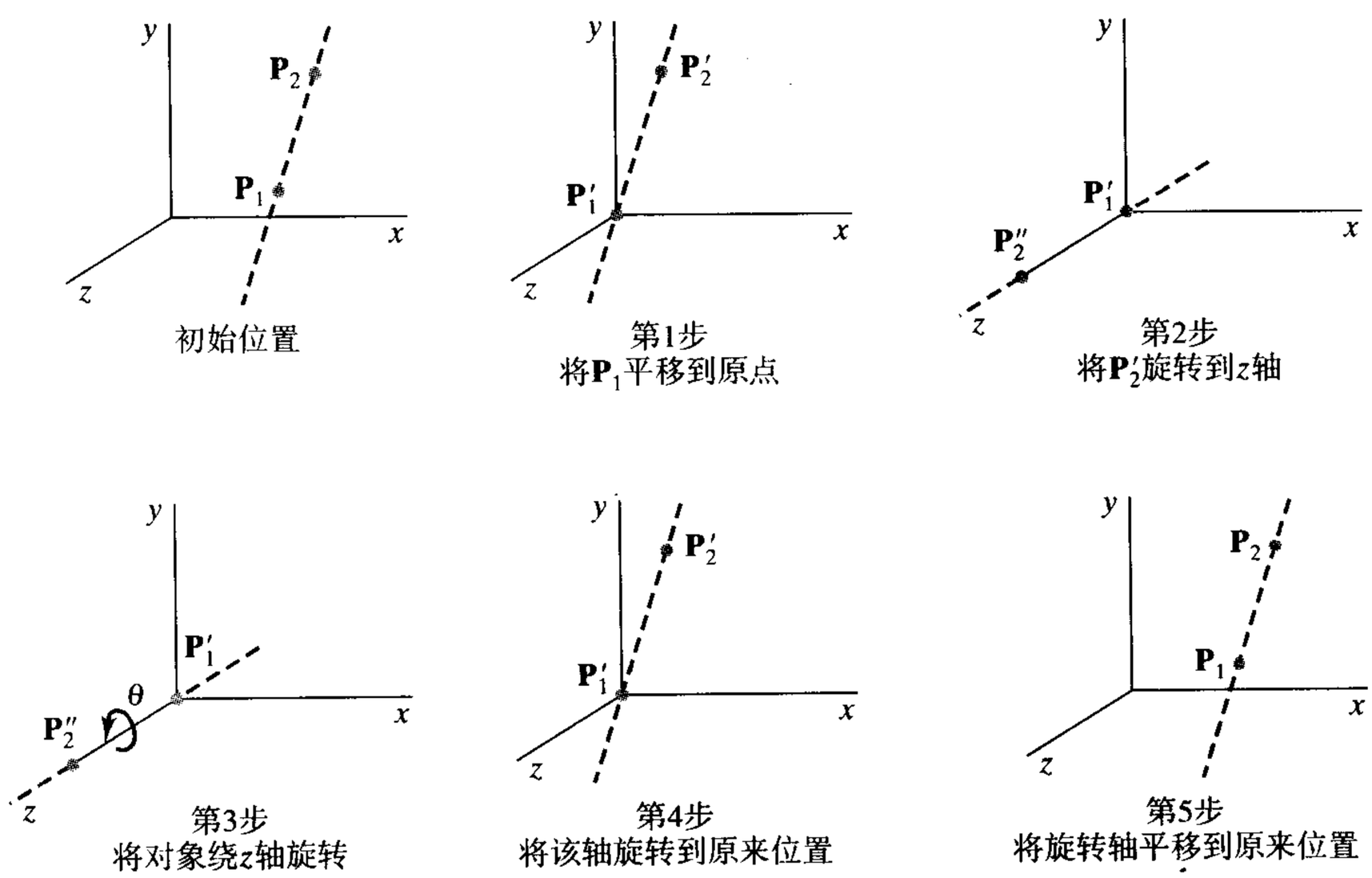
(b) 将旋转轴平移到x轴

(c) 将对象旋转 θ 角

(d) 将旋转轴平移到原始位置

图 5.41　将对象绕着与 x 轴平行的轴线进行旋转的变换序列

我们可以将旋转轴变换到三个坐标轴的任意一个。z 轴是比较方便的选择，而下面讨论使用 z 轴旋转矩阵的变换序列（参见图 5.42）。



初始位置

第1步 将$\mathbf{P}_1$平移到原点

第2步 将$\mathbf{P}_2'$旋转到z轴

第3步 将对象绕z轴旋转

第4步 将该轴旋转到原来位置

第5步 将旋转轴平移到原来位置

图 5.42　绕任意轴方向旋转时，通过将旋转轴投影至 z 轴方向求得复合变换矩阵的 5 个步骤

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

任意旋转轴可以由两个坐标点确定，如图 5.43 所示，或通过一个坐标点和旋转轴与两个坐标轴间的方向角（或方向余弦）来确定。假设由两点确定旋转轴，如果沿着从 $\mathbf{P}_2$ 到 $\mathbf{P}_1$ 的轴进行观察，并且旋转的方向为逆时针方向，则轴向量利用两点可以定义为

$$\begin{aligned}\mathbf{V} &= \mathbf{P}_2 - \mathbf{P}_1 \\ &= (x_2 - x_1, y_2 - y_1, z_2 - z_1)\end{aligned} \tag{5.81}$$

沿旋转轴的单位向量 $\mathbf{u}$ 则定义为

$$\mathbf{u} = \frac{\mathbf{V}}{|\mathbf{V}|} = (a, b, c) \tag{5.82}$$

其中，分量 a、b、c 是旋转轴的方向余弦：

$$a = \frac{x_2 - x_1}{|\mathbf{V}|}, \qquad b = \frac{y_2 - y_1}{|\mathbf{V}|}, \qquad c = \frac{z_2 - z_1}{|\mathbf{V}|} \tag{5.83}$$

若以相反方向旋转（当观察方向从 $\mathbf{P}_2$ 到 $\mathbf{P}_1$ 时，旋转方向为顺时针），则需要将轴向量 $\mathbf{V}$ 和单位向量 $\mathbf{u}$ 取反而从 $\mathbf{P}_2$ 指向 $\mathbf{P}_1$。

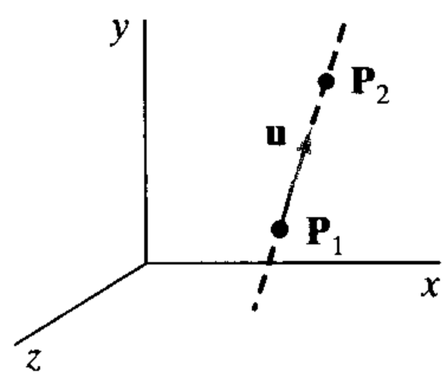


图 5.43　旋转轴（虚线）根据点 $\mathbf{P}_1$ 和 $\mathbf{P}_2$ 进行定义。单位轴向量 $\mathbf{u}$ 的方向由给定的旋转方向所确定

指定旋转所需的变换序列的第一步是建立使旋转轴通过原点的平移变换矩阵。由于需要一个从 $\mathbf{P}_2$ 往 $\mathbf{P}_1$ 看时的逆时针旋转（参见图 5.43），我们将 $\mathbf{P}_1$ 点移到原点。（如果旋转方向指定为相反方向，则将 $\mathbf{P}_2$ 移到原点。）该变换矩阵是

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.84}$$

这实现了对旋转轴和对象的重定位，如图 5.44 所示。

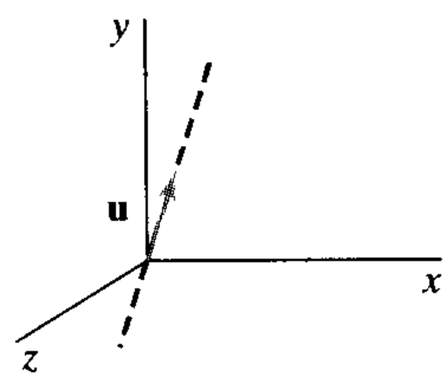


图 5.44　平移旋转轴到坐标原点

现在需要将旋转轴与 z 轴坐标轴重合的变换。可以利用二次坐标轴旋转来完成这一对齐。可以有多种方法来实现这两个步骤。在此例中，首先绕 x 轴旋转，然后绕 y 轴旋转。绕 x 轴旋转将向量 $\mathbf{u}$ 变换到 xz 平面上，绕 y 轴旋转将 $\mathbf{u}$ 变到 z 轴上。图 5.45 给出了某一方向的 $\mathbf{u}$ 向量的两次旋转。

由于旋转计算包括了正弦和余弦函数，可以利用标准向量运算（参见附录 A）来得到两个旋转矩阵的元素。向量的点积运算可以确定余弦项，向量的叉积运算可以获得正弦项。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

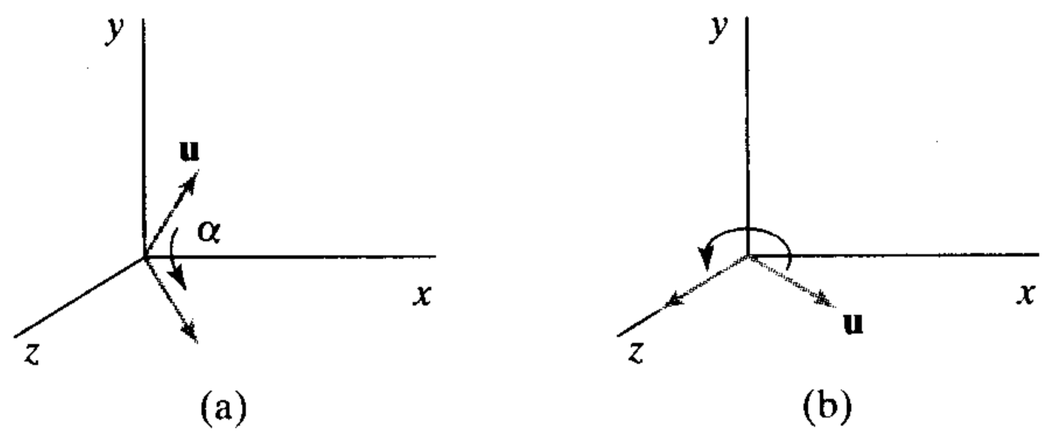


图 5.45　(a) 单位向量 **u** 绕 x 轴旋转到 xz 平面上；(b) 然后绕 y 轴旋转到 z 轴上

计算出使 **u** 变换到 xz 平面上的旋转角的正弦和余弦值，可以建立绕 x 轴旋转的变换矩阵。该旋转角是 **u** 在 yz 平面上的投影与 z 轴正向的夹角（参见图 5.46）。如果设 **u** 在 yz 平面上的投影为向量 $\mathbf{u}' = (0, b, c)$，则旋转角 α 的余弦可以由 $\mathbf{u}'$ 和 z 轴上单位向量 $\mathbf{u}_z$ 的点积来得到：

$$\cos\alpha = \frac{\mathbf{u}' \cdot \mathbf{u}_z}{|\mathbf{u}'|\,|\mathbf{u}_z|} = \frac{c}{d} \tag{5.85}$$

其中 d 是 $\mathbf{u}'$ 的模：

$$d = \sqrt{b^2 + c^2} \tag{5.86}$$

类似地，可以利用 $\mathbf{u}'$ 和 $\mathbf{u}_z$ 的叉积来给出 α 的正弦。与坐标无关的叉积形式是

$$\mathbf{u}' \times \mathbf{u}_z = \mathbf{u}_x\,|\mathbf{u}'|\,|\mathbf{u}_z|\sin\alpha \tag{5.87}$$

而且叉积的笛卡儿形式为

$$\mathbf{u}' \times \mathbf{u}_z = \mathbf{u}_x \cdot b \tag{5.88}$$

等式（5.87）和等式（5.88）的右边相等，并注意 $|\mathbf{u}_z| = 1$ 及 $|\mathbf{u}'| = d$，则有

$$d\sin\alpha = b$$

或

$$\sin\alpha = \frac{b}{d} \tag{5.89}$$

由于已使用向量 **u** 的分量来得出 $\cos\alpha$ 和 $\sin\alpha$ 的值，因此可以建立绕 x 轴将 **u** 旋转到 xz 平面的旋转矩阵：

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & -\frac{b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.90}$$

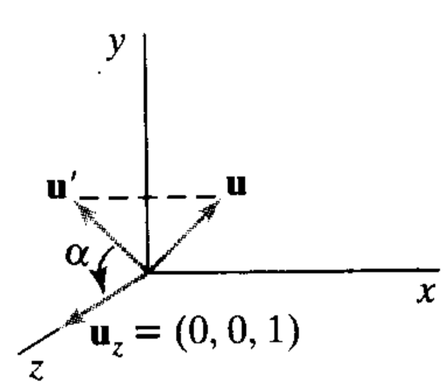


图 5.46　**u** 绕 x 轴旋转到 xz 平面上，由 $\mathbf{u}'$（**u** 在 yz 平面上投影）旋转 α 角到 z 轴上来实现

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

下一步需要确定变换矩阵，此矩阵将 xz 平面上的单位向量绕 y 轴逆时针旋转到 z 轴的正方向。绕 x 轴旋转后，xz 平面上单位向量的方向如图 5.47 所示。该向量记为 $\mathbf{u}''$，因为绕 x 轴的旋转保持 x 分量不变，故其 x 分量值为 a；因为向量 $\mathbf{u}'$ 旋转到 z 轴上，故其 z 分量为 d（$\mathbf{u}'$ 的模）；因为 $\mathbf{u}''$ 在 xz 平面上，故 $\mathbf{u}''$ 的 y 分量为 0。同样，由单位向量 $\mathbf{u}''$ 和 $\mathbf{u}_z$ 的点积可确定旋转角 β 的余弦：

$$\cos\beta = \frac{\mathbf{u}'' \cdot \mathbf{u}_z}{|\mathbf{u}''|\,|\mathbf{u}_z|} = d \tag{5.91}$$

因为 $|\mathbf{u}_z| = |\mathbf{u}''| = 1$。比较叉积与坐标无关的形式：

$$\mathbf{u}'' \times \mathbf{u}_z = \mathbf{u}_y\,|\mathbf{u}''|\,|\mathbf{u}_z|\sin\beta \tag{5.92}$$

和笛卡儿形式：

$$\mathbf{u}'' \times \mathbf{u}_z = \mathbf{u}_y \cdot (-a) \tag{5.93}$$

可以发现：

$$\sin\beta = -a \tag{5.94}$$

这样，$\mathbf{u}''$ 绕 y 轴的旋转变换矩阵为

$$\mathbf{R}_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.95}$$

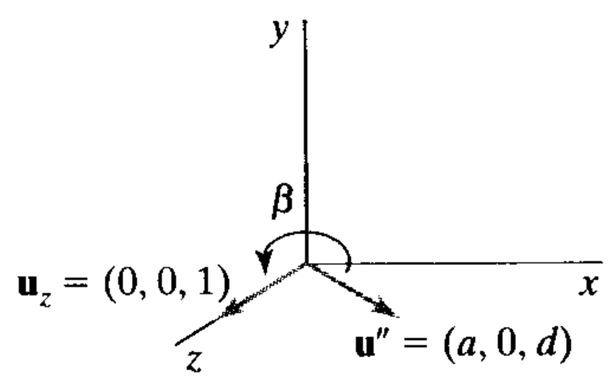


图 5.47 单位向量 $\mathbf{u}''$（$\mathbf{u}$ 向量旋转到 xz 平面上）绕 y 轴的旋转。正向旋转 β 角使 $\mathbf{u}''$ 与 $\mathbf{u}_z$ 向量对齐

利用等式（5.84）、等式（5.90）和等式（5.95）的变换矩阵，可以将旋转轴对齐到 z 轴的正方向。给定的旋转角 θ 已经可以用于关于 z 轴的旋转：

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.96}$$

为了实现对于给定轴的旋转，还需要将旋转轴变回到原来位置。这可以通过使用等式（5.84）、等式（5.90）和等式（5.95）中变换的逆矩阵来完成。这样，对于任意轴的旋转可以表示为上述7个变换的复合形式：

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{R}_x^{-1}(\alpha) \cdot \mathbf{R}_y^{-1}(\beta) \cdot \mathbf{R}_z(\theta) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{T} \tag{5.97}$$

得到复合旋转矩阵 $\mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha)$ 的更快但不太直观的方法是，利用任意三维旋转的复合矩阵形式：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{R}=\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.98}$$

该矩阵上面的 3 × 3 子矩阵是正交的。这意味着该子矩阵的行（或列）形成了由矩阵 **R** 分别绕 x 轴、y 轴和 z 轴旋转的正交单位向量组：

$$\mathbf{R}\cdot\begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \\ 1 \end{bmatrix}=\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix},\qquad \mathbf{R}\cdot\begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \\ 1 \end{bmatrix}=\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix},\qquad \mathbf{R}\cdot\begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \\ 1 \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{5.99}$$

因此，可以考虑由旋转轴定义的局部坐标系，并简单地建立以局部单位坐标向量为列的矩阵。假设旋转轴不平行于任何坐标轴，则可以建立下列局部单位向量组（参见图 5.48）：

$$\begin{aligned} \mathbf{u}'_z &= \mathbf{u} \\ \mathbf{u}'_y &= \frac{\mathbf{u}\times\mathbf{u}_x}{|\mathbf{u}\times\mathbf{u}_x|} \\ \mathbf{u}'_x &= \mathbf{u}'_y\times\mathbf{u}'_z \end{aligned} \tag{5.100}$$

如果将旋转轴的局部单位向量元素表示为

$$\begin{aligned} \mathbf{u}'_x &= (u'_{x1}, u'_{x2}, u'_{x3}) \\ \mathbf{u}'_y &= (u'_{y1}, u'_{y2}, u'_{y3}) \\ \mathbf{u}'_z &= (u'_{z1}, u'_{z2}, u'_{z3}) \end{aligned} \tag{5.101}$$

则所需的复合矩阵等于 $\mathbf{R}_y(\beta)\cdot\mathbf{R}_x(\alpha)$ 之积，即

$$\mathbf{R}=\begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.102}$$

这一矩阵分别把单位向量 $\mathbf{u}'_x$、$\mathbf{u}'_y$ 和 $\mathbf{u}'_z$ 变到 x、y 和 z 轴上。因为 $\mathbf{u}'_z=\mathbf{u}$，所以旋转轴与 z 轴对齐。

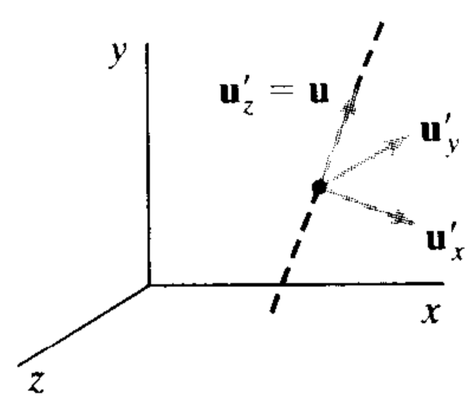


图 5.48　由单位向量 **u** 定义的旋转轴的局部坐标系

5.11.3　三维旋转的四元数方法

将四元数表示（参见附录 A）用于旋转变换是获得对于给定轴旋转的更有效方法。作为二维复数扩充的四元数在许多计算机图形程序都很有用，包括生成分形。它们比 4 × 4 矩阵需要更少的存储空间，并且更容易写出变换序列的四元数过程。这对于需要复杂运动序列和给定对象两个位置的运动插值尤其重要。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

描述四元数的一个方法是把它看做是一个有序对，由一个标量部（scalar part）和一个向量部（vector part）组成：

$$q = (s, \mathbf{v})$$

也可以把四元数看成为高阶复数，带有一个实部（标量部分）和三个复部（向量 **v** 的元素）。绕经过原点的任意轴旋转，可以通过建立有下列标量和向量部的四元数来完成：

$$s = \cos\frac{\theta}{2}, \qquad \mathbf{v} = \mathbf{u}\sin\frac{\theta}{2} \tag{5.103}$$

其中，**u** 是沿所选旋转轴的单位向量，θ 是绕此轴的指定旋转角（参见图 5.49）。任意点 **P** 绕该四元数旋转，可以用四元数符号表示为

$$\mathbf{P} = (0, \mathbf{p})$$

这里，点的坐标为向量部 $\mathbf{p} = (x, y, z)$。然后点的旋转由四元数运算来实现：

$$\mathbf{P}' = q\mathbf{P}q^{-1} \tag{5.104}$$

其中 $q^{-1} = (s, -\mathbf{v})$ 是等式（5.103）中所给的标量和向量部组成的单位四元数 q 的逆变换。这一变换产生了下面的新四元数：

$$\mathbf{P}' = (0, \mathbf{p}') \tag{5.105}$$

该有序对的第二项是旋转的点位置 **p′**，它由点积和叉积表示为

$$\mathbf{p}' = s^2\mathbf{p} + \mathbf{v}(\mathbf{p}\cdot\mathbf{v}) + 2s(\mathbf{v}\times\mathbf{p}) + \mathbf{v}\times(\mathbf{v}\times\mathbf{p}) \tag{5.106}$$

参数 s 和 **v** 由等式（5.103）给定其值。很多计算机图形系统利用向量计算中的有效硬件实现来执行快速三维对象旋转。

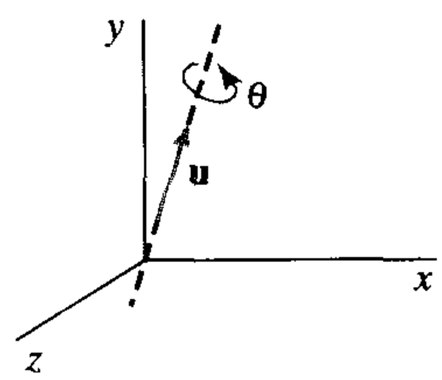


图 5.49 绕指定轴旋转的单位四元数参数 θ 或 **u**

变换式（5.104）等价于绕经过原点的轴旋转。这与等式（5.97）中将旋转轴与 z 轴对齐，然后绕 z 轴旋转，再将旋转轴变回到原来位置的旋转变换序列的作用一样。

运用附录 A 中的四元数乘法定义，可以计算等式（5.106）中的项。指定 q 的向量部分量为 $\mathbf{v} = (a, b, c)$，从而获得 3 × 3 的复合旋转矩阵 $\mathbf{R}_x^{-1}(\alpha)\cdot\mathbf{R}_y^{-1}(\beta)\cdot\mathbf{R}_z(\theta)\cdot\mathbf{R}_y(\beta)\cdot\mathbf{R}_x(\alpha)$ 如下：

$$\mathbf{M}_R(\theta) = \begin{bmatrix} 1-2b^2-2c^2 & 2ab-2sc & 2ac+2sb \\ 2ab+2sc & 1-2a^2-2c^2 & 2bc-2sa \\ 2ac-2sb & 2bc+2sa & 1-2a^2-2b^2 \end{bmatrix} \tag{5.107}$$

通过用显式值取代参数 a、b、c 和 s 并使用下列三角恒等式简化有关项，可以大大减少该矩阵中的计算。

$$\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2} = 1 - 2\sin^2\frac{\theta}{2} = \cos\theta, \qquad 2\cos\frac{\theta}{2}\sin\frac{\theta}{2} = \sin\theta$$

这样，我们可重写矩阵（5.107）为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{M}_R(\theta) = \begin{bmatrix} u_x^2(1-\cos\theta)+\cos\theta & u_xu_y(1-\cos\theta)-u_z\sin\theta & u_xu_z(1-\cos\theta)+u_y\sin\theta \\ u_yu_x(1-\cos\theta)+u_z\sin\theta & u_y^2(1-\cos\theta)+\cos\theta & u_yu_z(1-\cos\theta)-u_x\sin\theta \\ u_zu_x(1-\cos\theta)-u_y\sin\theta & u_zu_y(1-\cos\theta)+u_x\sin\theta & u_z^2(1-\cos\theta)+\cos\theta \end{bmatrix} \quad (5.108)$$

其中 u_x、u_y 和 u_z 是单位轴向量 **u** 的分量。

为了得到完整的绕任意指定旋转轴旋转的变换序列，需要包括将旋转轴移至坐标原点并回到其初始位置的变换。因此，与等式（5.97）对应的完整四元数旋转表示是

$$\mathbf{R}(\theta) = \mathbf{T}^{-1} \cdot \mathbf{M}_R \cdot \mathbf{T} \quad (5.109)$$

作为一个例子，可以通过将向量 **u** 设为单位 z 轴向量(0, 0, 1)来完成绕 z 轴的旋转。将该向量的分量代入矩阵（5.108）可得变换等式（5.74）中 z 轴旋转矩阵 $\mathbf{R}_z(\theta)$的 3 × 3 版本。同样，单位四元数旋转值代入等式（5.104），就产生了如方程（5.73）所示的旋转坐标值。

在下列程序中，给出了可用来构造三维旋转矩阵的过程示例。等式（5.109）中的四元数表示用来设定通用三维旋转的矩阵元素。

```
class wcPt3D {
   public:
      GLfloat x, y, z;
};
typedef float Matrix4x4 [4][4];

Matrix4x4 matRot;

/* Construct the 4 by 4 identity matrix. */
void matrix4x4SetIdentity (Matrix4x4 matIdent4x4)
{
   GLint row, col;

   for (row = 0; row < 4; row++)
      for (col = 0; col < 4 ; col++)
         matIdent4x4 [row][col] = (row == col);
}

/* Premultiply matrix m1 times matrix m2, store result in m2. */
void matrix4x4PreMultiply (Matrix4x4 m1, Matrix4x4 m2)
{
   GLint row, col;
   Matrix4x4 matTemp;

   for (row = 0; row < 4; row++)
      for (col = 0; col < 4 ; col++)
         matTemp [row][col] = m1 [row][0] * m2 [0][col] + m1 [row][1] *
                              m2 [1][col] + m1 [row][2] * m2 [2][col] +
                              m1 [row][3] * m2 [3][col];
   for (row = 0; row < 4; row++)
      for (col = 0; col < 4; col++)
         m2 [row][col] = matTemp [row][col];
}

void translate3D (GLfloat tx, GLfloat ty, GLfloat tz)
{
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    Matrix4x4 matTransl3D;

    /*  Initialize translation matrix to identity.  */
    matrix4x4SetIdentity (matTransl3D);

    matTransl3D [0][3] = tx;
    matTransl3D [1][3] = ty;
    matTransl3D [2][3] = tz;

    /*  Concatenate translation matrix with matRot.  */
    matrix4x4PreMultiply (matTransl3D, matRot);
}

void rotate3D (wcPt3D p1, wcPt3D p2, GLfloat radianAngle)
{
    Matrix4x4 matQuaternionRot;

    GLfloat axisVectLength = sqrt ((p2.x - p1.x) * (p2.x - p1.x) +
                          (p2.y - p1.y) * (p2.y - p1.y) +
                          (p2.z - p1.z) * (p2.z - p1.z));
    GLfloat cosA = cos (radianAngle);
    GLfloat oneC = 1 - cosA;
    GLfloat sinA = sin (radianAngle);
    GLfloat ux = (p2.x - p1.x) / axisVectLength;
    GLfloat uy = (p2.y - p1.y) / axisVectLength;
    GLfloat uz = (p2.z - p1.z) / axisVectLength;

    /*  Set up translation matrix for moving p1 to origin.  */
    translate3D (-p1.x, -p1.y, -p1.z);

    /*  Initialize matQuaternionRot to identity matrix.  */
    matrix4x4SetIdentity (matQuaternionRot);

    matQuaternionRot [0][0] = ux*ux*oneC + cosA;
    matQuaternionRot [0][1] = ux*uy*oneC - uz*sinA;
    matQuaternionRot [0][2] = ux*uz*oneC + uy*sinA;
    matQuaternionRot [1][0] = uy*ux*oneC + uz*sinA;
    matQuaternionRot [1][1] = uy*uy*oneC + cosA;
    matQuaternionRot [1][2] = uy*uz*oneC - ux*sinA;
    matQuaternionRot [2][0] = uz*ux*oneC - uy*sinA;
    matQuaternionRot [2][1] = uz*uy*oneC + ux*sinA;
    matQuaternionRot [2][2] = uz*uz*oneC + cosA;

    /*  Combine matQuaternionRot with translation matrix.  */
    matrix4x4PreMultiply (matQuaternionRot, matRot);

    /*  Set up inverse matTransl3D and concatenate with
     *  product of previous two matrices.
     */
    translate3D (p1.x, p1.y, p1.z);
}

void displayFcn (void)
{
    /*  Input rotation parameters.  */
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    /*  Initialize matRot to identity matrix:  */
    matrix4x4SetIdentity (matRot);

    /*  Pass rotation parameters to procedure rotate3D.  */

    /*  Display rotated object.  */
}
```

5.12 三维缩放

点 $\mathbf{P}=(x, y, z)$ 相对于坐标原点的三维缩放是二维缩放的简单扩充。只要在变换矩阵中引入 z 坐标缩放参数：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{5.110}$$

一个点的三维缩放变换矩阵可以表示为

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P} \tag{5.111}$$

其中，缩放参数 s_x、s_y 和 s_z 为指定的任意正值。相对于原点的比例缩放变换的显式表示为

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y, \qquad z' = z \cdot s_z \tag{5.112}$$

利用变换式（5.110）对一个对象的缩放使得对象大小和相对于坐标原点的对象位置发生变化。大于1的参数值将该点沿原点到该点坐标方向而向远处移动。类似地，小于1的参数值将该点沿其到原点的方向移近原点。同样，如果缩放变换参数不相同，则对象的相关尺寸也发生变化。可以使用统一的缩放参数（$s_x = s_y = s_z$）来保持对象的原有形状。使用相同的缩放参数（置为2）来缩放一个对象的结果如图5.50所示。

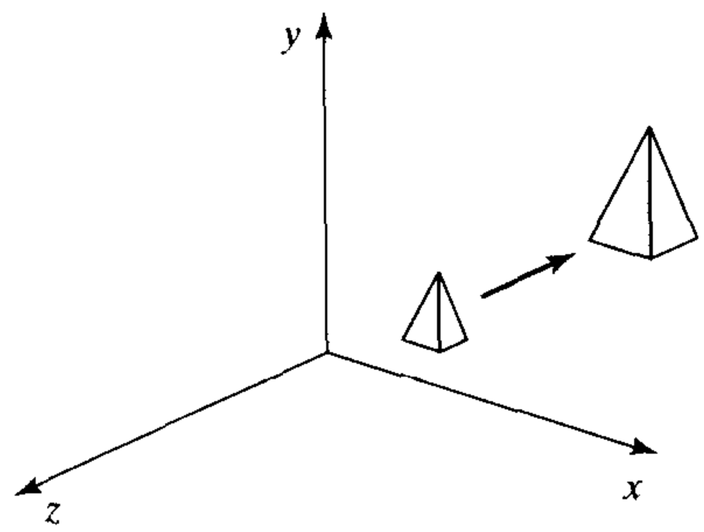


图5.50 利用变换式（5.110）放大对象两倍也会使对象远离原点

由于某些图形软件仅提供相对于坐标原点的缩放子程序，我们可以使用下列变换序列进行相对于任意给定点 (x_f, y_f, z_f) 的缩放变换：

1. 平移给定点到原点；
2. 使用等式（5.110），相对于坐标原点缩放对象；
3. 平移给定点回到原始位置。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图5.51中描述了此变换序列。有关任意点的缩放变换矩阵表达式，可以利用平移-缩放-平移变换组合表示为

$$\mathbf{T}(x_f, y_f, z_f)\cdot\mathbf{S}(s_x, s_y, s_z)\cdot\mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.113}$$

通过使用平移-缩放-平移序列或直接相对于固定点的方法可建立编程过程。下面的程序给出了使用等式（5.113）中的计算来直接构造相对于指定点的三维缩放矩阵的例子。

```
class wcPt3D
{
   private:
      GLfloat x, y, z;

   public:
   /*  Default Constructor:
    *  Initialize position as (0.0, 0.0, 0.0).
    */
   wcPt3D ( ) {
      x = y = z = 0.0;
   }

   setCoords (GLfloat xCoord, GLfloat yCoord, GLfloat zCoord) {
      x = xCoord;
      y = yCoord;
      z = zCoord;
   }

   GLfloat getx ( ) const {
      return x;
   }

   GLfloat gety ( ) const {
      return y;
   }

   GLfloat getz ( ) const {
      return z;
   }
};

typedef float Matrix4x4 [4][4];

void scale3D (GLfloat sx, GLfloat sy, GLfloat sz, wcPt3D fixedPt)
{
   Matrix4x4 matScale3D;

   /*  Initialize scaling matrix to identity.  */
   matrix4x4SetIdentity (matScale3D);

   matScale3D [0][0] = sx;
   matScale3D [0][3] = (1 - sx) * fixedPt.getx ( );
   matScale3D [1][1] = sy;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    matScale3D [1][3] = (1 - sy) * fixedPt.gety ( );
    matScale3D [2][2] = sz;
    matScale3D [2][3] = (1 - sz) * fixedPt.getz ( );
}
```

反过来，利用参数(s_x, s_y, s_z)的倒数来代替缩放参数，可以形成等式（5.110）或等式（5.113）的逆缩放矩阵。但如果其中一个缩放参数赋值为0，则其逆变换矩阵无定义。逆矩阵产生相反的缩放变换，因此缩放变换和其逆变换的复合生成单位矩阵。

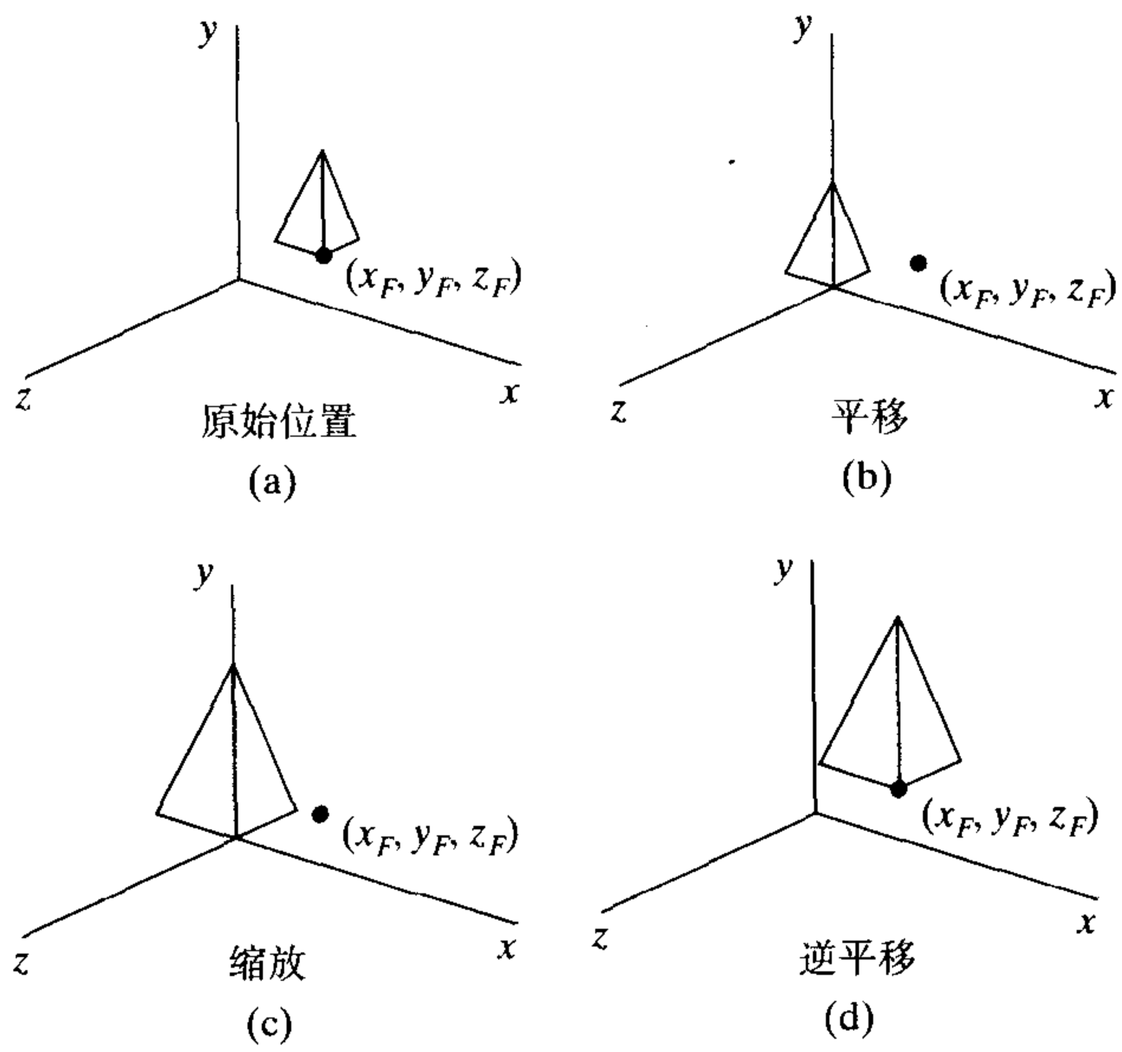


图 5.51 相对于指定点缩放，利用等式（5.110）缩放一对象的变换序列

5.13 三维复合变换

类似于二维变换，可以将变换序列中各次运算的矩阵相乘以形成三维复合变换。5.4 节讨论的任意的三维变换序列，如非坐标轴方向的缩放，可以在三维空间中实现。

依赖于指定的矩阵次序，从右到左或从左到右进行矩阵合并来实现变换序列。当然，其中最右边的矩阵是第一个作用于对象的变换，最左边的矩阵是最后一个变换。由于坐标位置用四元素列向量表示，4 × 4 矩阵必须放在其左边与之相乘，因而需使用上述矩阵的相乘次序。

下列程序提供了构造三维复合变换矩阵的例子。一系列基本的三维几何变换按选定次序组合成一个复合矩阵，该矩阵的初始化形式是单位矩阵。在该例中，先旋转，然后缩放，最后平移。我们选择从左往右的复合矩阵进行计算，从而按应用的顺序调用变换。这样，在构造完每一矩阵之后，将其置于当前复合矩阵左边进行合并以生成更新的积矩阵。

```
class wcPt3D {
   public:
      GLfloat x, y, z;
};
typedef GLfloat Matrix4x4 [4][4];

Matrix4x4 matComposite;

/* Construct the 4 by 4 identity matrix. */
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void matrix4x4SetIdentity (Matrix4x4 matIdent4x4)
{
   GLint row, col;

   for (row = 0; row < 4; row++)
      for (col = 0; col < 4 ; col++)
         matIdent4x4 [row][col] = (row == col);
}

/* Premultiply matrix m1 times matrix m2, store result in m2. */
void matrix4x4PreMultiply (Matrix4x4 m1, Matrix4x4 m2)

{
   GLint row, col;
   Matrix4x4 matTemp;

   for (row = 0; row < 4; row++)
      for (col = 0; col < 4 ; col++)
         matTemp [row][col] = m1 [row][0] * m2 [0][col] + m1 [row][1] *
                              m2 [1][col] + m1 [row][2] * m2 [2][col] +
                              m1 [row][3] * m2 [3][col];
   for (row = 0; row < 4; row++)
      for (col = 0; col < 4; col++)
         m2 [row][col] = matTemp [row][col];
}

/*  Procedure for generating 3D translation matrix.  */
void translate3D (GLfloat tx, GLfloat ty, GLfloat tz)
{
   Matrix4x4 matTransl3D;

   /*  Initialize translation matrix to identity.  */
   matrix4x4SetIdentity (matTransl3D);

   matTransl3D [0][3] = tx;
   matTransl3D [1][3] = ty;
   matTransl3D [2][3] = tz;

   /*  Concatenate matTransl3D with composite matrix.  */
   matrix4x4PreMultiply (matTransl3D, matComposite);
}

/*  Procedure for generating a quaternion rotation matrix.  */
void rotate3D (wcPt3D p1, wcPt3D p2, GLfloat radianAngle)
{
   Matrix4x4 matQuatRot;

   float axisVectLength = sqrt ((p2.x - p1.x) * (p2.x - p1.x) +
                        (p2.y - p1.y) * (p2.y - p1.y) +
                        (p2.z - p1.z) * (p2.z - p1.z));
   float cosA = cosf (radianAngle);
   float oneC = 1 - cosA;
   float sinA = sinf (radianAngle);
   float ux = (p2.x - p1.x) / axisVectLength;
   float uy = (p2.y - p1.y) / axisVectLength;
   float uz = (p2.z - p1.z) / axisVectLength;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
/*  Set up translation matrix for moving p1 to origin,
 *  and concatenate translation matrix with matComposite.
 */
translate3D (-p1.x, -p1.y, -p1.z);

/*  Initialize matQuatRot to identity matrix.  */
matrix4x4SetIdentity (matQuatRot);

matQuatRot [0][0] = ux*ux*oneC + cosA;
matQuatRot [0][1] = ux*uy*oneC - uz*sinA;
matQuatRot [0][2] = ux*uz*oneC + uy*sinA;
matQuatRot [1][0] = uy*ux*oneC + uz*sinA;
matQuatRot [1][1] = uy*uy*oneC + cosA;
matQuatRot [1][2] = uy*uz*oneC - ux*sinA;
matQuatRot [2][0] = uz*ux*oneC - uy*sinA;
matQuatRot [2][1] = uz*uy*oneC + ux*sinA;
matQuatRot [2][2] = uz*uz*oneC + cosA;

/*  Concatenate matQuatRot with composite matrix.  */
matrix4x4PreMultiply (matQuatRot, matComposite);

/*  Construct inverse translation matrix for p1 and
 *  concatenate with composite matrix.
 */
translate3D (p1.x, p1.y, p1.z);
}

/*  Procedure for generating a 3D scaling matrix.  */
void scale3D (Gfloat sx, GLfloat sy, GLfloat sz, wcPt3D fixedPt)
{
   Matrix4x4 matScale3D;

   /*  Initialize scaling matrix to identity.  */
   matrix4x4SetIdentity (matScale3D);

   matScale3D [0][0] = sx;
   matScale3D [0][3] = (1 - sx) * fixedPt.x;
   matScale3D [1][1] = sy;
   matScale3D [1][3] = (1 - sy) * fixedPt.y;
   matScale3D [2][2] = sz;
   matScale3D [2][3] = (1 - sz) * fixedPt.z;

   /*  Concatenate matScale3D with composite matrix.  */
   matrix4x4PreMultiply (matScale3D, matComposite);
}

void displayFcn (void)
{
   /*  Input object description.  */
   /*  Input translation, rotation, and scaling parameters.  */

   /* Set up 3D viewing-transformation routines. */

   /*  Initialize matComposite to identity matrix:  */
   matrix4x4SetIdentity (matComposite);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    /*  Invoke transformation routines in the order they
     *  are to be applied:
     */
    rotate3D (p1, p2, radianAngle);  //  First transformation: Rotate.
    scale3D (sx, sy, sz, fixedPt);   //  Second transformation: Scale.
    translate3D (tx, ty, tz);        //  Final transformation: Translate.

    /*  Call routines for displaying transformed objects.  */
}
```

5.14 其他三维变换

除了平移、旋转和缩放以外，二维变换应用中讨论的其他变换也用于三维图形应用中。这些另外的变换包括反射、错切和坐标系间的变换。

5.14.1 三维反射

三维反射可以相对于给定的反射轴，或者相对于给定的反射平面来实现。一般来说，三维反射矩阵的建立类似于二维。相对于给定轴的反射等价于绕此轴旋转180°。相对于平面的反射等价于四维空间中的180°旋转。当反射平面是坐标平面（xy、xz或yz）时，可以将此变换看成是左手系和右手系之间的转换（参见附录A）。

图5.52给出了将坐标描述从右手系转换到左手系（或反过来）的反射。该变换改变了z坐标的符号，保持x坐标和y坐标值不变。相对于xy平面的点反射矩阵的表达式是

$$M_{\text{zreflect}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.114}$$

类似地，关于yz平面和xz平面的反射变换矩阵分别将x和y的值取反。相对于其他平面的反射可以由旋转及坐标平面反射复合而得。

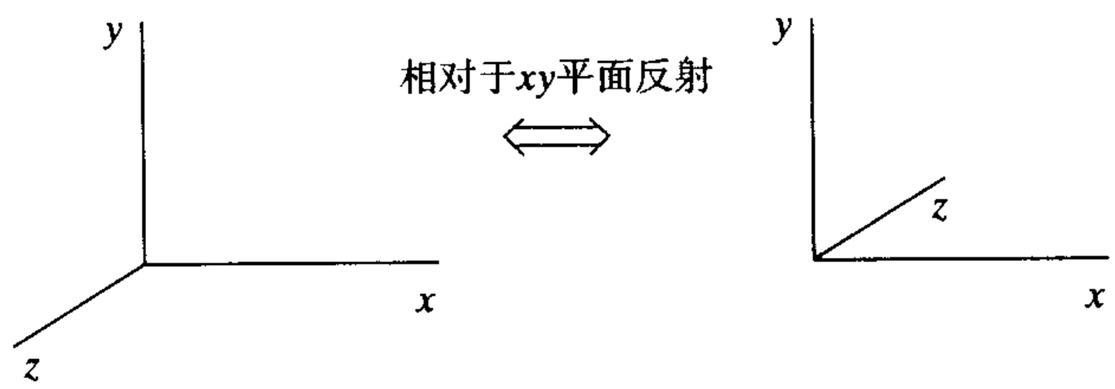


图5.52 坐标描述从右手系到左手系的变换可以使用反射变换式（5.114）来实现

5.14.2 三维错切

这些错切变换和二维中的一样可以用来修改对象形状，它们也可用于透视投影的三维观察中。相对于x或y轴的错切变换与5.5节讨论的相同，在三维空间中，我们还可以生成相对于z轴的错切。

相对于一选定参考位置的z轴错切变换使用下列矩阵生成。

$$M_{\text{zshear}} = \begin{bmatrix} 1 & 0 & \text{sh}_{zx} & -\text{sh}_{zx} \cdot z_{\text{ref}} \\ 0 & 1 & \text{sh}_{zy} & -\text{sh}_{zy} \cdot z_{\text{ref}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.115}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

错切参数 sh_{zx} 和 sh_{zy} 可赋以任何实数值。该变换矩阵的效果是使用与到 z_{ref} 的距离成比例的值来改变 x 和 y 坐标而不改变 z 坐标。与 z 轴垂直的平面区域使用与 $z - z_{ref}$ 等价的量移动。图 5.53 给出了在错切值为 $sh_{zx} = sh_{zy} = 1$ 且参考位置 $z_{ref} = 0$ 时该错切矩阵应用于单位立方体的效果。x 轴错切和 y 轴错切的三维变换矩阵与二维矩阵类似。对 z 轴错切只要增加一行和一列。

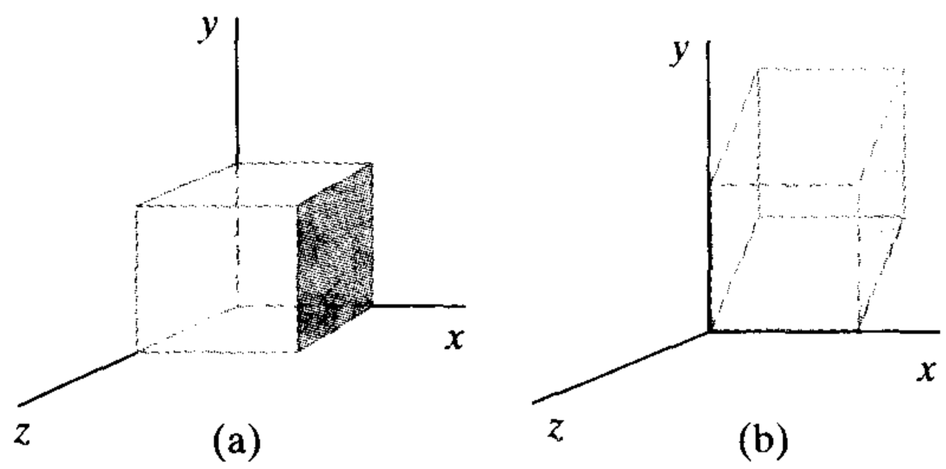


图 5.53 单位立方体(a)被变换矩阵（5.115）相对于原点错切成(b)，其中 $sh_{zx} = sh_{zy} = 1$

5.15 三维坐标系间的变换

在5.8节，我们讨论了将二维场景从一个参考坐标系统转换到另一个所需的操作。坐标系统变换在计算机图形软件包中用于场景的构造（建模）和实现二维及三维应用的观察子程序。如5.8节指出，将二维场景描述从一个坐标系统向另一个变换的矩阵利用迭加两个坐标系统的操作来构造。对于三维场景变换可应用同样的过程。

再次仅考虑笛卡儿参考系统并假定 $x'y'z'$ 系统在 xyz 系统中的定义。要将 xyz 坐标描述转换到 $x'y'z'$ 坐标系，先建立将 $x'y'z'$ 坐标原点移到 xyz 原点的平移。接着进行一系列的旋转使对应的坐标轴重合。如果两个坐标系使用不同的比例，则必须进行一次缩放变换以补偿两坐标系之间的差别。

图 5.54 给出了一个原点在 (x_0, y_0, z_0) 的 $x'y'z'$ 坐标系统和相对于 xyz 参考系统定义的单位轴向量。使用平移矩阵 $\mathbf{T}(-x_0, -y_0, -z_0)$ 将 $x'y'z'$ 坐标系原点与 xyz 原点对齐。可以使用单位轴向量来形成坐标轴旋转矩阵：

$$\mathbf{R} = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 \\ u'_{y1} & u'_{y2} & u'_{y3} & 0 \\ u'_{z1} & u'_{z2} & u'_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.116}$$

该矩阵将单位轴向量 $\mathbf{u}'_x$、$\mathbf{u}'_y$ 和 $\mathbf{u}'_z$ 分别转换到 x、y、z 轴上。完整的坐标变换序列通过复合矩阵 $\mathbf{R} \cdot \mathbf{T}$ 给出。该矩阵正确地将坐标描述从一个坐标系统变换到另一个，尽管一个系统是左手系而另一个是右手系也可以进行转换。

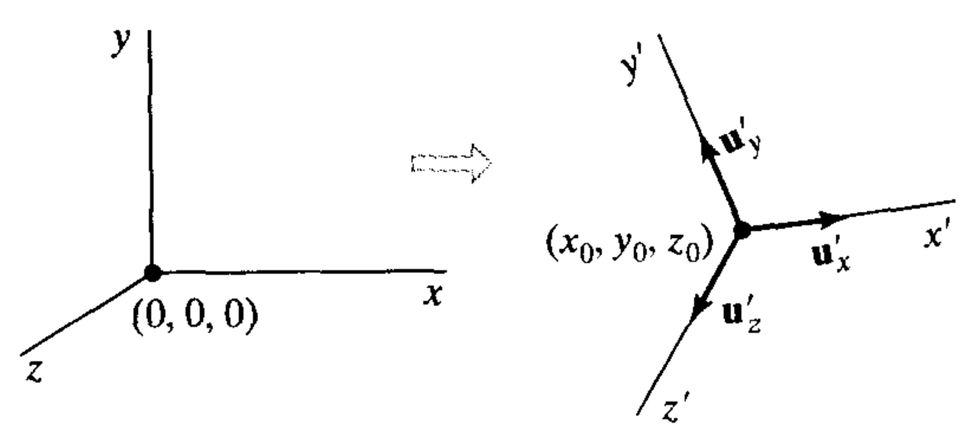


图 5.54 在 xyz 系统中定义的 $x'y'z'$ 坐标系统。使用将 $x'y'z'$ 坐标系统与 xyz 重叠的变换序列将场景描述变换到新的坐标系统

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

5.16 仿射变换

如果坐标变换的形式是

$$\begin{aligned} x' &= a_{xx}x + a_{xy}y + a_{xz}z + b_x \\ y' &= a_{yx}x + a_{yy}y + a_{yz}z + b_y \\ z' &= a_{zx}x + a_{zy}y + a_{zz}z + b_z \end{aligned} \tag{5.117}$$

则该变换称为**仿射变换**（affine transformation）。每一个变换后的坐标 x'、y' 和 z' 是原坐标 x、y 和 z 的线性函数，且参数 a_{ij} 和 b_k 是由变换类型确定的常数。仿射变换（在二维、三维或更高维中）有普遍的特性，即平行线变换到平行线且有限点变换到有限点。

平移、旋转、缩放、反射和错切是仿射变换的特例。任何仿射变换总可以表示成这五种变换的组合。场景的坐标描述从一个参考系统到另一个的转换也是仿射变换的一个例子，因为该变换可以利用平移和旋转的复合来描述。仅包括平移、旋转和反射的仿射变换保持角度和长度以及线条间的平行性不变。对于这三种变换的任意一种而言，线段长度和任意两条线之间的夹角在变换后保持不变。

5.17 OpenGL 几何变换函数

在OpenGL的核心库中，每一种基本的几何变换都有一个独立的函数，所有变换都在三维空间中定义。要完成一次平移，需要引用平移子程序且设定三维平移向量。在旋转函数中，需指定经过坐标系原点的旋转轴的角度和方向。而缩放函数用来设定相对于坐标系原点的三个坐标缩放系数。在每种情况中，变换子程序都建立一个 4 × 4 矩阵用于接收变换的对象坐标。

5.17.1 基本的 OpenGL 几何变换

4 × 4 平移矩阵用下列子程序构造。

```
glTranslate* (tx, ty, tz);
```

平移参数 tx、ty 和 tz 可赋以任意的实数值，而附加于该函数的单个后缀码或者是 f（浮点）或者是 d（双倍长浮点）。对于二维应用而言，设定 tz = 0.0。而一个二维位置表示成一个 z 分量为 0.0 的四元素列矩阵。由该函数生成的平移矩阵用来对调用此函数之后定义的对象的位置进行变换。作为一个例子，使用下列语句可将随后定义的坐标位置在 x 方向平移 25 个单位而在 y 方向平移 −10 个单位。

```
glTranslatef (25.0, -10.0, 0.0);
```

类似地，4 × 4 旋转矩阵用下列函数生成：

```
glRotate* (theta, vx, vy, vz);
```

这里，向量 v = (vx,vy,vz)的分量可以有任意的浮点数值。该向量用于定义通过坐标原点的旋转轴的方向。如果 v 并未定义成单位向量，则该旋转矩阵在用于计算之前会被规范化。后缀码可以是 f 或 d，而参数 theta 赋以旋转角度数，由该函数将其转换成弧度后再进行三角计算。这个函数使用等式（5.108）中的四元数计算旋转矩阵，该矩阵应用于在其后定义的位置。作为例子，下列语句：

```
glRotatef (90.0, 0.0, 0.0, 1.0);
```

设定绕 z 轴旋转 90° 的矩阵。

利用下列函数可得到相对于坐标原点的 4 × 4 缩放矩阵：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glScale* (sx, sy, sz);
```

后缀码还是f或d，而缩放参数可以赋以任何实数值。因此，当对缩放参数赋以负值时，该函数也可生成反射矩阵。例如，下列语句生成在x方向缩放2倍、在y方向缩放3倍，且相对于xz平面反射。

```
glScalef (2.0, -3.0, 1.0);
```

任何缩放参数为0会引起处理错误，因为其逆矩阵无法计算。该缩放-反射矩阵应用于在其后定义的对象。

5.17.2 OpenGL矩阵操作

2.9节指出了子程序glMatrixMode用来设定投影模式（projection mode），即指定将用于投影变换的矩阵。该变换确定怎样将一个场景投影到屏幕上。同样的子程序可用来设定几何变换矩阵。但此时将该矩阵看做建模观察矩阵（modelview matrix），它用于存储和组合几何变换。该矩阵也用于将几何变换与向观察坐标系的变换进行组合。建模观察模式用下列语句指定：

```
glMatrixMode (GL_MODELVIEW);
```

该语句指定了一个4×4建模观察矩阵作为**当前矩阵**（current matrix）。前一节讨论的OpenGL变换子程序用于修改建模观察矩阵，该矩阵用于对场景中的坐标位置进行变换。使用glMatrixMode函数还可以设定另外两个模式：纹理模式（texture mode）和颜色模式（color mode）。纹理模式用于映射表面的纹理图案，而颜色模式用于从一个颜色模型转换到另一个。后面几章将讨论观察、投影、纹理和颜色变换。目前，我们将讨论限定在几何变换细节。glMatrixMode函数的默认变量是GL_MODELVIEW。

建立建模观察模式（或任何其他模式）之后，调用变换子程序所生成的矩阵要与该模式的当前矩阵相乘。另外，我们可以对当前矩阵的元素赋值，OpenGL库中有两个函数可用于此目的。使用下列函数可设定当前矩阵为单位矩阵。

```
glLoadIdentity ( );
```

另外，可以使用其他值为当前矩阵的元素赋值：

```
glLoadMatrix* (elements16);
```

参数elements16指定了一个单下标、16元素的浮点值数组，而后缀f或d用来指定数据类型。该数组的元素必须按列优先次序指定。即先列出第一列的4个元素，接着列出第二列的4个元素，然后是第三列，而最后是第四列。我们用下列程序对建模观察矩阵进行初始化，以便说明该次序。

```
glMatrixMode (GL_MODELVIEW);

GLfloat elems [16];
GLint k;

For (k = 0; k < 16; k++)
  elems [k] = float (k);
glLoadMatrixf (elems);
```

该程序生成下面的矩阵：

$$\mathbf{M} = \begin{bmatrix} 0.0 & 4.0 & 8.0 & 12.0 \\ 1.0 & 5.0 & 9.0 & 13.0 \\ 2.0 & 6.0 & 10.0 & 14.0 \\ 3.0 & 7.0 & 11.0 & 15.0 \end{bmatrix}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们也可以将指定的矩阵与当前矩阵合并：

```
glMultMatrix* (otherElements16);
```

后缀码还是 f 或 d，而参数 otherElements16 是一个按列优先的另一个矩阵的 16 元素、单下标数组。当前矩阵以后乘方式与 glMultMatrix 中指定的矩阵相乘，其积取代当前矩阵。因此，假定当前矩阵是指定的建模观察矩阵 **M**，则更新后的矩阵按下式计算：

$$\mathbf{M} = \mathbf{M} \cdot \mathbf{M}'$$

这里$\mathbf{M}'$代表由前面 glMultMatrix 语句中的参数 otherElements16 指定元素的矩阵。

函数 glMultMatrix 也可用来设定任一单独指定矩阵的变换序列。例如，

```
glMatrixMode (GL_MODELVIEW);

glLoadIdentity ( );          //Set current matrix to the identity.
glMultMatrixf (elemsM2);     //Postmultiply identity with matrix M2.
glMultMatrixf (elemsM1);     //Postmultiply M2 with matrix M1.
```

生成下面的建模观察矩阵：

$$\mathbf{M} = \mathbf{M}_2 \cdot \mathbf{M}_1$$

在这个序列中最先应用的变换是程序中最后指定的。因此，如果我们在 OpenGL 中建立一个变换序列，可设想为将各个变换装进一个栈，因而最后的操作最先应用。这并不代表实际上是这么处理的，但用栈来比喻可帮助读者记住，在 OpenGL 程序中变换序列是描述的相反次序。

记住 OpenGL 按列优先次序存储矩阵是同样重要的。OpenGL 中引用矩阵元素 m_{jk} 是引用 j 列和 k 行的元素。这与标准的数学转换中行号先引用的情况相反。我们可以通过将 OpenGL 矩阵指定为 16 元素、单下标数组并记住按列优先次序列出其元素的方法来避免行 - 列引用的错误。

5.17.3 OpenGL 矩阵栈

对可用 glMatrixMode 选择的 4 种模式（建模观察、投影、纹理和颜色）中的每一种，OpenGL 都维护一个矩阵栈。开始，每一栈仅包含单位矩阵。在处理场景的任何时刻，栈顶的矩阵称为该模式的“当前矩阵”。在指定观察和几何变换后，**建模观察栈**（modelview matrix stack）顶是一个应用于场景的观察变换和各种几何变换的 4 × 4 复合矩阵。有时，需要创建多个视图和变换序列，然后分别保存复合矩阵。因此，OpenGL 提供至少深度为 32 的建模观察栈，而有的实现允许在建模观察栈中保存超过 32 个矩阵。我们可以使用下列语句来确定 OpenGL 的特定实现中建模观察栈的有效位置数。

```
glGetIntegerv (GL_MAX_MODELVIEW_STACK_DEPTH, stackSize);
```

它将一个整数值返回给数组 stackSize。另外三个矩阵模式的栈深度至少为 2，我们可以使用下列符号常量之一来确定每一个特定实现中的最大有效深度：GL_MAX_PROJECTION_STACK_DEPTH、GL_MAX_TEXTURE_STACK_DEPTH 或 GL_MAX_COLOR_STACK_DEPTH。

也可以使用下列语句找到栈中现有多少个矩阵：

```
glGetIntegerv (GL_MODELVIEW_STACK_DEPTH, numMats);
```

开始，建模观察栈仅包含单位矩阵，因此如果我们在发生任何栈处理前调用该函数则返回值 1。类似的符号常量可用来确定其他三个栈中的当前矩阵数。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

OpenGL有两个函数用来处理栈中的矩阵。这些栈处理函数比单独管理栈矩阵效率要高，尤其是在栈函数使用硬件实现时。例如，硬件实现可同时复制多个矩阵元素。而我们可以在栈中保留一个单位矩阵，从而在当前矩阵初始化时实现得比调用 glLoadIdentity 更快。

使用下列函数，可以复制活动栈顶的当前矩阵并将其存入第二个栈位置。

```
glPushMatrix ( );
```

这给了栈顶两个位置以双份矩阵，另一个栈函数是

```
glPopMatrix ( );
```

它破坏栈顶矩阵，而栈的第二个矩阵成为当前矩阵。如果要“弹出”栈顶，栈内至少要有两个矩阵；否则就会出错。

5.17.4　OpenGL 几何变换编程示例

在下面的程序段中，我们对一个矩形应用每一个基本的几何变换，每次一个。开始时，建模观察矩阵是单位矩阵且显示一个蓝色矩形。接着，将当前颜色重新设为红色，指定二维平移参数，并显示红色的平移后的矩形（参见图5.55）。因为我们不需要组合变换，所以接下来设定当前矩阵为单位矩阵。然后构造一个旋转矩阵并与当前矩阵（单位矩阵）合并。在再次引入原来的矩形时，它绕 z 轴旋转并显示成红色（参见图5.56）。再重复一次该过程就生成图5.57所示的缩放和反射后的红色矩形。

```
glMatrixMode (GL_MODELVIEW);
glColor3f (0.0, 0.0, 1.0);
glRecti (50, 100, 200, 150);        // Display blue rectangle.

glColor3f (1.0, 0.0, 0.0);
glTranslatef (-200.0, -50.0, 0.0); // Set translation parameters.
glRecti (50, 100, 200, 150);        // Display red, translated rectangle.

glLoadIdentity ( );                 // Reset current matrix to identity.
glRotatef (90.0, 0.0, 0.0, 1.0);    // Set 90-deg. rotation about z axis.
glRecti (50, 100, 200, 150);        // Display red, rotated rectangle.

glLoadIdentity ( );                 // Reset current matrix to identity.
glScalef (-0.5, 1.0, 1.0);          // Set scale-reflection parameters.
glRecti (50, 100, 200, 150);        // Display red, transformed rectangle.
```

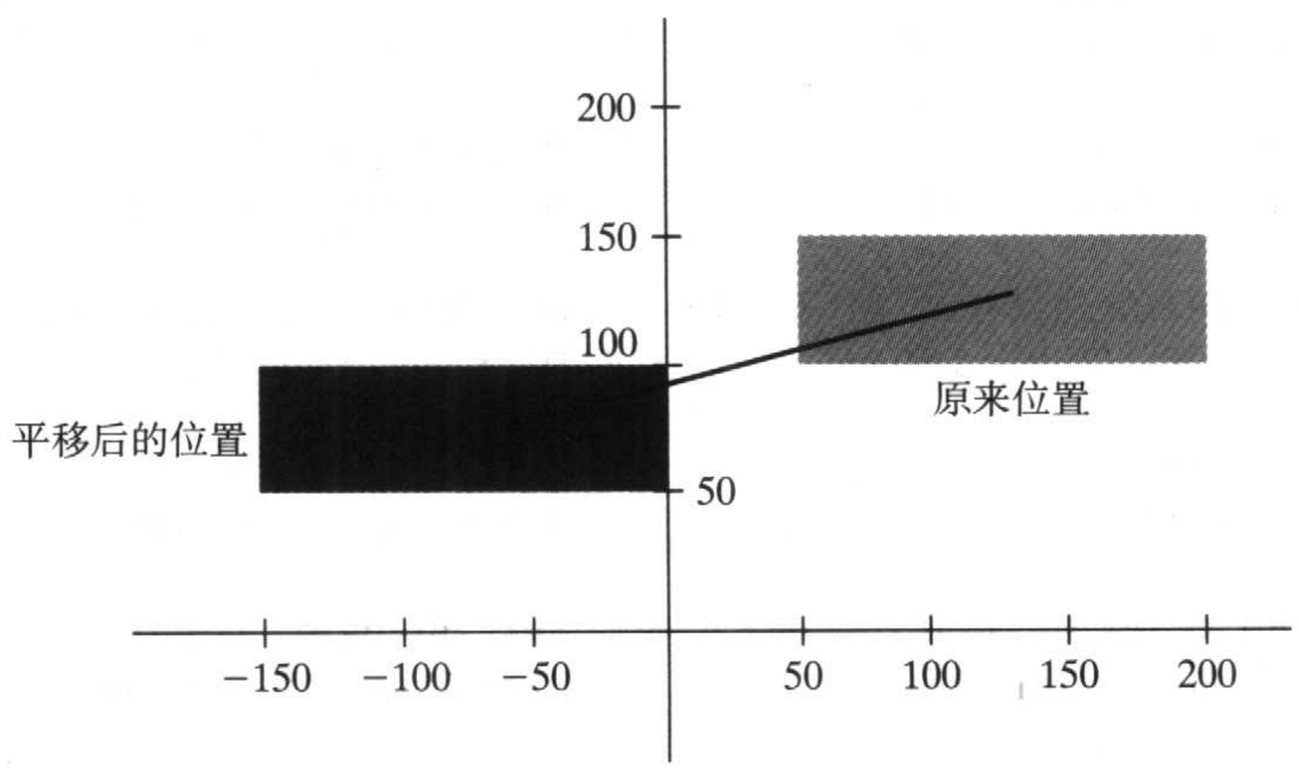


图 5.55　使用 OpenGL 的函数 glTranslatef (−200.0, −50.0, 0.0)平移一个矩形

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

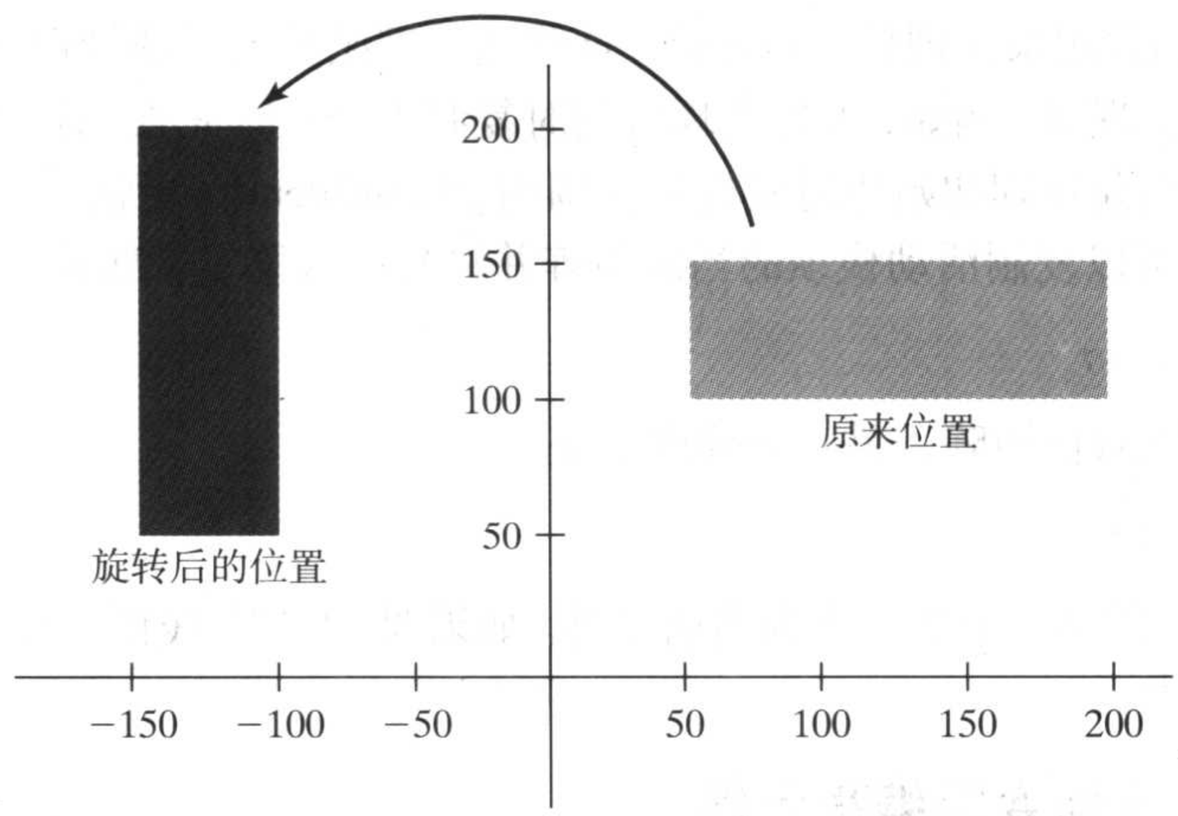


图 5.56　使用 OpenGL 的函数 glRotatef (90.0, 0.0, 0.0, 1.0)绕轴旋转一个矩形

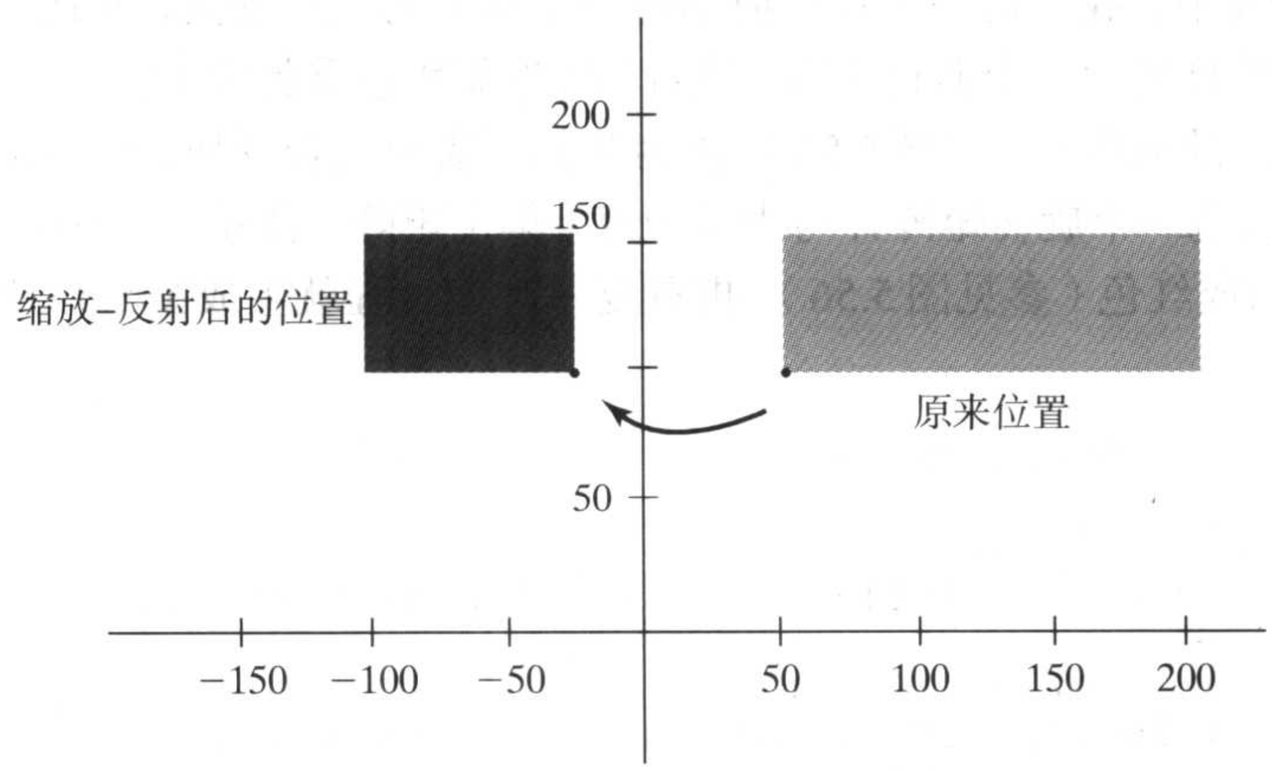


图 5.57　使用 OpenGL 的函数 glScalef (–0.5, 1.0, 1.0)缩放和反射一个矩形

通常，使用栈处理函数比使用矩阵管理函数更高效。在需要对观察或几何变换进行某些修改时更是这样。在下面的程序中，我们使用栈处理函数代替glLoadIdentity函数来重复上例中矩形的变换。

```
glMatrixMode (GL_MODELVIEW);

glColor3f (0.0, 0.0, 1.0);          // Set current color to blue.
glRecti (50, 100, 200, 150);        // Display blue rectangle.

glPushMatrix ( );                   // Make copy of identity (top) matrix.
glColor3f (1.0, 0.0, 0.0);          // Set current color to red.

glTranslatef (-200.0, -50.0, 0.0); // Set translation parameters.
glRecti (50, 100, 200, 150);        // Display red, translated rectangle.

glPopMatrix ( );                    // Throw away the translation matrix.
glPushMatrix ( );                   // Make copy of identity (top) matrix.

glRotatef (90.0, 0.0, 0.0, 1.0);    // Set 90-deg. rotation about z axis.
glRecti (50, 100, 200, 150);        // Display red, rotated rectangle.

glPopMatrix ( );                    // Throw away the rotation matrix.
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glScalef (-0.5, 1.0, 1.0);          // Set scale-reflection parameters.
glRecti (50, 100, 200, 150);        // Display red, transformed rectangle.
```

对于最后一个几何变换编程例子，我们给出 5.13 节的三维、复合变换的 OpenGL 版本。由于 OpenGL 的变换矩阵在被调用时后乘，故必须以和应用相反的次序引入变换。因此，每一个后继的变换调用将指定变换矩阵放在复合矩阵右边进行合并。因为我们还未探讨三维OpenGL观察子程序（参见第 7 章），这个程序可以用二维 OpenGL 观察操作并对 *xy* 平面的对象进行几何变换来完成。

```
class wcPt3D {
   public:
      GLfloat x, y, z;
};

\* Procedure for generating a matrix for rotation about an
 * an axis defined with points p1 and p2.
 */
void rotate3D (wcPt3D p1, wcPt3D p2, GLfloat thetaDegrees)
{
   /*  Set up components for rotation-axis vector.  */
   float vx = (p2.x - p1.x);
   float vy = (p2.y - p1.y);
   float vz = (p2.z - p1.z);

   /* Specify translate-rotate-translate sequence in reverse order: */
   glTranslatef (p1.x, p1.y, p1.z); // Move p1 back to original position.
   /*  Rotate about axis through origin:  */
   glRotatef (thetaDegrees, vx, vy, vz);
   glTranslatef (-p1.x, -p1.y, -p1.z);  // Translate p1 to origin.
}

/*  Procedure for generating a matrix for a scaling
 *  transformation with respect to an arbitrary fixed point.
 */
void scale3D (GLfloat sx, GLfloat sy, GLfloat sz, wcPt3D fixedPt)
{
   /* Specify translate-scale-translate sequence in reverse order: */
   /* (3) Translate fixed point back to original position: */
   glTranslatef (fixedPt.x, fixedPt.y, fixedPt.z);
   glScalef (sx, sy, sz);      // (2) Scale with respect to origin.
   /* (1) Translate fixed point to coordinate origin: */
   glTranslatef (-fixedPt.x, -fixedPt.y, -fixedPt.z);
}

void displayFcn (void)
{
   /*  Input object description.  */
   /*  Set up 3D viewing-transformation routines. */
   /*  Display object.  */

  glMatrixMode (GL_MODELVIEW);

   /* Input translation parameters tx, ty, tz. */
   /* Input the defining points, p1 and p2, for the rotation axis. */
   /* Input rotation angle in degrees. */
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    /* Input scaling parameters: sx, sy, sz, and fixedPt. */

    /*  Invoke geometric transformations in reverse order:  */
    glTranslatef (tx, ty, tz);      // Final transformation: Translate.
    scale3D (sx, sy, sz, fixedPt);     // Second transformation: Scale.
    rotate3D (p1, p2, thetaDegrees);  // First transformation: Rotate.

    /*  Call routines for displaying transformed objects.  */
}
```

5.18 小结

基本的几何变换是平移、旋转和缩放。平移将一个对象从一个位置沿直线路径移动到另一位置。旋转将一个对象从一个位置绕指定旋转轴沿圆周路径移动到另一位置。对于二维应用而言，该旋转路径位于在 xy 平面内且围绕与 z 轴平行的轴。缩放变换改变对象相对于固定点的尺寸。

可以用 3×3 矩阵操作表示二维变换而用 4×4 矩阵操作表示三维变换，从而使一系列变换可合并成一个复合矩阵。或者，更一般地可以用 4×4 矩阵表示二维和三维变换。用矩阵表示几何变换是一种有效的形式，因为它允许通过将复合矩阵应用于对象描述获得变换后的位置来减少计算。为此，坐标位置用列矩阵表示。选择列矩阵表示坐标位置是因为这是一种标准的数学惯例，且多数图形软件包现在使用这一惯例。一个三元素或四元素的列矩阵（向量）看做齐次坐标表示。对于几何变换，齐次系数赋为 1。

复合变换由平移、旋转、缩放和其他变换的矩阵乘积来形成。可以将平移和旋转的复合用于动画应用，将旋转和缩放的复合应用于在任一指定方向缩放对象。一般情况下，矩阵乘法不符合交换律。例如，改变平移-旋转序列的次序会导致不同的结果。仅包含平移和旋转的变换序列是一个刚体变换，因为角度和距离都不变。同样，刚体变换的左上角子矩阵是一个正交矩阵。因此，旋转矩阵可通过设定与两个正交单位向量相同的左上角 3×3 子矩阵来形成。当角度较小时，我们可以使用 sin 和 cos 函数的一阶近似来减少旋转计算。然而，经历若干次旋转后，近似误差可能积累成有影响的值。

其他变换包括反射和错切。反射是将对象绕反射轴旋转 180° 的变换。这生成一个相对于该轴的对象镜像。当反射轴垂直于 xy 平面时，反射可通过在 xy 平面的旋转来得到。当反射轴在 xy 平面上时，反射可通过在垂直于 xy 平面的平面上旋转来得到。错切变换通过按与到错切参考线的距离成比例的量移动一个或多个坐标值来改变对象的形状。

笛卡儿坐标系统间的变换用一组使两个坐标系变成一致的平移和旋转变换来实现。通过在原坐标系中指定坐标原点和轴向量来确定一个新坐标系。对于二维系统而言，一个向量可完全确定坐标轴方向。但对于三维系统，则必须指定三个轴之中两个的方向。对象描述从原坐标系到第二个坐标系的变换通过将新原点平移到旧原点的矩阵和将两组坐标轴对齐的旋转矩阵相乘所得的矩阵来计算。对齐两个坐标系所需的旋转可从为新系统建立的一组正交轴向量来得到。

几何变换是仿射变换，即它们可表示成坐标位置的线性函数。平移、旋转、缩放、反射和错切都是仿射变换。它们将平行线仍变换为平行线且将有限坐标位置仍变换为有限位置。不包含缩放和错切的几何变换也保持角度和长度不变。

可以将光栅操作应用于像素数组的某些简单几何变换。对于二维应用，可利用光栅操作完成快速平移、反射和 90° 倍数的旋转。稍做一点处理，可以完成一般的光栅旋转和缩放。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

OpenGL 基本库包含三个函数，用于对坐标位置进行单独的平移、旋转和缩放变换。每个函数生成一个与建模观察前乘的矩阵。因此，几何变换必须按逆向次序指定。最后引入的变换最先应用于坐标位置。变换矩阵应用于随后定义的对象。除了在建模观察矩阵中累积变换序列，我们可以将该矩阵设定为单位矩阵或某种其他矩阵。还可以使用建模观察矩阵和任意指定的矩阵形成乘积。所有矩阵存储在栈中，而 OpenGL 为图形应用中使用的各种类型的变换维护 4 个栈。我们可以使用 OpenGL 的查询函数确定当前栈大小或系统的最大允许栈深度。可使用两个栈处理子程序：一个用来将栈顶矩阵复制到第二位置，而另一个则用来移走栈顶矩阵。OpenGL 中有若干种操作可用来完成光栅变换。使用这些光栅操作可平移、旋转、缩放或反射一个光栅块。

表 5.1 总结了本章讨论的 OpenGL 几何变换函数和矩阵子程序。另外，该表还列出了某些相关的函数。

表 5.1　OpenGL 几何变换函数小结

函数	描述
glTranslate*	指定平移参数
glRotate*	指定绕过原点的任意轴旋转的参数
glScale*	指定相对于坐标原点的缩放参数
glMatrixMode	为几何 - 观察变换、投影变换、纹理变换或颜色变换指定当前矩阵
glLoadIdentity	将当前矩阵设定为单位矩阵
glLoadMatrix* (elems);	设定当前矩阵的元素
glMultMatrix* (elems);	用指定矩阵前乘当前矩阵
glGetIntegerv	为选择的矩阵模式获取最大深度和当前矩阵数
glPushMatrix	复制栈顶矩阵并将其存入第二个栈位置
glPopMatrix	删除栈顶矩阵并将第二位置矩阵移到栈顶
glPixelZoom	为光栅操作指定二维缩放参数

参考文献

关于矩阵建模和几何变换的其他技术请参见Glassner(1990)、Arvo(1991)、Kirk(1992)、Heckbert(1994)和 Paeth(1995)。计算机图形学齐次坐标的讨论请参见 Blinn and Newell(1978) 和 Blinn(1993, 1996, and 1998)。

用 OpenGL 几何变换函数编程的其他例子请参见 Woo, Neider, Davis, and Shreiner(1999)。使用 OpenGL 几何变换函数编程的例子还可参见 Nate Robins 的辅导网站：http://www.cs.utah.edu/narobins/opengl.html。OpenGL 几何变换的完整列表在 Shreiner(2000)中提供。

练习题

5.1　编写一个实现 5.1 节中二维旋转示例的动画程序。一个输入多边形绕 xy 平面上的基准点连续旋转，每次旋转一个小角度，并对 sin 和 cos 函数进行近似计算以加快计算速度。为了避免坐标误差的积累，在每次新的旋转开始时，重新设置对象原来的坐标值。

5.2　通过对 $\mathbf{R}(\theta_1)$和 $\mathbf{R}(\theta_2)$矩阵表达式的合并得到 $\mathbf{R}(\theta_1)\cdot\mathbf{R}(\theta_2)=\mathbf{R}(\theta_1+\theta_2)$，证明两个旋转的复合是可相加的。

5.3　对在任意方向缩放的变换矩阵（5.39）进行修改，使其包括任意指定的缩放固定点(x_f, y_f)的坐标。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

5.4 证明对于下列每个操作序列，矩阵相乘是可交换的：
(a) 两个连续的旋转。
(b) 两个连续的平移。
(c) 两个连续的缩放。

5.5 证明一致缩放和旋转形成可交换的操作对，但通常缩放和旋转不是可交换操作。

5.6 将方程（5.42）中的单个缩放、旋转和平移矩阵相乘，以验证复合变换矩阵中的各元素。

5.7 修改 5.4 节的示例程序使变换参数可通过用户输入来指定。

5.8 修改上题中的程序使变换序列可用于任意多边形，其顶点通过用户输入来指定。

5.9 修改 5.4 节中的示例程序，使几何变换的次序可通过用户输入来指定。

5.10 证明关于直线 $y = x$ 的反射变换矩阵（5.55）等价于相对 x 轴的反射加上逆时针旋转 90°。

5.11 证明关于直线 $y = -x$ 的反射变换矩阵（5.56）等价于相对 y 轴的旋转加上逆时针旋转 90°。

5.12 证明相对于 x 或 y 坐标轴的两次连续反射，等价于在 xy 平面上关于坐标原点的一次旋转。

5.13 确定相对于任意直线 $y = mx+b$ 的反射变换矩阵的形式。

5.14 证明相对于 xy 平面上任何通过坐标原点的直线的两次连续反射，等价于相对于原点的单个旋转。

5.15 确定等价于 x 方向错切矩阵（5.57）的基本变换序列。

5.16 确定等价于 y 方向错切矩阵（5.61）的基本变换序列。

5.17 根据给定的向量字体定义，建立显示二维斜体字符的错切程序。即这种字体中的所有字符形状都使用直线段进行定义，斜体字符则根据错切变换形成。通过比较斜体和某些可用字体的正常文本来确定错切参数的合适值。请读者为自己的程序输入定义简单的向量字体。

5.18 推导下列方程，可以将一个笛卡儿坐标系中的坐标点 $\mathbf{P} = (x, y)$ 变换到由该系统相对于自己逆时针旋转 θ 角所得的另一个笛卡儿系统中的坐标值 (x', y')。变换等式可通过将点 $\mathbf{P}$ 投影到四个轴并分析所得到的合适的三角形来获得。

$$x' = x\cos\theta + y\sin\theta \qquad y' = -x\sin\theta + y\cos\theta$$

5.19 编写一个将对象描述从一个二维笛卡儿系统变换到另一个系统时计算矩阵元素的程序。第二个坐标系将使用原点 $\mathbf{P}_0$ 和给出其正 y' 轴方向的向量 $\mathbf{V}$ 进行定义。

5.20 建立实现帧缓存矩形区域的块转移程序，其中使用一个函数将该区域读入一个数组，并使用另一个函数将该数组复制到指定的转移区域。

5.21 确定使用各种布尔操作将两个连续的块转移到帧缓存相同区域的执行结果。

5.22 使用算术运算操作，将两个连续的块转移到帧缓存相同区域的执行结果是什么？

5.23 实现使用任何指定的布尔操作或替代（复制）操作来完成帧缓存中块转移的程序。

5.24 编写一个在帧缓存块转移中以 90° 倍数实现旋转的程序。

5.25 编写一个在帧缓存块转移中以任意指定角度实现旋转的程序。

5.26 编写一个实现将缩放作为像素块的光栅变换的程序。

5.27 证明旋转矩阵（5.102）等价于复合矩阵 $\mathbf{R}_y(\beta)\cdot\mathbf{R}_x(\alpha)$。

5.28 计算方程（5.106）中的项，推导矩阵（5.107）中给定的一般旋转矩阵的元素。

5.29 证明当旋转轴是 z 轴时，四元数旋转矩阵（5.107）退化成等式（5.74）中的矩阵表示。

5.30 证明等式（5.109）等价于等式（5.97）中所给出的一般旋转变换。

5.31 利用三角一致性，从矩阵（5.107）推导出四元旋转矩阵（5.108）的元素。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

5.32 设计对于任意给定轴，增量地旋转一个三维对象的动画程序。利用合适的三角函数近似计算来加速计算，并在绕轴完整地循环一圈后将对象重定位到原来的位置。

5.33 推导在 α、β 和 γ 定义的方向用缩放参数 s 缩放一个对象的三维变换矩阵。

5.34 设计三维对象相对于任选平面反射的程序。

5.35 利用输入值作为错切参数，写出一个三维对象相对于三个坐标轴中任意一轴的错切程序。

5.36 设计将定义在一个三维坐标参照系中的对象变换到与第一个系统有关的另一个坐标系的程序。

5.37 实现 5.17 节的示例程序，使三维 OpenGL 几何变换函数用于图 5.15(a)的二维三角形以生成图 5.15(b)中示例的变换。

5.38 修改上题中的程序使该变换序列可应用于任意的二维多边形，其中顶点通过用户输入来指定。

5.39 修改上题中的程序使该几何变换序列的次序可通过用户输入来指定。

5.40 修改上题中的程序使该几何变换的参数可通过用户输入来指定。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第6章 二维观察

第2章已经简要介绍了二维观察的概念和函数。现在我们深入讨论在输出设备上显示二维图的问题。一般来说，图形软件让用户指定已定义的图中哪部分要显示，以及在显示设备的什么位置显示。任何一个称为世界坐标系的常规笛卡儿坐标系统都可以用来定义图形。在二维情况下，可使用*xy*平面上的包含全图或任意部分的区域来选择视图。用户可以只选择一个区域，也可以同时显示几个区域，或者显示一个场景的动态扫视序列。所选区域中的图形映射到设备坐标系的指定区域中。当选择多个观察区域时，这些区域分别放在不同的显示位置，或者将某些区域插入其他的大区域中。从世界坐标系到设备坐标系的变换包括平移、旋转、缩放操作以及删除位于显示区域范围以外的图形部分。

6.1 二维观察流水线

二维场景中要显示的部分称为**裁剪窗口**（clipping window），因为所有在此区域之外的场景均要裁去。只有在裁剪窗口内部的场景才能显示在屏幕上。裁剪窗口有时暗指世界窗口（world window）或观察窗口（viewing window）。图形系统曾一度简称裁剪窗口为“窗口”，但由于现在有众多的窗口系统在计算机上使用，因此必须把它们区分开来。例如，窗口管理系统创建和管理监视器上的若干个区域，其中每一区域称为一个“窗口”，它可用来显示图形和文字（参见图6.1）。因此，我们永远使用术语裁剪窗口来表示可能要转换为监视器上某显示窗口的点阵的场景部分。图形系统还用称为**视口**（viewport）的另一“窗口”来控制在显示窗口的定位。对象在裁剪窗口内的部分映射到显示窗口中指定位置的视口中。窗口选择要看什么，而视口指定在输出设备的什么位置进行观察。

图6.1 同时显示多个窗口的监视器（Sun公司提供）

通过改变视口的位置，我们可以在输出设备的显示区域的不同位置观察物体。使用多个视口可在不同的屏幕位置观察场景的不同部分。我们也可以通过改变视口的尺寸来改变显示对象的尺寸和位置。如果将不同尺寸的裁剪窗口连续映射到固定尺寸的视口中，则可以得到“拉镜头”的效果。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

当裁剪窗口越变越小时，就可以聚焦到场景中的某一部分，从而观察到使用较大的裁剪窗口时未显示出的细节。同样，通过从一个场景部分开始连续地放大的裁剪窗口，可以得到逐步扩大的场景。通过将一个固定尺寸的裁剪窗口移过场景中的不同的对象，就可以产生“摇镜头”的效果。

裁剪窗口和视口一般都是正则矩形，其各边分别与坐标轴平行。有时也会采用多边形和圆形等其他形状的窗口和视口，但是处理时间长一些。我们先考虑如图 6.2 所示的矩形视口和裁剪窗口。

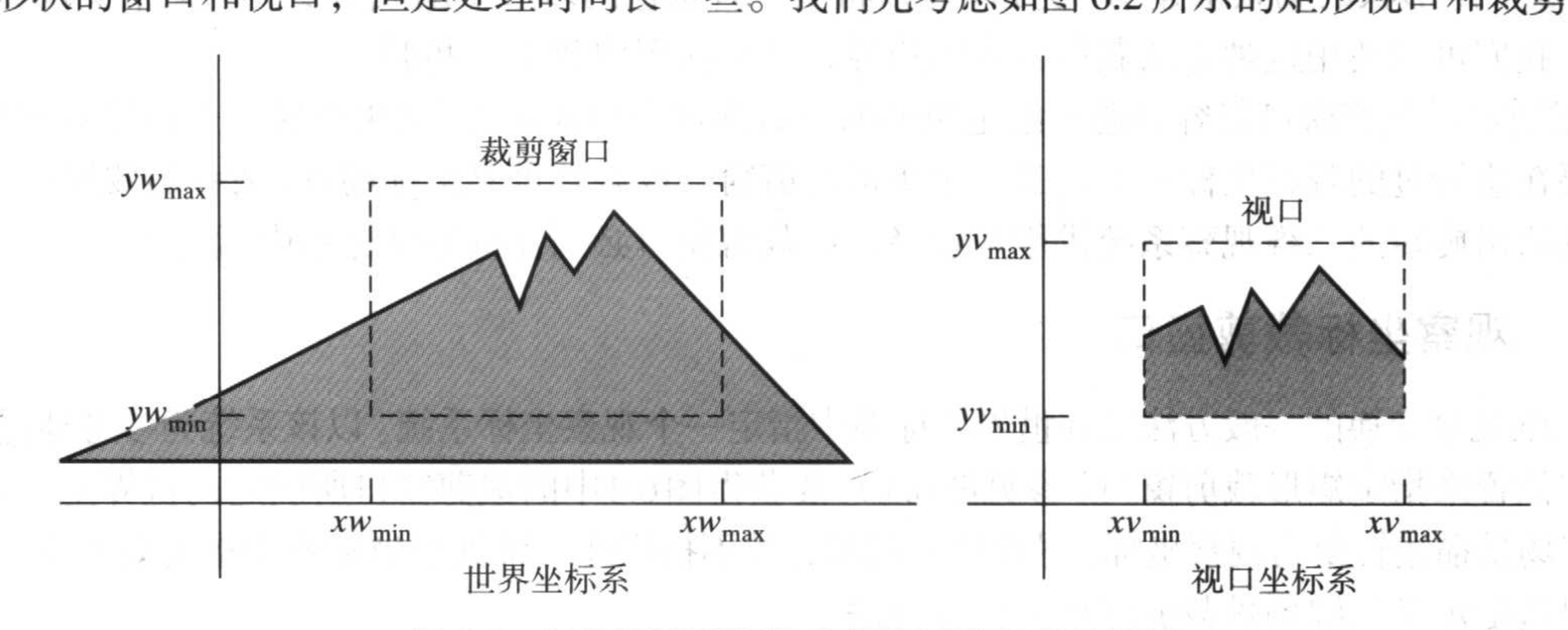


图 6.2　与坐标轴平行的裁剪窗口及对应的视口

二维世界坐标系场景描述到设备坐标系的映射称为**二维观察变换**（two-dimensional viewing transformation）。有时将二维观察变换简单地称为窗口到视口的变换（window-to-viewport transformation）或窗口变换（window transformation）。一般来说，观察包含的内容不仅仅是从窗口到视口的变换。图 6.3 给出了与三维观察相仿的二维观察步骤。构造世界坐标系的场景后，我们可以建立独立的**二维观察坐标参考系**（viewing-coordinate reference frame）来指定裁剪窗口。由于裁剪窗口常在世界坐标系中定义，因此二维应用的观察坐标系和世界坐标系一致。（而三维场景则需要独立的观察坐标系来指定观察位置、观察方向和坐标系方向等参数。）

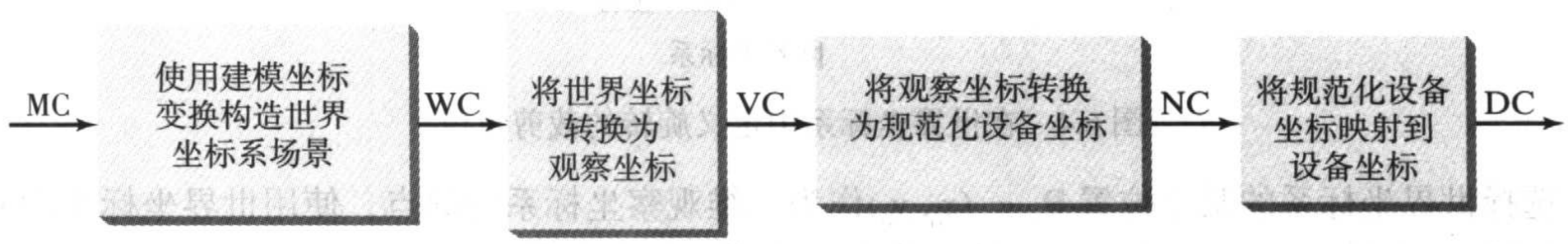


图 6.3　二维观察变换流水线

为了使观察处理独立于输出设备，图形系统将对象描述转换到规范化设备坐标系并提供裁剪程序。有些系统的规范化设备坐标范围从 0 到 1 而另一些从 -1 到 1。视口在规范化设备坐标系还是在设备坐标系中定义则取决于使用什么图形库。在观察变换的最后一步，视口中的内容转换到显示窗口的相应位置。

裁剪工作通常在规范化设备坐标系中进行。这使我们可以在此之前合并变换矩阵从而减少计算时间。裁剪函数是计算机图形学中重要的基本函数。它们不仅用于观察变换也用于窗口管理系统、绘画软件中擦除部分图片及其他许多方面。

6.2　裁剪窗口

应用程序要得到特殊的裁剪效果，可通过选择裁剪窗口的不同形状、大小和方向来实现。例如，可以使用星形模子、椭圆或由样条曲线围成的形状作为裁剪窗口。但使用凹多边形或非线性边

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

界裁剪窗口来裁剪比用矩形裁剪要花费更多时间。确定对象与圆的交点比确定它与直线的交点需要更多的计算。最简单的用于裁剪的窗口边界是与坐标轴平行的直线段。因此，图形软件一般仅允许使用平行于 x 和 y 轴的矩形裁剪窗口。

如果要使用其他形状的裁剪窗口，就必须自己实现裁剪和坐标变换算法。也可以对图进行编辑，生成一定形状的场景来显示。例如，可以将填上背景色的多边形围成所要的图案来实现对图的修剪。我们可以使用这种方法获得任意的边界，甚至在图中放上一些洞。

正则矩形裁剪窗口很容易通过给定矩形的一对顶点坐标来定义。如果要转一个角度观察场景，则需要在旋转过的观察坐标系中定义一个矩形裁剪窗口或旋转世界坐标场景，两者的效果一样。有些系统给出旋转的二维观察系统供用户选择，但裁剪窗口必须在世界坐标系中指定。

6.2.1 观察坐标裁剪窗口

二维观察变换的一般方法是在世界坐标系中指定一个观察坐标系统。以该系统为参考通过选定方向和位置来指定矩形裁剪窗口（参见图6.4）。要获得图6.4中的裁剪窗口所确定的世界坐标场景，只需将场景描述转换为观察坐标。尽管许多图形软件不提供在二维观察坐标系中指定裁剪窗口的功能，但这是定义三维场景裁剪区域的标准方法。

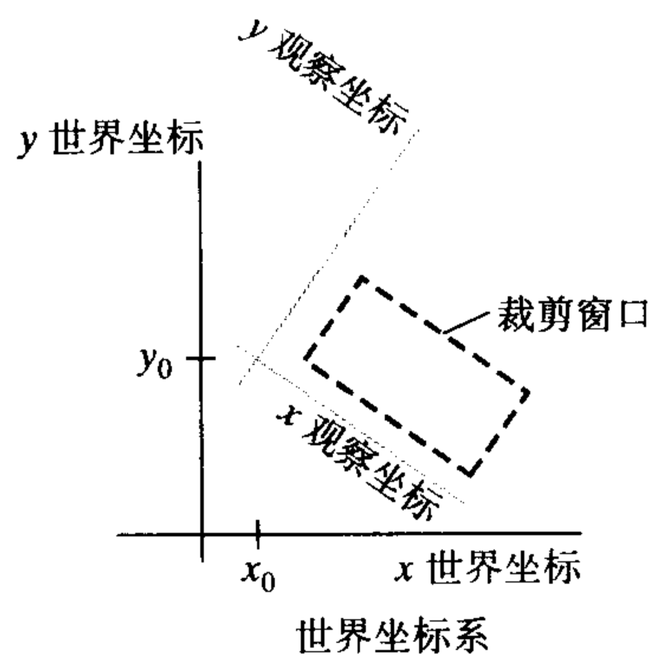


图 6.4 在世界坐标系中定义旋转的裁剪窗口

选择世界坐标系的某个位置 $\mathbf{P}_0 = (x_0, y_0)$ 作为二维观察坐标系的原点，使用世界坐标系的向量 $\mathbf{V}$ 作为观察坐标系 y_{view} 轴的方向。向量 $\mathbf{V}$ 称为**二维观察向上向量**（view up vector）。另一种指定观察坐标系方向的方法是在世界坐标系中给定观察坐标系相对于 x 或 y 坐标轴的转动角度。根据旋转角度可以获得观察向上向量。一旦确定了观察坐标系的参数，就可使用5.8节的函数将场景描述变换到观察坐标系，其中引入的变换相当于在世界坐标系上叠加观察坐标系。

变换的第一步是将观察坐标系原点移动到与世界坐标系原点重合。接着，旋转观察坐标系使其与世界坐标系重合。给定方向向量 $\mathbf{V}$，可为 x_{view} 和 y_{view} 轴分别计算出单位向量 $\mathbf{v} = (v_x, v_y)$ 和 $\mathbf{u} = (u_x, u_y)$。这两个向量用来形成将观察系 $x_{\text{view}}y_{\text{view}}$ 轴与世界系 x_w 和 y_w 轴重合的旋转变换矩阵 $\mathbf{R}$ 的第 1 列和第 2 列。

对象在世界坐标系的位置随后由组合的二维变换矩阵转换到观察坐标系中：

$$\mathbf{M}_{WC,VC} = \mathbf{R} \cdot \mathbf{T} \tag{6.1}$$

这里 $\mathbf{T}$ 是将观察坐标系原点与世界坐标系原点重合的平移变换，$\mathbf{R}$ 是使平移后的观察坐标系与世界坐标系重合的旋转变换。图 6.5 示出了该坐标变换的步骤。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

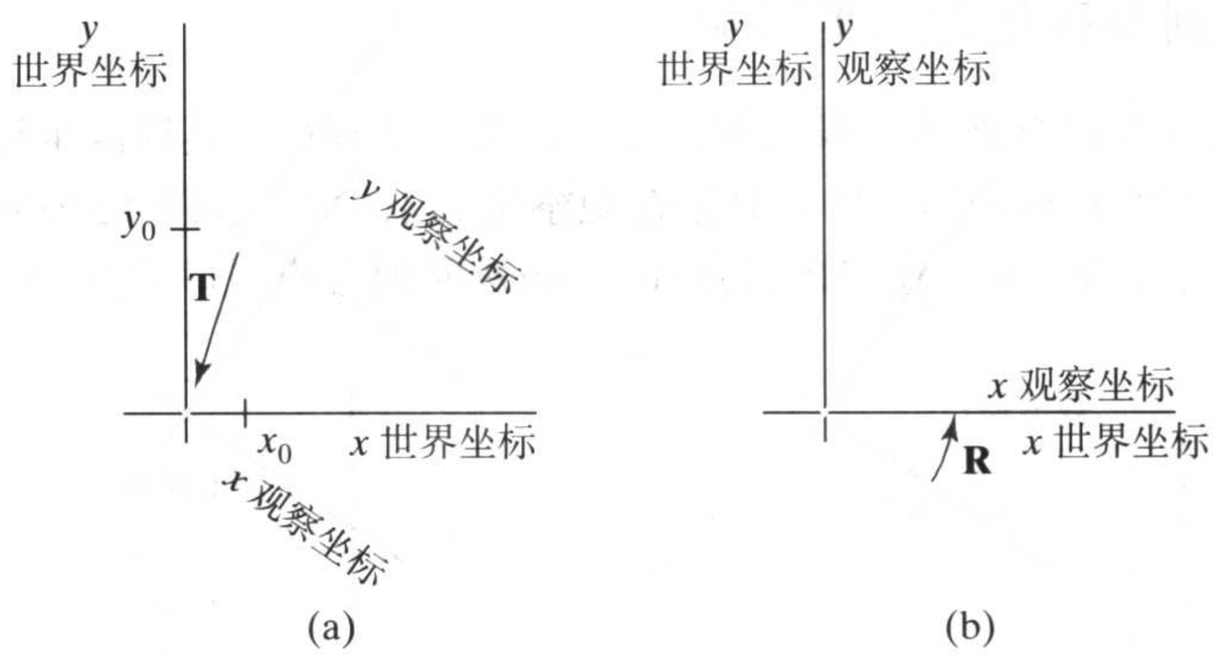


图 6.5　观察坐标系变换成与世界坐标系重合的步骤：(a)平移变换使两坐标系原点重合；(b)旋转变换使两系统的坐标轴分别重合

6.2.2　世界坐标系裁剪窗口

图形系统程序库中一般均提供定义标准矩形裁剪窗口的函数。我们可以指定世界坐标系中的两点作为标准矩形的两个对角顶点。一旦建立了裁剪窗口，观察函数就对场景描述进行处理并将结果送到输出设备。

如上所述，如果要旋转一个二维场景，需要完成与此相同的步骤，但不考虑观察坐标系。因此，可以简单地在世界坐标系中旋转（可能有平移）对象到所需位置并建立裁剪窗口。例如，将图 6.6(a)中的三角形旋转到指定位置并建立一个标准的裁剪矩形之后就可以显示旋转后的三角形。与上面所述的坐标变换类似，我们可以将三角形绕原点旋转并定义一个包围该三角形的裁剪窗口。那样，我们定义一个方向向量并选择一个参考点（参见附录 A），比如三角形的中心，接着将参考点平移到世界坐标系原点并将方向向量利用变换矩阵（6.1）旋转到 y_{world} 轴。三角形转到了所希望的方向后，可以使用世界坐标系中的标准裁剪窗口来获取旋转后的三角形。图 6.6(b)给出了变换后的三角形位置和所选的裁剪窗口。

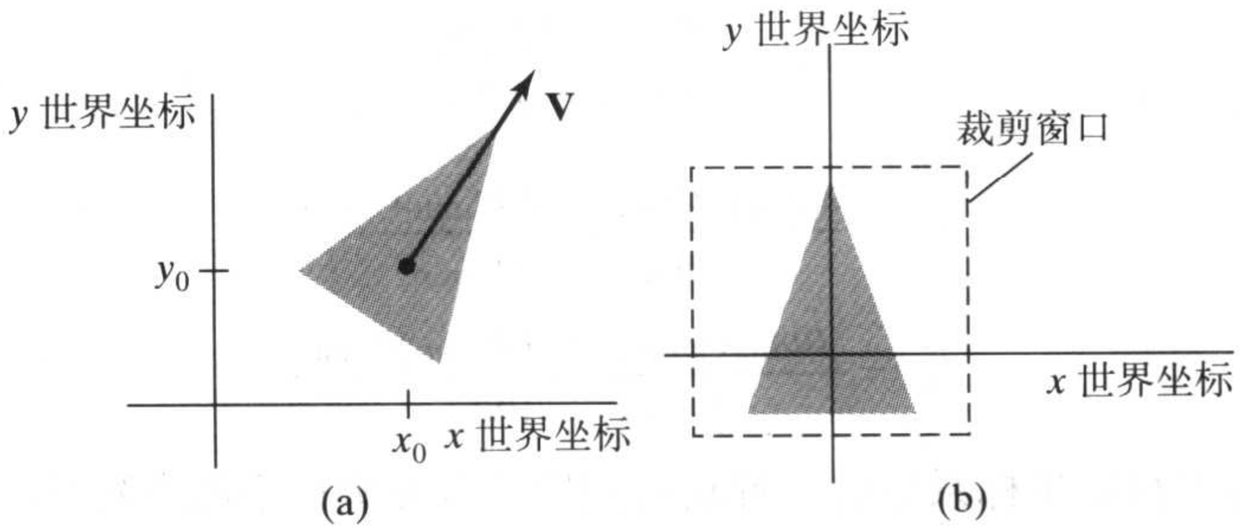


图 6.6　(a)一个三角形及选定的参考点和定向向量；(b)平移和旋转后位于裁剪窗口中

6.3　规范化和视口变换

有些图形系统将规范化和窗口——视口转换合并成一步。这样，视口坐标是从 0 到 1，即视口位于一个单位正方形内。裁剪后包含视口的单位正方形映射到输出显示设备。在其他一些系统中，规范化和裁剪在窗口——视口转换之前进行。这些系统的视口边界在与显示窗口位置对应的屏幕坐标系指定。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

6.3.1 裁剪窗口到规范化视口的映射

为了说明规范化和视口变换的一般过程，首先定义一个视口，其规范坐标值从0到1。按点的变换方式将对象描述变换到该视口。如果对象在观察坐标系中心，则它也必然显示在视口的中心。图6.7显示了窗口到视口的映射。窗口内的点(xw, yw)映射到对应视口的点(xv, yv)。

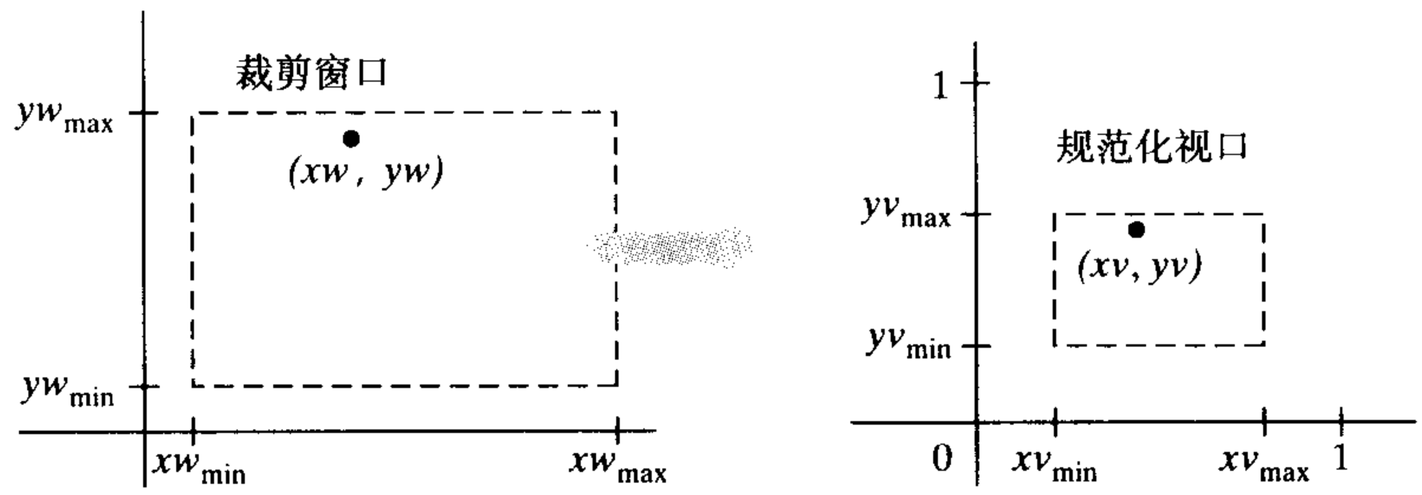


图6.7 在指定窗口中位于(xw, yw)的点映射到视口中坐标为(xv, yv)的点，因此这两点在两个区域的相对位置相同

为了保持视口与窗口中的对象具有同样的相对位置，必须满足：

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$
$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}} \tag{6.2}$$

对上述方程求解(xv, yv)，可得

$$xv = s_x xw + t_x$$
$$yv = s_y yw + t_y \tag{6.3}$$

其中，缩放系数为

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$
$$s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} \tag{6.4}$$

平移参数为

$$t_x = \frac{xw_{max}xv_{min} - xw_{min}xv_{max}}{xw_{max} - xw_{min}}$$
$$t_y = \frac{yw_{max}yv_{min} - yw_{min}yv_{max}}{yw_{max} - yw_{min}} \tag{6.5}$$

由于我们简单地把世界坐标系中的位置映射到位于世界坐标系原点附近的视口，等式(6.3)也可以根据从窗口到视口的变换而推出。该变换按下列顺序进行：

1. 以点(xw_{min}, yw_{min})为中心执行缩放变换，将窗口变换成视口的大小。
2. 将(xw_{min}, yw_{min})移到(xv_{min}, yv_{min})。

第一步的缩放变换可以表示成二维矩阵：

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & xw_{min}(1 - s_x) \\ 0 & s_y & yw_{min}(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \tag{6.6}$$

这里s_x和s_y与等式（6.4）中的相同。裁剪窗口左下角平移到视口左下角的二维矩阵是

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{T}=\begin{bmatrix}1 & 0 & xv_{\min}-xw_{\min}\\ 0 & 1 & yv_{\min}-yw_{\min}\\ 0 & 0 & 1\end{bmatrix} \tag{6.7}$$

变换到规范化视口的组合矩阵是

$$\mathbf{M}_{\text{window, normviewp}}=\mathbf{T}\cdot\mathbf{S}=\begin{bmatrix}s_x & 0 & t_x\\ 0 & s_y & t_y\\ 0 & 0 & 1\end{bmatrix} \tag{6.8}$$

该变换给出和等式（6.3）同样的结果。裁剪窗口的任一其他参考点，如右上角或窗口中心，都可用在缩放-平移的操作中。也可以先将裁剪窗口位置平移到视口的对应位置，再相对于视口位置进行缩放。

窗口-视口变换保持对象描述的相对位置。在裁剪窗口内部的对象映射到视口内，在裁剪窗口外部的对象则映射到视口之外。

另外，只有在裁剪窗口和视口有相同的纵横比时才能保持对象的相对比例不变。换句话说，如果缩放因子 sx 和 sy 相同，则对象的比例不变。否则，世界坐标系中的对象显示到输出设备上时可能在 x 或 y 方向上（或同时）被拉长或缩短。

裁剪函数可使用裁剪窗口边界或视口边界实现裁剪。裁剪后，规范化坐标变换到设备坐标。单位正方形经过与窗口-视口变换相同的过程而映射到输出设备，其内部全部变换到输出设备的显示区域。

6.3.2　裁剪窗口到规范化正方形的映射

二维观察的另一种方法是先将裁剪窗口变换到规范化正方形，在规范化坐标系中进行裁剪，然后将场景描述变换到屏幕坐标系中指定的视口中。图6.8给出了规范化坐标范围从-1到1的这种变换。在这个变换过程中裁剪操作是标准的，因此在由 $x=\pm1$ 和 $y=\pm1$ 确定的边界外的对象被检测出来并从场景描述中移走。在观察变换的最后一步，视口中的对象定位到显示窗口。

裁剪窗口到规范化正方形的变换使用了与窗口-视口变换一样的过程。规范化变换的矩阵由等式（6.8）中用-1代入 $xv_{\min}$ 和 $yv_{\min}$、用+1代入 $xv_{\max}$ 和 $yv_{\max}$ 而获得。上述代入在 t_x、t_y、s_x 和 s_y 用表达式代入后进行，有

$$\mathbf{M}_{\text{window, normsquare}}=\begin{bmatrix}\dfrac{2}{xw_{\max}-xw_{\min}} & 0 & -\dfrac{xw_{\max}+xw_{\min}}{xw_{\max}-xw_{\min}}\\ 0 & \dfrac{2}{yw_{\max}-yw_{\min}} & -\dfrac{yw_{\max}+yw_{\min}}{yw_{\max}-yw_{\min}}\\ 0 & 0 & 1\end{bmatrix} \tag{6.9}$$

类似地，在裁剪操作完成后，边长为2的正方形变换到指定的视口。这时，在等式（6.8）中用-1代入 $xw_{\min}$ 和 $yw_{\min}$、用+1代入 $xw_{\max}$ 和 $yw_{\max}$，即可得变换矩阵：

$$\mathbf{M}_{\text{normsquare, viewport}}=\begin{bmatrix}\dfrac{xv_{\max}-xv_{\min}}{2} & 0 & \dfrac{xv_{\max}+xv_{\min}}{2}\\ 0 & \dfrac{yv_{\max}-yv_{\min}}{2} & \dfrac{yv_{\max}+yv_{\min}}{2}\\ 0 & 0 & 1\end{bmatrix} \tag{6.10}$$

观察过程的最后一步是将视口在显示窗口中定位。一般将视口左下角定位到与显示窗口左下角对应的坐标位置。图6.9给出了显示窗口中视口的定位情况。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

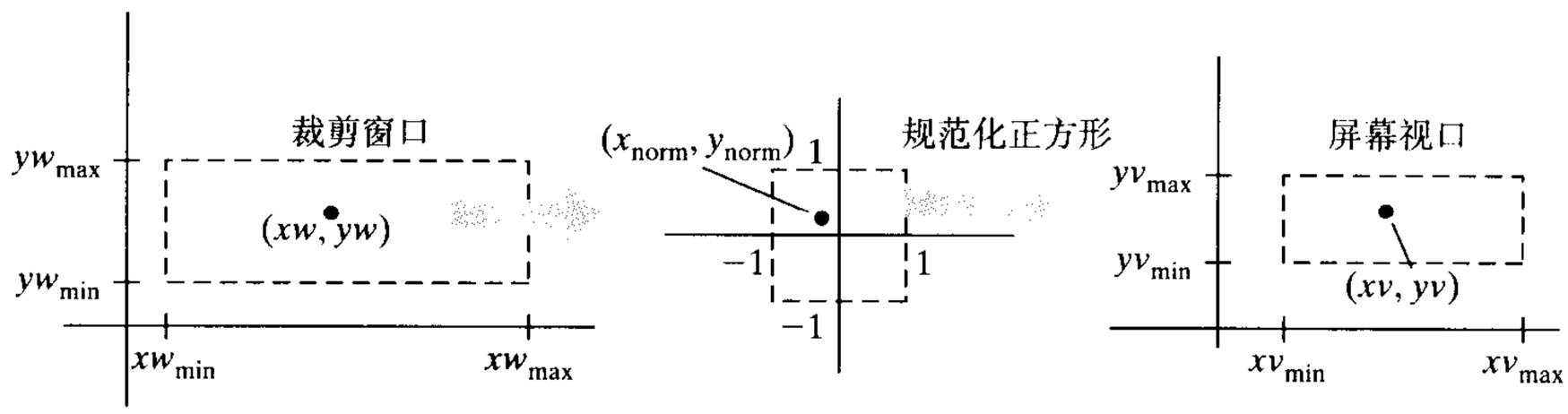


图 6.8　裁剪窗口中一个点(xw, yw)映射到规范化坐标系中的位置(x_{norm}, y_{norm})，然后再到视口的屏幕坐标位置(xv, yv)。在变换到视口坐标系之前用规范化正方形裁剪对象

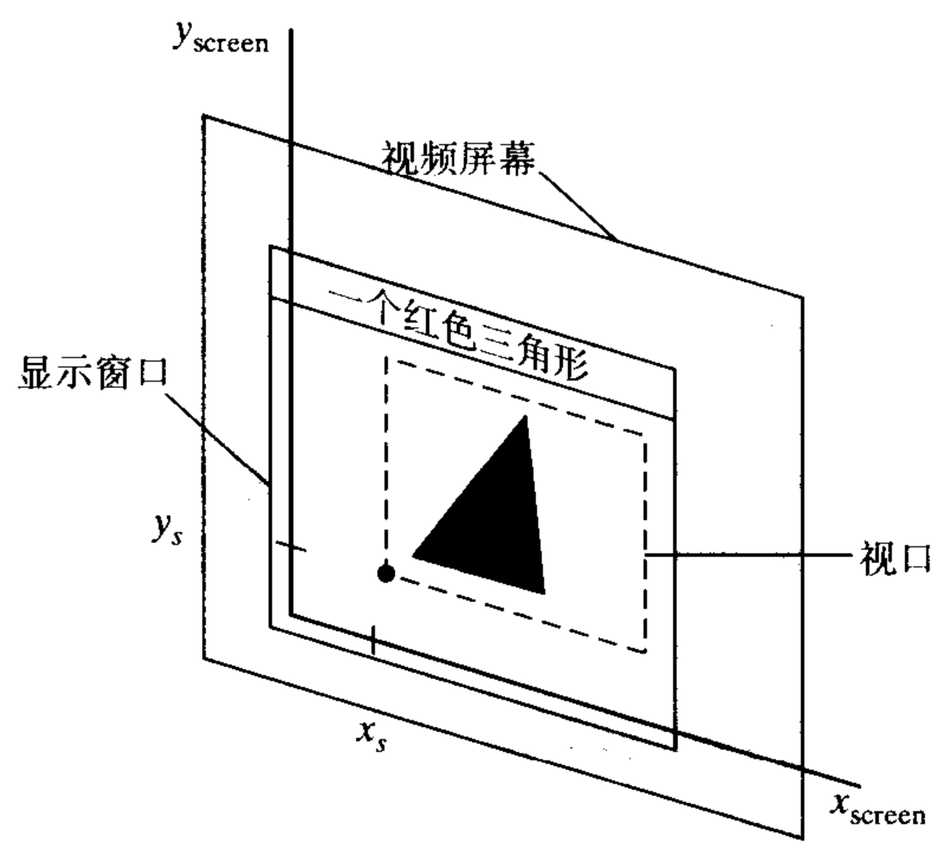


图 6.9　一个位于显示窗口中坐标位置(x_s, y_s)的视口

如前所述，通过选择与裁剪窗口有相同纵横比的视口可保持对象的初始比例不变。否则，对象可能在x或y方向拉长或缩短。显示窗口的纵横比也会影响对象的比例。如果视口映射到显示窗口的整个范围且显示窗口的尺寸有改变，对象就有可能变形，除非视口的纵横比跟着调整。

6.3.3　字符串的显示

有两种处理方法通过观察流水线将字符串映射到视口上。最简单的映射是保持字符串的大小不变。在使用点阵字体时采用这种方法。但轮廓线字体可以和其他图元一样变换，即只要对字型轮廓中线段的定义位置进行变换。在场景中处理过其他图元后对变换后的字符确定像素图案。

6.3.4　分画面效果和多输出设备

通过为一个场景选择不同的裁剪窗口及配对的视口，可同时显示两个或多个对象、多个图片部分或单个场景的不同观察。我们可以将这些画面放到同一个显示窗口的不同位置或放到屏幕上的多个显示窗口中。例如在设计应用中，可以在一个视口中显示线框图而在另一视口中显示完整绘制的图，并且在第三个视口中列出其他信息或菜单。

在一个特定系统内同时使用两个或多个输出设备以及为每个输出设备建立多对裁剪窗口/视口也是可以的。到所选输出设备的映射称为**工作站变换**（workstation transformation）。此时，视口在具体的显示设备的坐标系中指定。或者每一视口都在单位正方形中指定，然后把正方形映射到选定的设备。一些图形系统为此提供了一对工作站函数。一个函数用于为选定输出设备指定裁剪窗口，用工作站号（workstation number）标识；另一个函数用来为该设计建立相应的视口。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

6.4　OpenGL 二维观察函数

实际上，由于 OpenGL 主要为三维应用而设计，其基本库中没有专为二维观察而设计的函数。但我们可以将三维观察函数用于二维场景且核心库中包含一个视口函数。另外，OpenGL 实用函数库（GLU）提供了指定裁剪窗口的二维函数及处理显示窗口的 GLUT 函数。因此，我们可以使用这些二维函数以及 OpenGL 观察函数进行所需的观察操作。

6.4.1　OpenGL 投影模式

在选择OpenGL裁剪窗口和视口之前，必须建立合适的模式以便构建从世界坐标系到屏幕坐标系变换的矩阵。在 OpenGL 中，不能建立独立的图 6.4 所示的二维观察坐标系，必须将裁剪窗口的参数作为投影变换的一部分来设置。因此，必须先选择投影模式。我们可以使用在几何变换中设定建模观察模式的函数来设置。下列定义一个裁剪窗口和视口的函数将应用于投影矩阵。

```
glMatrixMode(GL_PROJECTION);
```

这指明投影矩阵作为当前矩阵，它原来设定为单位矩阵。然而，如果我们要回过来获得场景的另一观察，则可以建立初始化：

```
glLoadIdentity( );
```

这保证在每次进入投影模式时矩阵重新设定为单位矩阵，因此新的观察参数不会与前面的观察参数混在一起。

6.4.2　GLU 裁剪窗口函数

定义一个二维裁剪窗口使用下列 OpenGL 实用函数：

```
gluOrtho2D(xwmin,xwmax,ywmin,ywmax);
```

裁剪窗口边界的坐标位置使用双精度浮点数给出。该函数给出了将场景映射到屏幕的正交投影。对于三维场景来说，这意味着将对象沿垂直于二维 *xy* 显示平面的平行线投影。但是在二维投影中，对象是在二维 *xy* 平面上定义的。因此，正交投影对二维场景除了将对象位置转换到规范化坐标系之外没有其他作用。但是，由于二维场景要交给完整的三维 OpenGL 观察流水线处理，所以必须指定正交投影。实际上我们可以使用 gluOrtho2D 函数的三维 OpenGL 核心库版本（参见 7.10 节）。

OpenGL 裁剪函数使用 -1 到 1 的规范化坐标范围。gluOrtho2D 函数设定变换矩阵（6.9）的三维版本，将裁剪窗口中的对象映射到规范化坐标系。规范化正方形外的对象（及在裁剪窗口外的对象）不在显示的场景中出现。

如果没有为应用程序指定裁剪窗口，就使用默认的坐标$(xw_{min}, yw_{min}) = (-1.0, -1.0)$和$(xw_{max}, yw_{max}) = (1.0,1.0)$。这样，默认的裁剪窗口是以坐标系原点为中心、边长为 2 的规范化正方形。

6.4.3　OpenGL 视口函数

下列 OpenGL 函数用来指定视口参数：

```
glViewport(xvmin,yvmin,vpWidth,vpHeight);
```

这里的所有参数用对应于显示窗口的整数屏幕坐标给出。参数 xvmin 和 yvmin 指定视口左下角的位置，它与显示窗口的左下角对应。视口的宽度像素数和高度像素数用参数 vpWidth 和 vpHeight 来设定。如果我们并未在程序中使用 glViewport，则默认的视口大小及位置与显示窗口一样。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

使用裁剪函数后，再用矩阵（6.10）将规范化正方形中的位置变换到视口矩形。视口的右上角坐标由该变换矩阵及视口的宽和高计算而得：

$$xv_{\max} = xv_{\min} + vpWidth, \qquad yv_{\max} = yv_{\min} + vpHeight \tag{6.11}$$

最后，视口中图元的像素颜色装入指定的屏幕位置的刷新缓存中。

OpenGL 可以为各种应用建立多个视口（参见 6.3 节）。获取当前活动视口参数的查询函数是

```
glGetIntegerv(GL_VIEWPORT,vpArray);
```

这里的 vpArray 是一个单下标、四元素的矩阵。这一 Get 函数将当前视口的参数按 xvmin、yvmin、vpWidth、vpHeight 的顺序返回给 vpArray。例如在交互式应用中，我们可以使用该函数获得光标所在视口的参数。

6.4.4 建立 GLUT 显示窗口

2.9 节已经简单介绍了 GLUT 库的某些函数。由于 GLUT 库与任意窗口管理系统接口，使用 GLUT子程序来建立和管理显示窗口可以使程序例子不依赖于任意特定的计算机。为了调用这些子程序，必须先使用下列函数来对 GLUT 进行初始化。

```
glutInit(&argc,argv);
```

该初始化函数的参数与主程序的相同，可以使用 glutInit 处理命令行变量。

有三个 GLUT 函数用来定义显示窗口和选择其尺寸及位置：

```
glutInitWindowPosition(xTopLeft,yTopLeft);
glutInitWindowSize(dwWidth,dwHeight);
glutCreateWindow("Title of Display Window");
```

其中第一个函数给出显示窗口的左上角相对于屏幕左上角的整数型屏幕坐标位置。如果两个坐标值均为负，则显示窗口在屏幕上的位置由窗口管理系统确定。第二个函数按正整数的像素尺寸给出显示窗口的宽和高。如果并未使用这两个函数来指定尺寸和位置，则默认尺寸为 300 × 300 而默认位置是(-1, -1)，显示窗口位置的决定权交给窗口管理系统。任何情况下，只要窗口管理系统得到有关的说明或要求，就可忽略用GLUT函数指定的显示窗口尺寸和位置。这样，窗口系统可以另外确定显示窗口的尺寸和位置。第三个函数用指定的尺寸和位置以及指定的标题来建立显示窗口，但标题是否使用也取决于窗口系统。这时，定义好的显示窗口在 GLUT 的 setup 操作完成前不会在屏幕上出现。

6.4.5 设定 GLUT 显示窗口的模式和颜色

显示窗口的参数由下列 GLUT 函数选择：

```
glutInitDisplayMode(mode);
```

该函数用来选择颜色模式（RGB 或索引号）和不同的缓存组合，所选参数以逻辑‘或’操作方式组合。默认模式是单缓存和 RGB（或 RGBA）颜色模式，这与使用下列语句设定的一样：

```
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
```

指定为GLUT_RGB 颜色模式与GLUT_RGBA 是等价的。显示窗口的背景颜色用下列 OpenGL 子程序在 RGB 模式中选择：

```
glClearColor(red, green, blue, alpha);
```

在颜色索引模式下，我们用

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glClearIndex(index);
```

来设定显示窗口的颜色，这里的参数 index 被赋以与表达颜色表中位置相对应的整数值。

6.4.6 GLUT 显示窗口标识

一个应用可以建立多个显示窗口，其中每一个都要设定一个整数的**显示窗口标识**(display-window identifier)，从数值 1 开始赋给建立的第一个窗口。在初始化一个显示窗口时，可以使用下列语句记录它的标识：

```
windowID = glutCreateWindow ("A Display Window");
```

一旦在整数变量WindowID中保存了这个整型显示窗口标识，就可以使用该标识号来改变显示参数和删除显示窗口。

6.4.7 删除 GLUT 显示窗口

GLUT库中也包含删除一个已建立的显示窗口的函数。如果我们知道该显示窗口的标识，则可使用下列语句删除它：

```
glutDestroyWindow (windowID);
```

6.4.8 当前 GLUT 显示窗口

我们指定的任何显示窗口操作都针对当前显示窗口，即最后建立的**当前显示窗口**(current display window) 或用下列命令指定的显示窗口。

```
glutSetWindow (windowID);
```

在任何时候可以通过询问系统来确定当前的显示窗口是哪一个：

```
currentWindowID = glutGetWindow ( );
```

在没有显示窗口或当前显示窗口已被删除的情况下返回 0。

6.4.9 GLUT 显示窗口位置和大小的修改

如果要改变当前显示窗口的位置，使用

```
glutPositionWindow (xNewTopLeft, yNewTopLeft);
```

这里的坐标指定显示窗口左上角相对于屏幕左上角的新位置。类似地，下列函数设定当前显示窗口的尺寸。

```
glutReshapeWindow (dwNewWidth, dwNewHeight);
```

使用如下函数可将当前显示窗口扩展到整个屏幕。

```
glutFullScreen ( );
```

执行该函数后的显示窗口实际尺寸取决于窗口管理系统。在此之后对 glutPositionWindow 或 glutReshapeWindow 的调用将取消扩展到整个屏幕的要求。

无论何时改变显示窗口，都可能改变窗口的纵横比并使对象变形。如 3.24 节所述，可以使用下列语句来调整显示窗口的变化：

```
glutReshapeFunc (winReshapeFcn);
```

这一GLUT函数在显示窗口的尺寸改变时被激活，新的宽和高将送给其所属变量：在此例中就是函数 winReshapeFcn。因此，winReshapeFcn 是一个“重定型事件”的“回调函数”。可以使用该回调

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

函数来改变视口的参数从而保持场景原有的纵横比。另外，也可重新设定裁剪窗口的边界、改变显示窗口的颜色、调整其他观察参数并完成任何其他任务。

6.4.10 管理多个 GLUT 显示窗口

GLUT库中还有一些用来以多种方式管理显示窗口的函数。当屏幕上有多个显示窗口且需要对它们重新安排或对其中的某个显示窗口重定位时，这些函数特别有用。

使用下列函数可将当前显示窗口变为一个图符，该图符通过小图片或符号形式来表示该窗口。

```
glutIconifyWindow ( );
```

该图符将用赋予该窗口的名字来标记，但我们可以用下列函数改变其名字：

```
glutSetIconTitle ("Icon Name");
```

也可以用类似的命令改变显示窗口的名字：

```
glutSetWindowTitle ("New Window Name ");
```

屏幕上打开多个显示窗口时，有些窗口可能部分或完全覆盖另外的窗口。可以通过先指定某个显示窗口为当前窗口，然后调用“pop-window”命令来使它成为所有其他窗口之前的窗口。

```
glutSetWindow (windowID);
glutPopWindow ( );
```

可以用类似的方法将当前显示窗口“压”到背后，使它位于所有其他显示窗口之后。操作顺序是

```
glutSetWindow (windowID);
glutPushWindow ( );
```

也可以让当前窗口从屏幕上消失：

```
glutHideWindow ( );
```

我们可以通过将隐藏的或变为图符的显示窗口指定为当前显示窗口并调用下列函数让它重新显示。

```
glutShowWindow ( );
```

6.4.11 GLUT 子窗口

我们可以在一个选中的显示窗口中建立任意数量的二级显示窗口，称为子窗口。这可用来将显示窗口分成不同的显示区域。创建子窗口的函数是

```
glutCreateSubWindow (windowID, xBottomLeft, yBottomLeft, width, height);
```

参数windowID标识将在其中建立子窗口的显示窗口。其余参数用来指定子窗口左上角相对于该显示窗口左上角的定位。

与第一级显示窗口的编号一样，子窗口用正整数来编号。我们可以将一个子窗口放在另一个子窗口中。每一子窗口可以有独立的显示模式和其他参数。如同对第一级显示窗口一样，我们可以对子窗口进行重定型、重定位、压后、弹出、隐藏和显示。但不能将 GLUT 子窗口变为图符。

6.4.12 显示窗口屏幕光标形状的设定

使用下列函数可以为当前窗口设定屏幕光标的形状。

```
glutSetCursor (shape);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们可以选择的光标形状有按选定方向指向的箭头、双向箭头、旋转箭头、十字游丝（crosshair）、手表、问号，甚至头盖骨图案和交叉腿骨图案。例如，我们可以将符号常量GLUT_CURSOR_UP_DOWN赋给参数 shape 来获得上－下的双向箭头。利用GLUT_CURSOR_CYCLE来选择旋转箭头，使用GLUT_CURSOR _WAIT来选择手表图案，利用常量GLUT_CURSOR_DESTROY来得到头盖骨图案和交叉腿骨图案。赋给显示窗口一个特定的光标形状用来表明一种特定的应用，比如动画。但可用的形状种类依赖于系统。

6.4.13 在GLUT显示窗口中观察图形对象

创建显示窗口并选定其位置、大小、颜色和其他特征后，需要指定在该窗口中显示什么。如果创建了多个显示窗口，则需要先指定哪个是当前显示窗口。然后引用下列函数来指定该窗口显示的内容。

```
glutDisplayFunc (pictureDescrip);
```

其中的变量是一个函数，它指定将什么放在当前窗口中显示。本例中的pictureDescrip函数是一种回调函数（callback function），在GLUT确定应该更新显示窗口内容时执行。pictureDescrip函数可以指定菜单显示等结构组件，但通常包括定义图片的原语和属性。

如果创建了多个显示窗口，则需为每个显示窗口或子窗口重复这一过程。同样，如果显示窗口在重新显示过程中被破坏，则需要在glutPopWindow命令之后调用glutDisplayFunc。此时，使用下列函数来指出当前显示窗口的内容应该更新。

```
glutPostRedisplay ( );
```

在显示窗口给出一弹出式菜单等附加对象时也使用这一函数。

6.4.14 执行应用程序

装入程序完毕、创建和初始化显示窗口后需发布最后一个GLUT命令来启动程序执行：

```
glutMainLoop ( );
```

此时，显示窗口和其中的图形送到屏幕上。程序同时进入GLUT处理循环（GLUT processing loop），反复查询从鼠标或数据板传来的交互输入等新事件。

6.4.15 其他GLUT函数

GLUT函数库提供各种函数用于依赖于系统的处理并为基本OpenGL库增加功能。例如，库中包含了生成位图式和轮廓式字符的函数（参见3.21节），并提供装载颜色表的函数（参见4.3节）。另外，在第8章中将讨论的一些GLUT函数可显示实体或线框方式的三维对象。包括球、圆环和五种规则多面体（立方体、四面体、八面体、十二面体、二十面体）。

在没有其他事件需要系统处理时可以很方便地指定一个函数来运行。调用下列函数：

```
glutIdleFunc (function);
```

在没有任何处理任务时，该函数的参数可以是一个背景函数或更新一个动画参数的过程。

在第11章中还将讨论用于获得和处理交互输入及用于创建和管理菜单的GLUT函数。对输入设备如鼠标、键盘、图形板及空间球等提供了单独的GLUT函数。

最后，我们可以使用下列函数来查询系统某些参数的当前值。

```
glutGet (stateParam);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

该函数返回一个整数值，它对应于为其变量所选的符号常量。例如，可用常量GLUT_WINDOW_X来获得当前显示窗口左上角相对于屏幕左上角的 x 坐标位置。用GLUT_WINDOW_WIDTH或GLUT_SCREEN_WIDTH来获得当前显示窗口宽度和屏幕宽度。

6.4.16 OpenGL的二维观察程序示例

作为OpenGL视口功能的一个演示，我们通过拆分屏幕来给出中心位于世界坐标系原点的 xy 平面中一个三角形的两个视图。先将视口定义在显示窗口的左半区，以蓝色显示原始三角形。然后将视口定义在显示窗口的右半区，用相同的裁剪窗口将三角形显示成红色。接下来将三角形绕其中心旋转。图6.10给出了用该示例程序显示的两个三角形。

```
#include <GL/glut.h>

class wcPt2D {
   public:
      GLfloat x, y;
};

void init (void)
{
   /*  Set color of display window to white.  */
   glClearColor (1.0, 1.0, 1.0, 0.0);

   /*  Set parameters for world-coordinate clipping window.  */
   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (-100.0, 100.0, -100.0, 100.0);

   /*  Set mode for constructing geometric transformation matrix.  */
   glMatrixMode (GL_MODELVIEW);
}

void triangle (wcPt2D *verts)
{
   GLint k;

   glBegin (GL_TRIANGLES);
      for (k = 0; k < 3; k++)
         glVertex2f (verts [k].x, verts [k].y);
   glEnd ( );
}

void displayFcn (void)
{
   /*  Define initial position for triangle.  */
   wcPt2D verts [3] = { {-50.0, -25.0}, {50.0, -25.0}, {0.0, 50.0} };

   glClear (GL_COLOR_BUFFER_BIT);    //  Clear display window.

   glColor3f (0.0, 0.0, 1.0);        //  Set fill color to blue.
   glViewport (0, 0, 300, 300);      //  Set left viewport.
   triangle (verts);                 //  Display triangle.

   /*  Rotate triangle and display in right half of display window.  */
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
   glColor3f (1.0, 0.0, 0.0);          //  Set fill color to red.
   glViewport (300, 0, 300, 300);      //  Set right viewport.
   glRotatef (90.0, 0.0, 0.0, 1.0);    //  Rotate about z axis.
   triangle (verts);              //  Display red rotated triangle.

   glFlush ( );
}
void main (int argc, char ** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (50, 50);
   glutInitWindowSize (600, 300);
   glutCreateWindow ("Split-Screen Example");

   init ( );
   glutDisplayFunc (displayFcn);

   glutMainLoop ( );
}
```

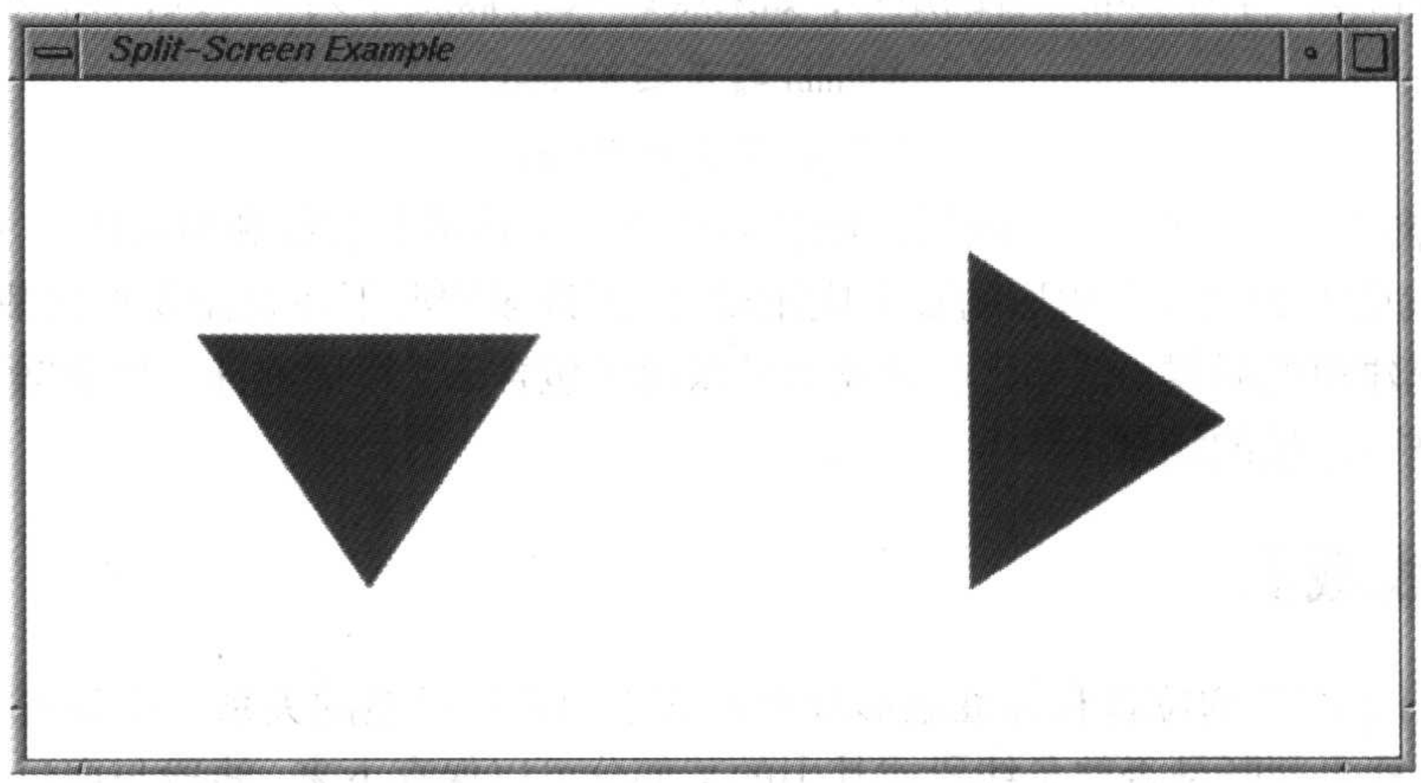


图 6.10 过程 displayFcn 在一个显示窗口中的拆分屏幕效果

6.5 裁剪算法

一般情况下，任何用来消除指定区域内或区域外的图形部分的过程称为**裁剪算法**（clipping algorithm），简称**裁剪**（clipping）。尽管在裁剪应用中可以使用任何形状的裁剪区域，但我们通常使用正则矩形。

裁剪最多应用于观察流水线，目的是为了从场景（二维或三维）中提取指定部分显示在输出设备上。裁剪也用于对象边界的反走样、实体建模法构造对象、管理多窗口环境及在绘画程序中将图的一部分移动、复制或擦除。

在二维观察函数中的裁剪算法用来识别出裁剪窗口中的图形部分。任何位于裁剪窗口外的内容都从将要送到输出设备上显示的场景中消除。对裁剪窗口的规范化边界应用裁剪算法是实现观察流水线裁剪的高效方法。由于可以在裁剪前合并所有的几何和观察变换矩阵并应用于场景描述，因此大大减少了计算量。裁剪后的场景送到屏幕坐标系进行最后的处理。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

下节讨论对下列图元类型的二维裁剪算法：

- 点的裁剪
- 线段的裁剪（直线段）
- 区域的裁剪（多边形）
- 曲线的裁剪
- 文字的裁剪

点、线段和多边形的裁剪是图形软件包中的标准部分。类似的方法除用于样条曲线和曲面外，还用于圆、椭圆等其他二次曲线以及球面。但非线性边界的对象常近似为直线段或多边形表面以便减少计算量。

除非特别声明，我们都假设裁剪区域是一个正则矩形，其边界位于xw_{min}、xw_{max}、yw_{min}、yw_{max}。这些边界与从0到1或从-1到1的规范化正方形的边界对应。

6.6 二维点裁剪

假设裁剪窗口是一个在标准位置的矩形，如果点$\mathbf{P}=(x,y)$满足下列不等式，则保存该点用于显示：

$$\begin{aligned} xw_{\min} \leqslant x \leqslant xw_{\max} \\ yw_{\min} \leqslant y \leqslant yw_{\max} \end{aligned} \tag{6.12}$$

如果这四个不等式中有任何一个不满足，则裁剪掉该点（将不会存储和显示该点）。

虽然点的裁剪不如线或多边形的裁剪应用得多，但许多情况下还是需要点的裁剪过程，特别是当使用特定系统建模的时候。例如，点的裁剪可以用于包含云、海面泡沫、烟或爆炸等用小圆或小球这样的粒子进行建模的场景。

6.7 二维线裁剪

图6.11给出了线段的位置和标准矩形裁剪区域之间各种可能的关系。线裁剪算法通过一系列的测试和求交计算来判断是否整条线段或其中的某部分可以保存下来。线段与窗口边界的交点计算是线裁剪函数的耗时部分。因此，减少交点计算是任一线裁剪算法的主要目标。为此，我们可以先进行测试，确定线段是否完整地在裁剪窗口的内部或完整地位于外部。确定一线段是否完整地在裁剪窗口的内部是很容易的，但要确认整条线段都在窗口外部就比较困难。如果不能确定一线段是否完整地在裁剪窗口的内部或外部，则必须通过进行交点计算来确定是否该线段有一部分落在窗口内部。

我们通过上一节中的点裁剪测试来测试一线段是否完整地落在所指定的裁剪窗口的内部或外部。如果两个端点都在四条裁剪边界内，比如图6.11中$\mathbf{P}_1$到$\mathbf{P}_2$的线段，则该线段完全在裁剪窗口内，就将其存储起来。如果一线段的两个端点都在四条边界中任意一条的外侧（图6.11中的线段$\overline{\mathbf{P}_3\mathbf{P}_4}$），则该线段完全在裁剪窗口的外部，因而应从场景描述中清除。但如果上述两个测试都失败，则线段必定和至少一条边界线相交，也许穿过也许不穿过裁剪窗口。

直线段可使用下列参数公式表示，其中坐标点(x_0, y_0)和(x_{end}, y_{end})给出线段的两个端点。

$$\begin{aligned} x &= x_0 + u(x_{\text{end}} - x_0) \\ y &= y_0 + u(y_{\text{end}} - y_0) \qquad 0 \leqslant u \leqslant 1 \end{aligned} \tag{6.13}$$

通过将某一边界赋值给x或y，解出u值，我们便可确定线段与每一裁剪窗口边界的相交位置。比如，当窗口左边界位于xw_{min}时，将其代入x并解出u，即可求出交点的y值。如果u值在0到1之

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

外，则线段与窗口边界不相交。但如果 u 值在 0 到 1 之内，就有部分线段位于该边界之内。我们再对位于内部的线段部分使用另一边界进行处理，直到线段不再有边界内的部分或找到窗口内的部分。

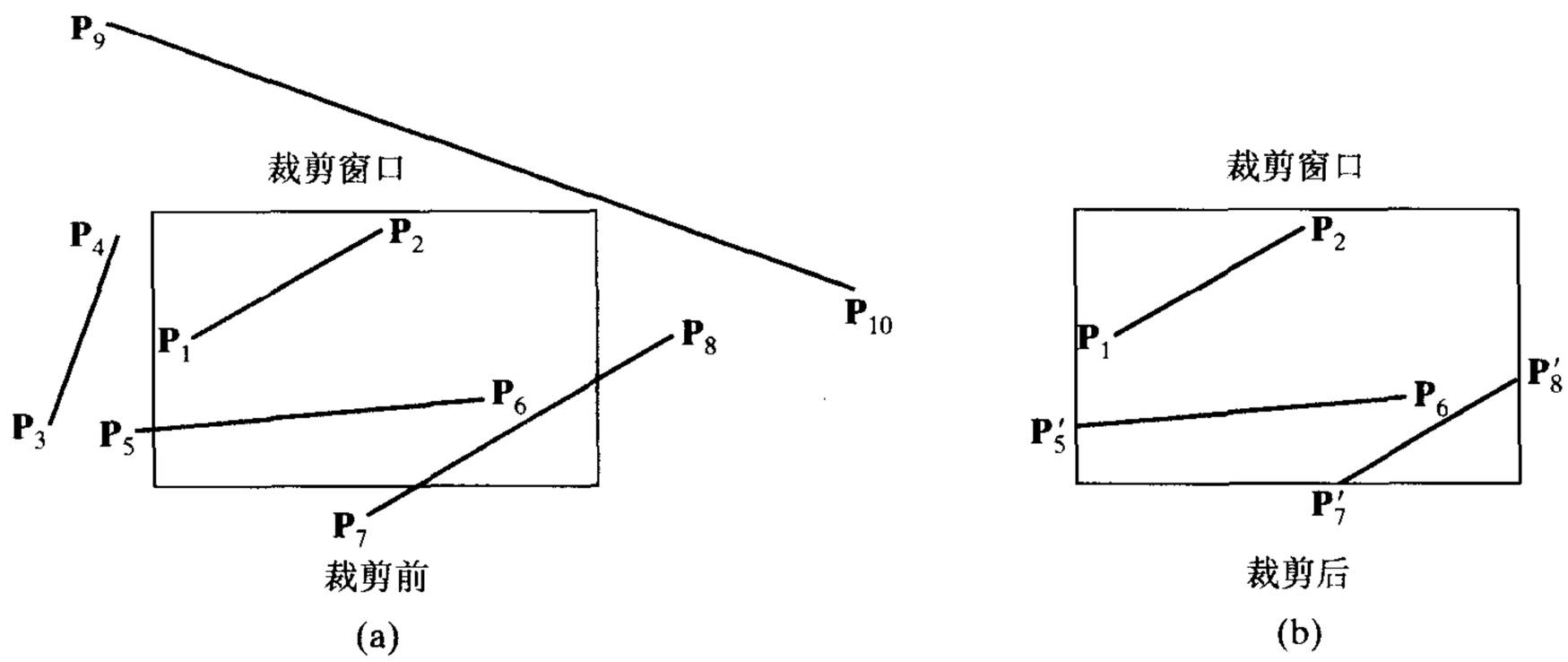


图 6.11　使用标准矩形窗口的线段裁剪

使用前面的简单裁剪方法对场景中的线段进行处理比较直接但效率不高。有可能重新组织初始测试和求交计算来减少一组线段的处理时间，并且已经开发出一些快速裁剪算法。有些算法是针对二维图形的，有些算法可以很容易地移植到三维应用中。

6.7.1　Cohen-Sutherland 线段裁剪算法

这是一个最早开发的快速线段裁剪算法，已经得到了广泛的使用。该算法通过初始测试来减少交点计算，从而减少线段裁剪算法所用的时间。每条线段的端点都赋以称为**区域码**（region code）的四位二进制码，每一位用来标识端点相对于相应裁剪矩形边界的里面还是外面。我们可以按任意次序引用窗口边界，图 6.12 给出了从右到左编号、从 1 到 4 的一种顺序。在这个顺序下，最右边位置（位 1）对应裁剪窗口的左边界，最左边位置（位 4）对应窗口的上边界。任何码位的值为 1（真）表示端点在相应窗口边界的外面。类似地，码位的值为 0（假）表示端点不在相应窗口边界的外面（在内部或在边界上）。有时，区域码称为**“外部”码**（out code），因为任何码位的值为 1（真）表示点在相应窗口边界的外面。

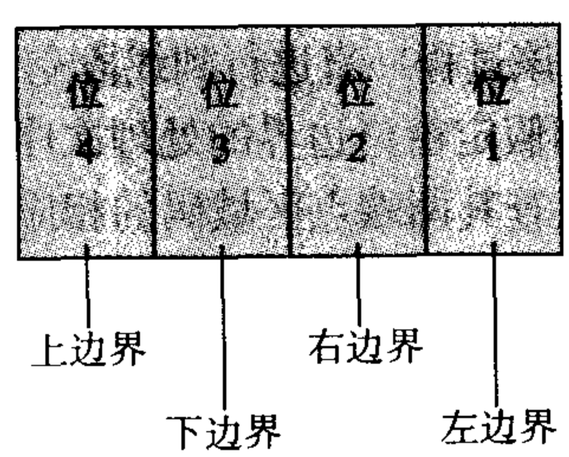


图 6.12　Cohen-Sutherland 端点区域码各位与相应裁剪窗口边界的一种顺序

每一裁剪窗口边界将二维空间划分成内部和外部的两个半空间。四个窗口边界一起生成了九个区域，图 6.13 列出了这些区域的二进码。因此，在裁剪窗口左下角的端点赋以区域码 0101 而任一在裁剪窗口内部的点的区域码是 0000。

区域码的位值通过将端点的坐标值(x, y)与裁剪窗口边界相比较而确定。如果 $x < xw_{min}$ 则位置 1 为 1，其他各位的值与此类似。除了使用不等式测试，我们还可以使用位处理操作和下列两步操

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

作来更高效地确定区域码的值：(1)计算端点坐标与裁剪边界的差。(2)用各差值计算的符号位来设置区域码中相应的值。按图 6.12 中的顺序，位 1 设为 $x - xw_{min}$ 的符号位；位 2 设为 $xw_{max} - x$ 的符号位；位 3 设为 $y - yw_{min}$ 的符号位；位 4 设为 $yw_{max} - y$ 的符号位。

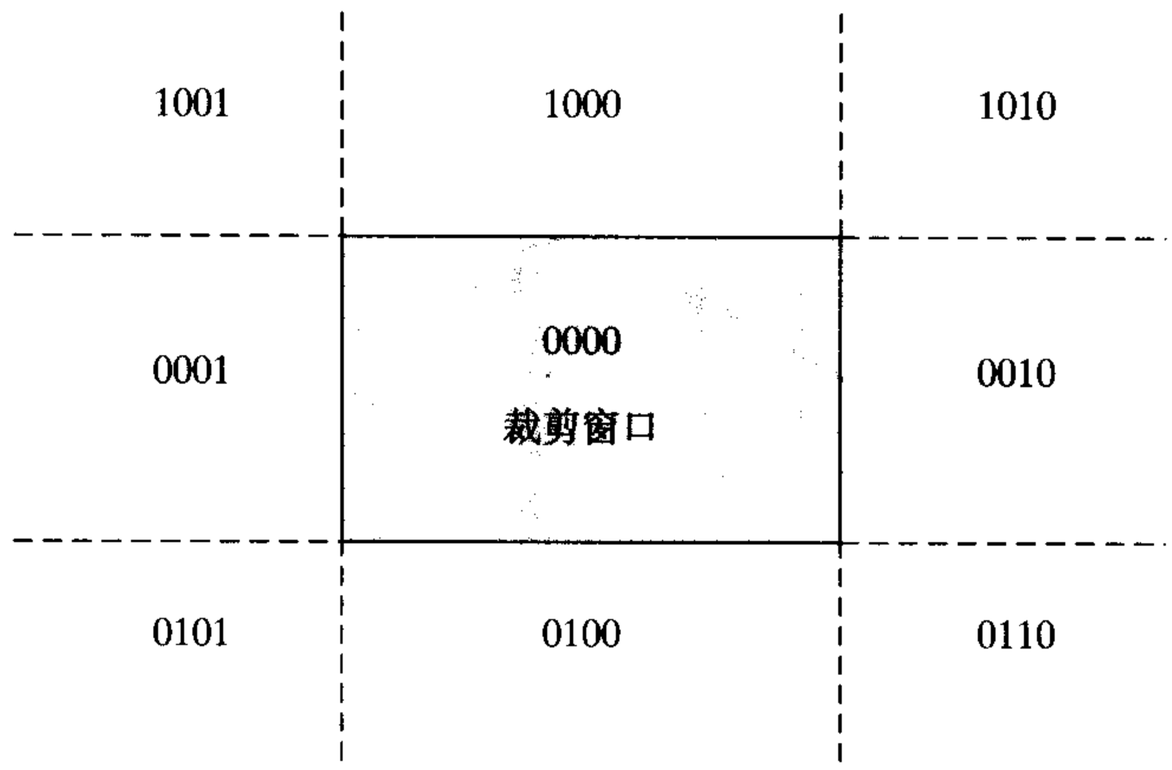


图 6.13　标识线段端点相对于裁剪窗口位置的九个二进区域码

一旦给所有的线段端点建立了区域码，就可以快速判断哪条线段完全在裁剪窗口之内，哪条线段完全在窗口之外。完全在窗口边界内的线段，其两个端点的区域码均为0000，因此保留了这些线段。两个端点的区域码中，有一对相同位置都为 1 的线段则完全落在裁剪矩形之外，因此丢弃这些线段。例如，线段的一个端点的区域码为 1001，而另一端点的区域码为0101，则丢弃这条线段，因为这条线段的两个端点都在裁剪矩形的左边，端点区域码的第一位都为 1。

测试线段是否在内部或外部的方法是对两个端点的区域码进行逻辑操作。如果两个端点的区域码进行逻辑或的结果为0000，则线段完全位于裁剪区域之内。因此保存该线段并测试场景描述中的下一条线。如果两个端点的区域码进行逻辑与操作的结果是真（不为0000），则线段完全位于裁剪区域之外，我们可以将该线从场景中清除。

对于不能判断为完全在窗口外或窗口内的线段，则要测试其与窗口边界的交点。如图 6.14 所示，这些线段可能穿过或不穿过窗口内部。因此，可能要进行多次求交运算才能完成一条线段的裁剪，求交次数依赖于选择裁剪边界的次序。每次处理一条裁剪窗口边界后，裁掉其中一部分，余下部分对照窗口的其余边界进行检查。该过程一直进行到线段完全被裁剪掉或余下的线段部分完全在裁剪窗口内。在以下的讨论中，我们假定窗口边界的处理次序是：左、右、下、上。要检查一线段是否与某裁剪边界相交，我们可以检查其两端点区域码的相应位。如果其中一个是1而另一个是0，则线段与该边界相交。

图6.14给出了两条不能马上判断出完全在窗口内或窗口外的线段。$\mathbf{P}_1$到$\mathbf{P}_2$的线段区域码是0100和 1001。因此，$\mathbf{P}_1$ 在左边界之内而 $\mathbf{P}_2$ 在左边界之外。接着我们计算交点位置 $\mathbf{P}_2'$ 并裁掉 $\mathbf{P}_2$ 到 $\mathbf{P}_2'$ 的部分。余下的线段部分位于右边界的内部，因此我们接下来检查下边界。端点 $\mathbf{P}_1$ 在下裁剪边界之下而 $\mathbf{P}_2'$ 在其上，因此求出在该边界上的交点（$\mathbf{P}_1'$）。我们清除从 $\mathbf{P}_1$ 到 $\mathbf{P}_1'$ 的部分，再处理窗口的上边界。在其上我们确定交点位置 $\mathbf{P}_2''$。最后一步是裁掉上边界之上的部分并保存从 $\mathbf{P}_1'$ 到 $\mathbf{P}_2''$ 的内部段。对于第二条线段，我们发现 $\mathbf{P}_3$ 在左边界的外部而 $\mathbf{P}_4$ 在左边界的内部。因此我们计算交点 $\mathbf{P}_3'$ 并清除 $\mathbf{P}_3$ 到 $\mathbf{P}_3'$ 的段。通过对端点 $\mathbf{P}_3'$ 和 $\mathbf{P}_4$ 区域码的测试，我们发现余下的线段部分在裁剪窗口之下而将其清除。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

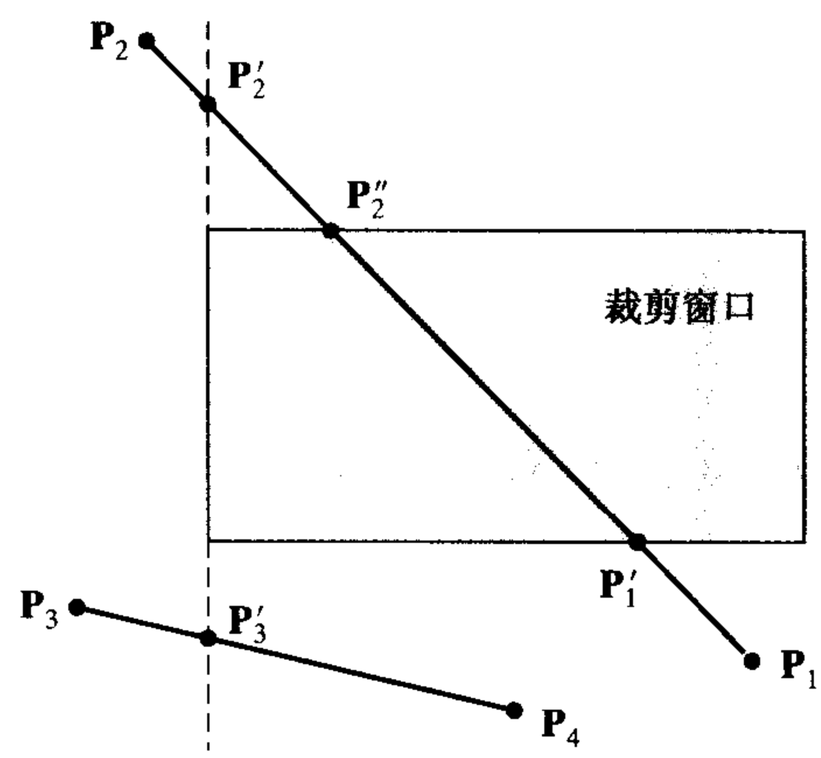


图 6.14　从裁剪窗口的一个区域到另一个区域的线段穿过裁剪窗口或者不穿过窗口但与边界相交

使用这一方法裁剪线段时，很可能要计算与所有四条裁剪边界的交点，这依赖于线段端点如何处理以及按什么样的边界次序。图6.15给出了一线段与裁剪窗口边界按左、右、下、上次序计算所得的四个交点。因此，为了减少求交计算，人们对该算法进行了许多改进。

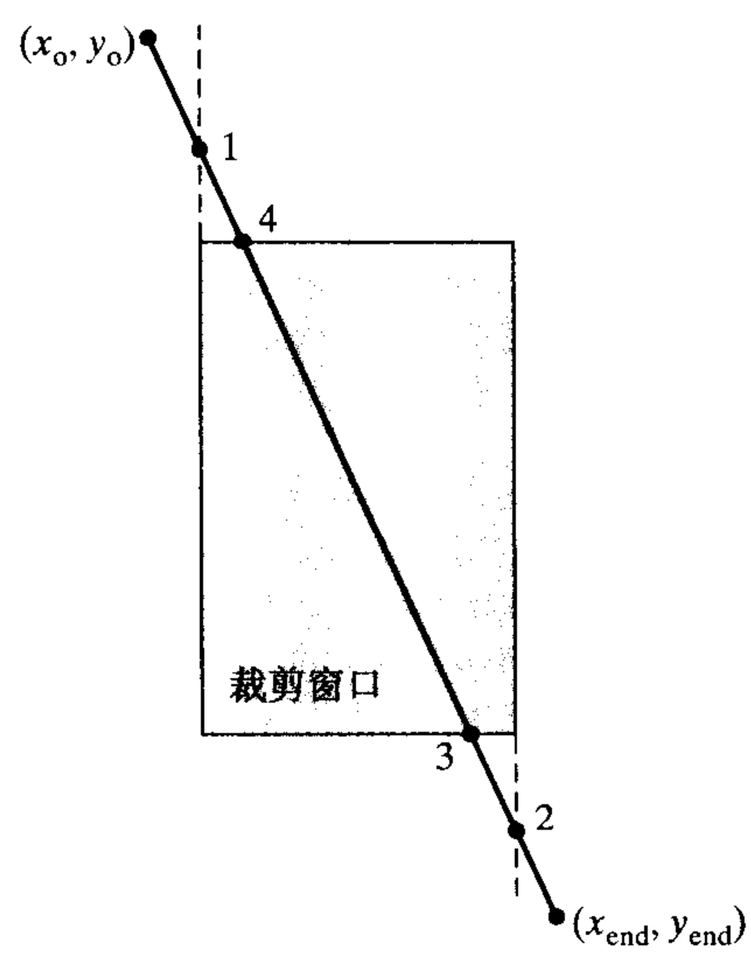


图 6.15　按左、右、下、上次序计算出的线段与窗口边界的四个交点（从 1 到 4 编号）

线段与裁剪边界的交点计算可以使用斜率截距式的直线方程。对于端点坐标为(x_0, y_0)和(x_{end}, y_{end})的直线段，与垂直边界交点的 y 坐标可以由下列等式计算得到：

$$y = y_0 + m(x - x_0) \tag{6.14}$$

其中，x 值置为 xw_{min} 或 xw_{max}，线段的斜率根据 $m = (y_{end} - y_0)/(x_{end} - x_0)$进行计算。同样，我们要寻找与水平边界相交的交点，其 x 坐标可以按下列等式进行计算：

$$x = x_0 + \frac{y - y_0}{m} \tag{6.15}$$

其中，y 设为 yw_{min} 或 yw_{max}。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

二维Cohen-Sutherland线裁剪算法在下面的函数中给出。该算法到三维的扩充是很直接的，我们下一章将讨论三维的观察方法。

```
class wcPt2D {
   public:
      GLfloat x, y;
};

inline GLint round (const GLfloat a)  { return GLint (a + 0.5); }

/*  Define a four-bit code for each of the outside regions of a
 *  rectangular clipping window.
 */
const GLint winLeftBitCode = 0x1;
const GLint winRightBitCode = 0x2;
const GLint winBottomBitCode = 0x4;
const GLint winTopBitCode = 0x8;

/*  A bit-mask region code is also assigned to each endpoint of an input
 *  line segment, according to its position relative to the four edges of
 *  an input rectangular clip window.
 *
 *  An endpoint with a region-code value of 0000 is inside the clipping
 *  window, otherwise it is outside at least one clipping boundary.  If
 *  the 'or' operation for the two endpoint codes produces a value of
 *  false, the entire line defined by these two endpoints is saved
 *  (accepted).  If the 'and' operation between two endpoint codes is
 *  true, the line is completely outside the clipping window, and it is
 *  eliminated (rejected) from further processing.
 */
inline GLint inside (GLint code) { return GLint (!code); }
inline GLint reject (GLint code1, GLint code2)
                        { return GLint (code1 & code2); }
inline GLint accept (GLint code1, GLint code2)
                        { return GLint (!(code1 | code2)); }

GLubyte encode (wcPt2D pt, wcPt2D winMin, wcPt2D winMax)
{
  GLubyte code = 0x00;

  if (pt.x < winMin.x)
    code = code | winLeftBitCode;
  if (pt.x > winMax.x)
    code = code | winRightBitCode;
  if (pt.y < winMin.y)
    code = code | winBottomBitCode;
  if (pt.y > winMax.y)
    code = code | winTopBitCode;
  return (code);
}

void swapPts (wcPt2D * p1, wcPt2D * p2)
{
  wcPt2D tmp;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
  tmp = *p1; *p1 = *p2; *p2 = tmp;
}

void swapCodes (GLubyte * c1, GLubyte * c2)
{
  GLubyte tmp;

  tmp = *c1; *c1 = *c2; *c2 = tmp;
}


void lineClipCohSuth (wcPt2D winMin, wcPt2D winMax, wcPt2D p1, wcPt2D p2)
{
  GLubyte code1, code2;
  GLint done = false, plotLine = false;
  GLfloat m;

  while (!done) {
    code1 = encode (p1, winMin, winMax);
    code2 = encode (p2, winMin, winMax);
    if (accept (code1, code2)) {
      done = true;
      plotLine = true;
    }
    else
      if (reject (code1, code2))
        done = true;
      else {
        /*  Label the endpoint outside the display window as p1. */
        if (inside (code1)) {
          swapPts (&p1, &p2);
          swapCodes (&code1, &code2);
        }
        /*  Use slope m to find line-clipEdge intersection.  */
        if (p2.x != p1.x)
          m = (p2.y - p1.y) / (p2.x - p1.x);
        if (code1 & winLeftBitCode) {
          p1.y += (winMin.x - p1.x) * m;
          p1.x = winMin.x;
        }
        else
          if (code1 & winRightBitCode) {
            p1.y += (winMax.x - p1.x) * m;
            p1.x = winMax.x;
          }
          else
            if (code1 & winBottomBitCode) {
              /*  Need to update p1.x for nonvertical lines only.  */
              if (p2.x != p1.x)
                p1.x += (winMin.y - p1.y) / m;
              p1.y = winMin.y;
            }
            else
              if (code1 & winTopBitCode) {
                if (p2.x != p1.x)
                  p1.x += (winMax.y - p1.y) / m;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
                p1.y = winMax.y;
              }
          }
      }
      if (plotLine)
        lineBres (round (p1.x), round (p1.y), round (p2.x), round (p2.y));
    }
```

6.7.2 梁友栋-Barsky线段裁剪算法

已开发的快速线段裁剪算法都在计算交点之前进行了更多的测试。Cyrus和Beck提出的算法是这方面最早的努力之一，它基于分析线段的参数化方程。后来，梁友栋和Barsky分别提出了参数线裁剪的更快算法。

对端点为(x_0, y_0)和(x_{end}, y_{end})的直线段，可以使用参数形式描述直线段：

$$\begin{aligned} x &= x_0 + u\Delta x \\ y &= y_0 + u\Delta y \qquad 0 \leqslant u \leqslant 1 \end{aligned} \tag{6.16}$$

其中，$\Delta x = x_{end} - x_0$、$\Delta y = y_{end} - y_0$。在梁友栋-Barsky算法中，线段的参数方程与点裁剪条件（6.12）结合起来，获得不等式：

$$\begin{aligned} xw_{min} &\leqslant x_0 + u\Delta x \leqslant xw_{max} \\ yw_{min} &\leqslant y_0 + u\Delta y \leqslant yw_{max} \end{aligned} \tag{6.17}$$

这四个不等式可以表示为

$$u\ p_k \leqslant q_k, \qquad k = 1, 2, 3, 4 \tag{6.18}$$

其中，参数p、q定义为

$$\begin{aligned} p_1 &= -\Delta x, & q_1 &= x_0 - xw_{min} \\ p_2 &= \Delta x, & q_2 &= xw_{max} - x_0 \\ p_3 &= -\Delta y, & q_3 &= y_0 - yw_{min} \\ p_4 &= \Delta y, & q_4 &= yw_{max} - y_0 \end{aligned} \tag{6.19}$$

任何平行于裁剪边界之一的直线$p_k = 0$，其中k对应于该裁剪边界（$k = 1, 2, 3, 4$对应于左、右、下、上边界）。如果还满足$q_k < 0$，则线段完全在边界以外，因此舍弃该线段。如果$q_k \geqslant 0$，则线段位于平行边界内。

当$p_k < 0$时，线段从裁剪边界延长线的外部延伸到内部。当$p_k > 0$，线段从裁剪边界的内部延伸到外部。当$p_k \neq 0$时，可以计算出线段与边界k的延长线的交点u值：

$$u = \frac{q_k}{p_k} \tag{6.20}$$

对于每条直线，可以计算出参数u_1和u_2，它们定义了裁剪矩形内的线段部分。u_1的值由线段从外到内遇到的矩形边界所决定（$p < 0$）。对于这些边界，计算$r_k = q_k/p_k$。u_1取0和各个r值中的最大值。u_2的值则由线段从内到外遇到的矩形边界所决定（$p > 0$）。根据这些边界计算出r_k，u_2取1和各个r值之中的最小值。如果$u_1 > u_2$，则线段完全落在裁剪窗口之外，因此将舍弃该线段。否则，由参数u的两个值计算出裁剪后的线段端点。

我们使用下面的程序描述该算法。线段交点的参数初始化为$u_1 = 0$、$u_2 = 1$。计算出各个裁剪边界的p、q值，函数clipTest根据p、q来判断是舍弃线段还是改变交点的参数。当$p < 0$时，参数r用于更新u_1；当$p > 0$时，参数r用于更新u_2。如果更新了u_1或u_2后使$u_1 > u_2$，则舍弃该线段。否

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

则，更新适当的 u 参数，使新值仅仅缩短了线段。当 $p=0$ 且 $q<0$ 时，舍弃该线段，因为该线段平行于边界并且位于边界之外。如果测试完 p、q 的四个值之后，结果并未舍弃该线段，则由 u_1、u_2 值决定裁剪的线段的端点。

```
class wcPt2D
  {
     private:
        GLfloat x, y;

     public:
     /*  Default Constructor: initialize position as (0.0, 0.0).  */
     wcPt3D ( ) {
        x = y = 0.0;
     }

     setCoords (GLfloat xCoord, GLfloat yCoord) {
        x = xCoord;
        y = yCoord;
     }

     GLfloat getx ( ) const {
        return x;
     }

     GLfloat gety ( ) const {
        return y;
     }
  };

inline GLint round (const GLfloat a)  { return GLint (a + 0.5); }

GLint clipTest (GLfloat p, GLfloat q, GLfloat * u1, GLfloat * u2)
{
  GLfloat r;
  GLint returnValue = true;

  if (p < 0.0) {
    r = q / p;
    if (r > *u2)
      returnValue = false;
    else
      if (r > *u1)
        *u1 = r;
  }
  else
    if (p > 0.0) {
      r = q / p;
      if (r < *u1)
        returnValue = false;
      else if (r < *u2)
        *u2 = r;
    }
    else
      /*  Thus p = 0 and line is parallel to clipping boundary.  */
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        if (q < 0.0)
          /*  Line is outside clipping boundary.  */
          returnValue = false;

    return (returnValue);
  }

  void lineClipLiangBarsk (wcPt2D winMin, wcPt2D winMax, wcPt2D p1, wcPt2D p2)
  {
    GLfloat u1 = 0.0, u2 = 1.0, dx = p2.getx ( ) - p1.getx ( ), dy;

    if (clipTest (-dx, p1.getx ( ) - winMin.getx ( ), &u1, &u2))
      if (clipTest (dx, winMax.getx ( ) - p1.getx ( ), &u1, &u2)) {
        dy = p2.gety ( ) - p1.gety ( );
        if (clipTest (-dy, p1.gety ( ) - winMin.gety ( ), &u1, &u2))
          if (clipTest (dy, winMax.gety ( ) - p1.gety ( ), &u1, &u2)) {
            if (u2 < 1.0) {
              p2.setCoords (p1.getx ( ) + u2 * dx, p1.gety ( ) + u2 * dy);
            }
            if (u1 > 0.0) {
              p1.setCoords (p1.getx ( ) + u1 * dx, p1.gety ( ) + u1 * dy);
            }
            lineBres (round (p1.getx ( )), round (p1.gety ( )),
                      round (p2.getx ( )), round (p2.gety ( )));
          }
      }
  }
```

通常，梁友栋-Barsky算法比Cohen-Sutherland算法更有效。更新参数u_1、u_2仅仅需要一次除法；计算出u_1、u_2的最后值后，线段与窗口的交点只计算一次。相比之下，即使一条线段完全落在裁剪窗口之外，Cohen-Sutherland算法也要对其反复求交点，而且每次求交计算都需要除法和乘法运算。二维梁友栋-Barsky算法可以扩展为三维裁剪算法（参见第7章）。

6.7.3 Nicholl-Lee-Nicholl 线段裁剪算法

Nicholl-Lee-Nicholl（NLN）算法通过在裁剪窗口边界创立多个区域，从而避免对一个直线段进行多次裁剪。在Cohen-Sutherland算法中，在找到与裁剪矩形边界的交点之前或者完全舍弃该线段之前，必须对一条线段进行多次求交计算。NLN算法则在求交计算前进行更多的区域测试，从而减少求交计算。与Cohen-Sutherland和梁友栋-Barsky算法相比，NLN算法的比较次数和除法次数减少。但是NLN算法仅仅用于二维裁剪，而梁友栋-Barsky和Cohen-Sutherland算法可以很方便地扩展为三维裁剪算法。

确定一线段完全在裁剪窗口内部或外部的初始测试可以像前面两个算法一样用区域码测试来完成。如果一线段不能明确接受或拒绝，则NLN算法进一步建立另外的裁剪区域。

对于端点为$\mathbf{P}_0$和$\mathbf{P}_{end}$的线段，首先确定$\mathbf{P}_0$相对于裁剪矩形的九个可能区域的位置。在图6.16中只要考虑三个区域。如果$\mathbf{P}_0$位于其他六个区域中的任何一个位置，则可以利用对称变换将其变换到图6.16中三个区域中的一个。例如，在裁剪窗口正上方的区域，可以相对于直线$y = -x$投影到裁剪窗口的左边区域；或者使用90°逆时针旋转，也可以变换到裁剪窗口的左边区域。

假定$\mathbf{P}_0$和$\mathbf{P}_{end}$不同时在裁剪窗口的内部，下一步判断$\mathbf{P}_{end}$相对于$\mathbf{P}_0$的位置。为此，根据$\mathbf{P}_0$的位置在平面上创立新的区域。新区域的边界是以$\mathbf{P}_0$为起始点、穿过窗口的顶点的射线。如果$\mathbf{P}_0$在

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

裁剪窗口之内，$\mathbf{P}_{end}$在窗口之外，我们就设置四个区域，如图6.17所示。根据包含$\mathbf{P}_{end}$点的某一个区域（L、T、R和B），可以得到线段与窗口边界的交点。

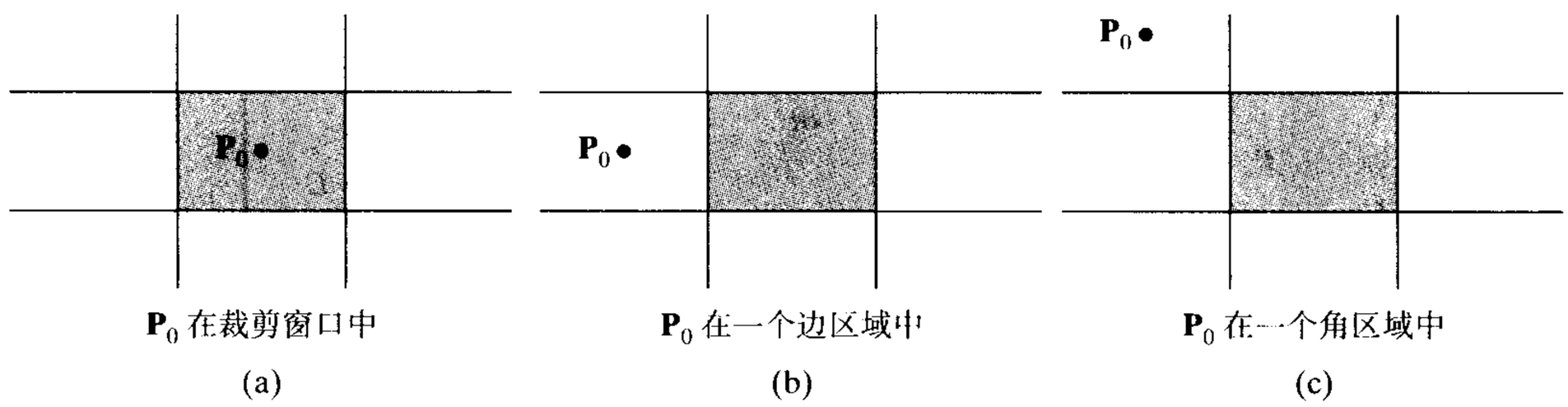


图6.16 在NLN线段裁剪算法中，线段端点$\mathbf{P}_0$的三种位置

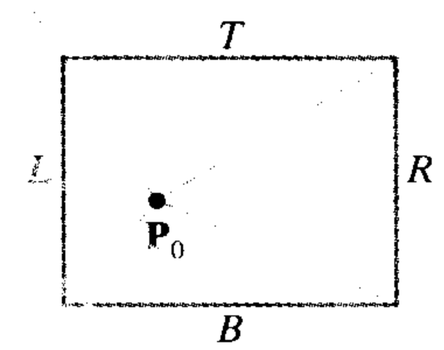


图6.17 当$\mathbf{P}_0$在裁剪窗口内而$\mathbf{P}_{end}$在窗口外时，采用NLN算法的四个裁剪区域

如果$\mathbf{P}_0$位于窗口的左边区域，则设定四个区域：L、LT、LR和LB，如图6.18所示。这四个区域决定了线段的惟一边界。如$\mathbf{P}_{end}$在L区域，我们在左边界裁剪该线段，且保存从交点到$\mathbf{P}_{end}$之间的线段。如果$\mathbf{P}_{end}$在LT区域，则存储窗口左边界到上边界之间的线段部分。对区域LR、LB的处理相类似。如果$\mathbf{P}_{end}$不在四个区域（L、LT、LR或LB）的任何一个之内，则舍弃整个线段。

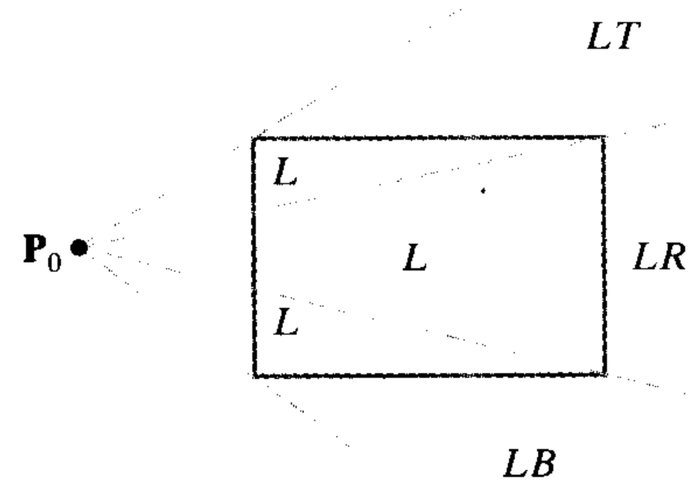


图6.18 当$\mathbf{P}_0$在裁剪窗口的正左边时，NLN算法中的四个裁剪区域

对于第三种情况，当$\mathbf{P}_0$在裁剪窗口的左上方时，我们采用图6.19所示的裁剪区域。在这种情况下，根据$\mathbf{P}_0$相对于窗口左上角的位置，有两种可能。如果$\mathbf{P}_0$接近裁剪窗口左边界，我们使用图中(a)的区域。否则，如果$\mathbf{P}_0$接近裁剪窗口上边界，则使用(b)中区域。如果$\mathbf{P}_{end}$在区域T、L、TR、TB、LR或LB中，则确定了求交计算的惟一裁剪窗口边界，否则舍弃整个线段。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

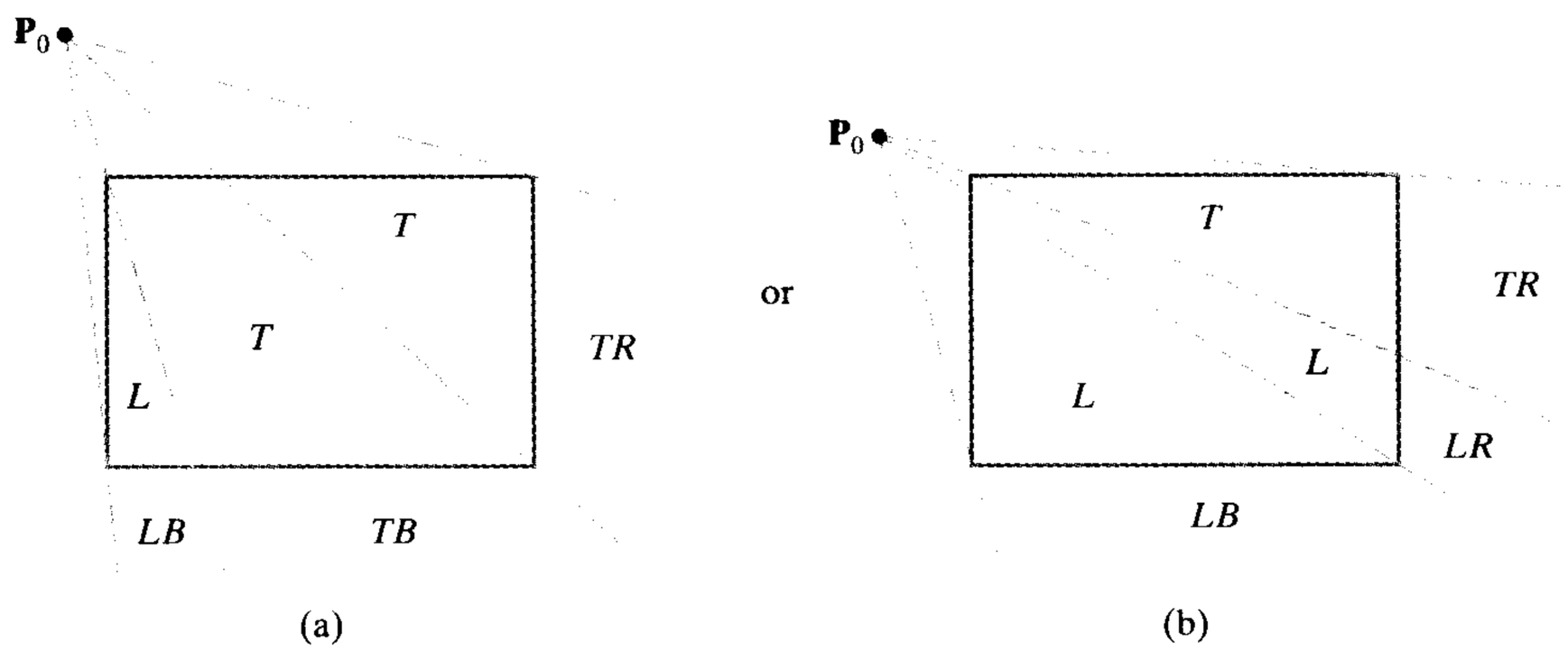


图 6.19 当 $\mathbf{P}_0$ 在裁剪窗口的左上方时，NLN 算法中的两组裁剪区域

为了确定 $\mathbf{P}_{end}$ 位于哪一个区域，要比较该线段的斜率和裁剪区域边界的斜率。例如，如果 $\mathbf{P}_0$ 在裁剪边界的左边（参见图 6.18），并且满足下列条件，则 $\mathbf{P}_{end}$ 在区域 LT 中：

$$\text{斜率}\overline{\mathbf{P}_0\mathbf{P}_{TR}} < \text{斜率}\overline{\mathbf{P}_0\mathbf{P}_{end}} < \text{斜率}\overline{\mathbf{P}_0\mathbf{P}_{TL}} \tag{6.21}$$

或

$$\frac{y_T - y_0}{x_R - x_0} < \frac{y_{end} - y_0}{x_{end} - x_0} < \frac{y_T - y_0}{x_L - x_0} \tag{6.22}$$

如果满足下列条件则舍弃整条直线：

$$(y_T - y_0)(x_{end} - x_0) < (x_L - x_0)(y_{end} - y_0) \tag{6.23}$$

在斜率测试中的坐标差值和计算结果将被存储，可用于以后进行求交计算。参数方程式

$$x = x_0 + (x_{end} - x_0)u$$
$$y = y_0 + (y_{end} - y_0)u$$

与窗口左边界的交点 x 位置是 $x = x_L$，而且 $u = (x_L - x_0)/(x_{end} - x_0)$，所以交点的 y 位置是

$$y = y_0 + \frac{y_{end} - y_0}{x_{end} - x_0}(x_L - x_0) \tag{6.24}$$

并且与窗口顶部边界的交点是 $y = y_T$，而且 $u = (y_T - y_0)/(y_{end} - y_0)$，因此

$$x = x_0 + \frac{x_{end} - x_0}{y_{end} - y_0}(y_T - y_0) \tag{6.25}$$

6.7.4 非矩形裁剪窗口的线段裁剪

在某些应用中，需要使用任意形状的多边形对线段进行裁剪。基于参数化直线方程的算法，如 Cyrus-Beck 算法或梁友栋-Barsky 算法，都可以扩充到凸多边形窗口。只需修改算法使参数化方程适合裁剪区域的边界。根据裁剪多边形的坐标范围来处理线段，从而完成线段的屏幕显示。

对于凹多边形，可以在使用 3.15 节所述方法将它分解为一组凸多边形后再使用参数化裁剪算法。另外一种方法是添加一些边使凹裁剪区域成为凸裁剪区域。然后使用修改后的凸多边形组对线段进行一系列裁剪操作，如图 6.20 所示。图 6.20(a)中线段 $\overline{\mathbf{P}_1\mathbf{P}_2}$ 要用由顶点 $\mathbf{V}_1$、$\mathbf{V}_2$、$\mathbf{V}_3$、$\mathbf{V}_4$、$\mathbf{V}_5$ 确

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

定的凹窗口裁剪。此时，从 $\mathbf{V}_4$ 到 $\mathbf{V}_1$ 添加一条线段，就得到两个凸裁剪区域。然后线段经过两轮裁剪：(1)线段 $\overline{\mathbf{P}_1\mathbf{P}_2}$ 用由 $\mathbf{V}_1$、$\mathbf{V}_2$、$\mathbf{V}_3$、$\mathbf{V}_4$ 确定的凸多边形裁剪，得到线段 $\overline{\mathbf{P}_1'\mathbf{P}_2'}$（参见图 6.20(b)）；(2)线段 $\overline{\mathbf{P}_1'\mathbf{P}_2'}$ 用由 $\mathbf{V}_1$、$\mathbf{V}_5$、$\mathbf{V}_4$ 确定的凸多边形（参见图 6.20(c)）裁剪，得到最终裁剪后的线段 $\overline{\mathbf{P}_1''\mathbf{P}_2'}$。

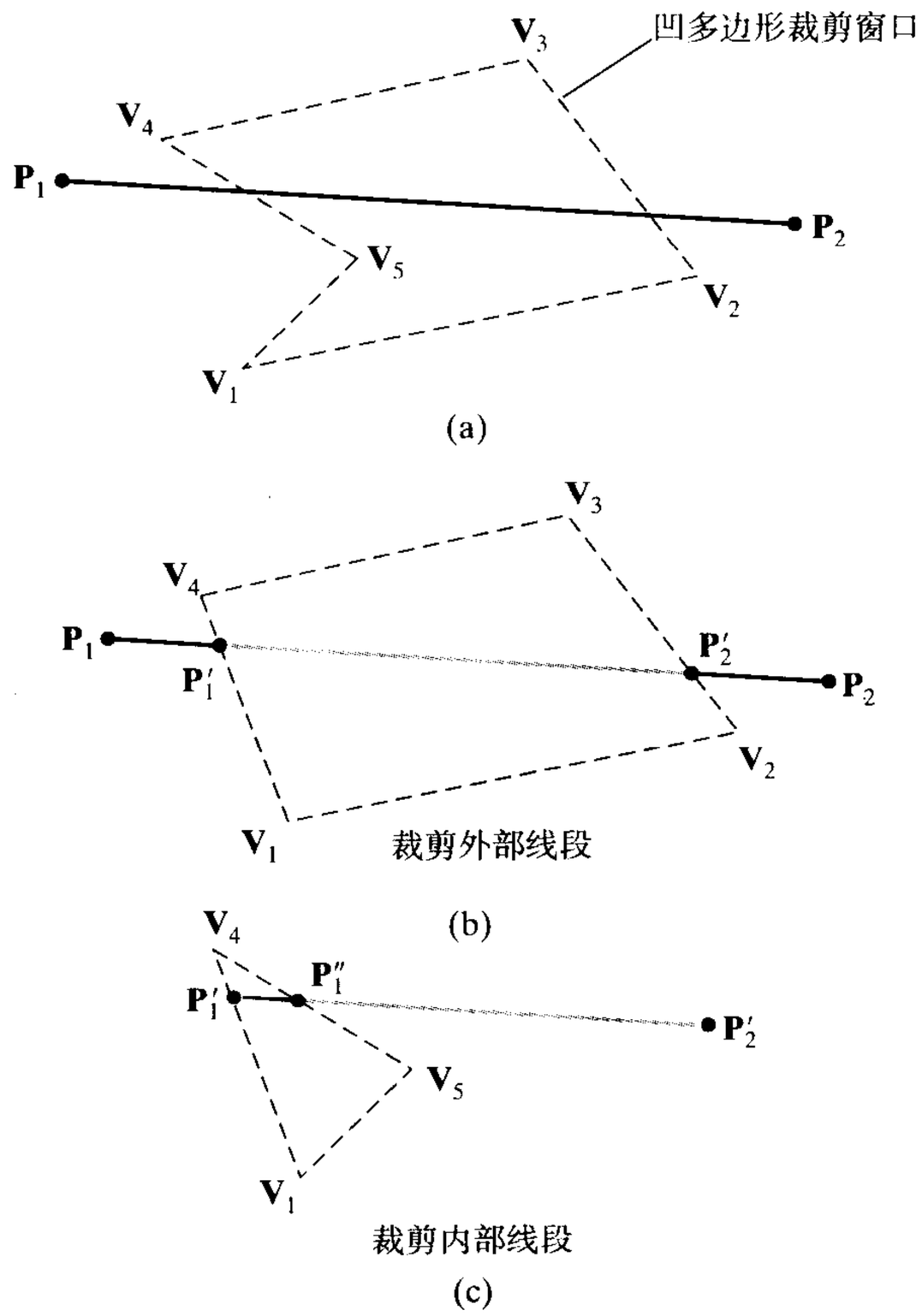


图 6.20　一个凹多边形裁剪窗口(a)有 5 个顶点（$\mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{V}_4, \mathbf{V}_5$），修改成(b)中 4 顶点（$\mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{V}_4$）的凸多边形。线段 $\overline{\mathbf{P}_1\mathbf{P}_2}$ 的外部段用该凸多边形切断。留下的线段 $\overline{\mathbf{P}_1'\mathbf{P}_2'}$ 接着用三角形（$\mathbf{V}_1, \mathbf{V}_5, \mathbf{V}_4$）处理。(c)裁掉 $\overline{\mathbf{P}_1'\mathbf{P}_1''}$ 生成最后的裁剪后线段 $\overline{\mathbf{P}_1''\mathbf{P}_2'}$

6.7.5　非线性裁剪窗口边界的线裁剪

也可以使用圆或其他曲线边界进行裁剪，使用这些区域的裁剪算法的速度较慢，因为该算法的求交计算涉及非线性曲线方程。首先要由曲线裁剪区域的包围矩形（坐标范围）对线段进行裁剪。完全落在包围矩形之外的线段将被舍弃。我们可以通过计算圆心到线段端点的距离，以识别出内部线段。如果线段的两端点到圆心距离的平方小于等于半径的平方，则存储整条线段。其他线段则通过求解圆－直线联立方程来计算交点。

6.8　多边形填充区裁剪

图形软件包通常会提供对填充凸多边形的裁剪功能。为了裁剪一个填充多边形，不能直接使用线裁剪算法对多边形的每一条边进行裁剪，因为该方法一般不能生成封闭的折线。相反，使用线段裁剪进行处理的多边形边界将显示为一系列不连接的线段，并且没有关于如何形成裁剪后的封闭边界的完整信息。图6.21给出了将线裁剪过程施加于一多边形填充区域的可能输出。我们需要的是输

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

出一个或多个裁剪后的填充区边界的封闭多边形，继而这样的多边形能被扫描转换成如图6.22所示用指定颜色或图案填充的图形。

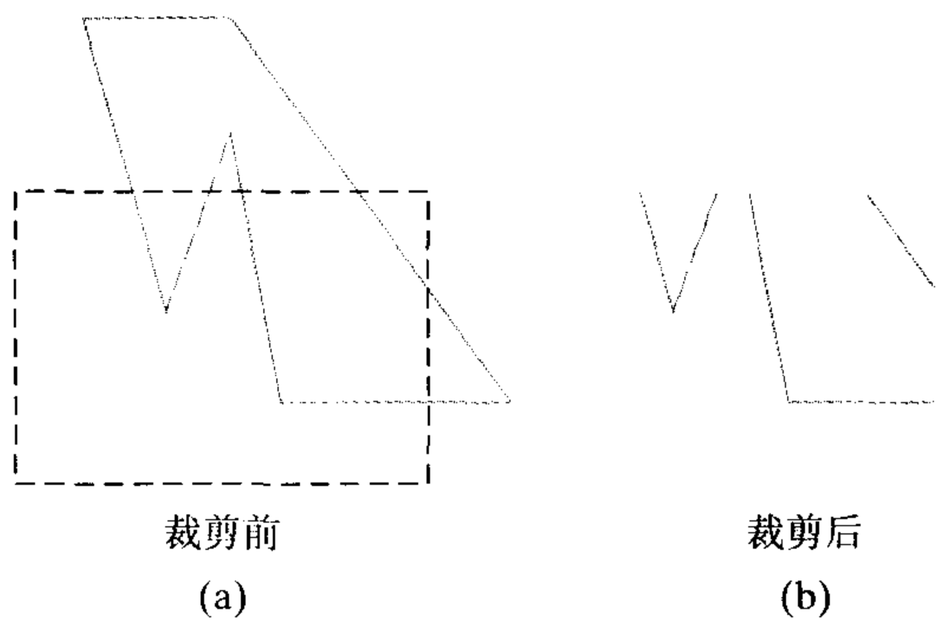


图 6.21　用线段裁剪算法处理多边形(a)生成未连接的线段集(b)

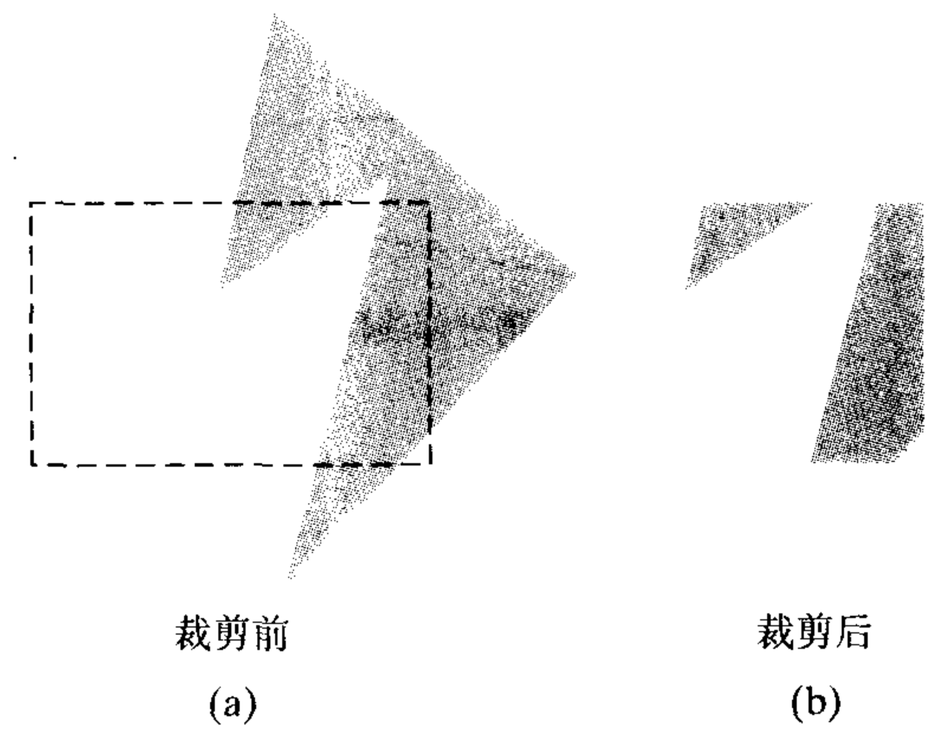


图 6.22　显示正确裁剪后的多边形填充区

我们可以使用与线裁剪一样的方法，通过裁剪窗口边界对多边形填充区裁剪。一条线段由两端点定义，这些端点经线裁剪函数处理获得一组位于裁剪边界上的端点。同样我们需要保持填充区作为一个整体进行裁剪处理。因此，通过确定多边形经每一裁剪窗口边界处理后的新形状来实现一填充多边形的裁剪，如图6.23所示。当然，在最后的裁剪边界确定以前是不需要对多边形进行内部填充的。正如先测试一线段是否完全保存或完全裁掉一样，我们对多边形填充区也进行坐标范围的测试。如果填充区的最小、最大坐标值均在所有四个裁剪边界内，则填充区被保存，留待进一步处理。如果这些坐标范围位于任意一条裁剪边界的外部，则将该多边形从场景中清除（参见图 6.24）。

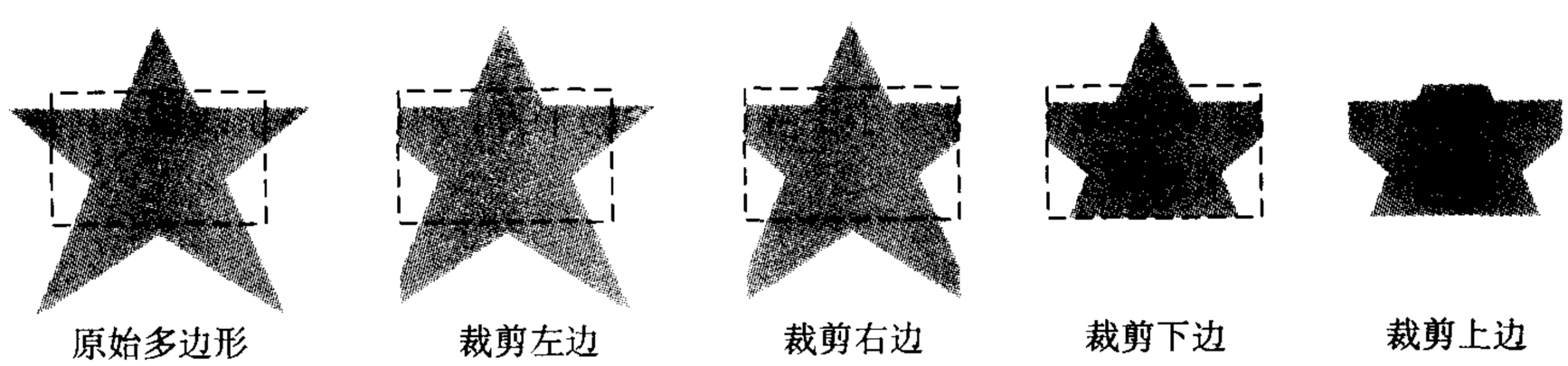


图 6.23　用窗口边界依次裁剪多边形填充区

在我们不能确定一个填充区完全在裁剪窗口内部或外部时，需要定位多边形与裁剪窗口的交点。实现凸多边形裁剪的一种方法是为每一裁剪边界建立一个新的顶点队列，并将其传递给下一边

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

界。最后一次裁剪的输出就是定义裁剪后多边形的顶点队列（参见图6.25）。对于凹多边形的裁剪，我们需要修改这一基本方法使之适合多个顶点队列。

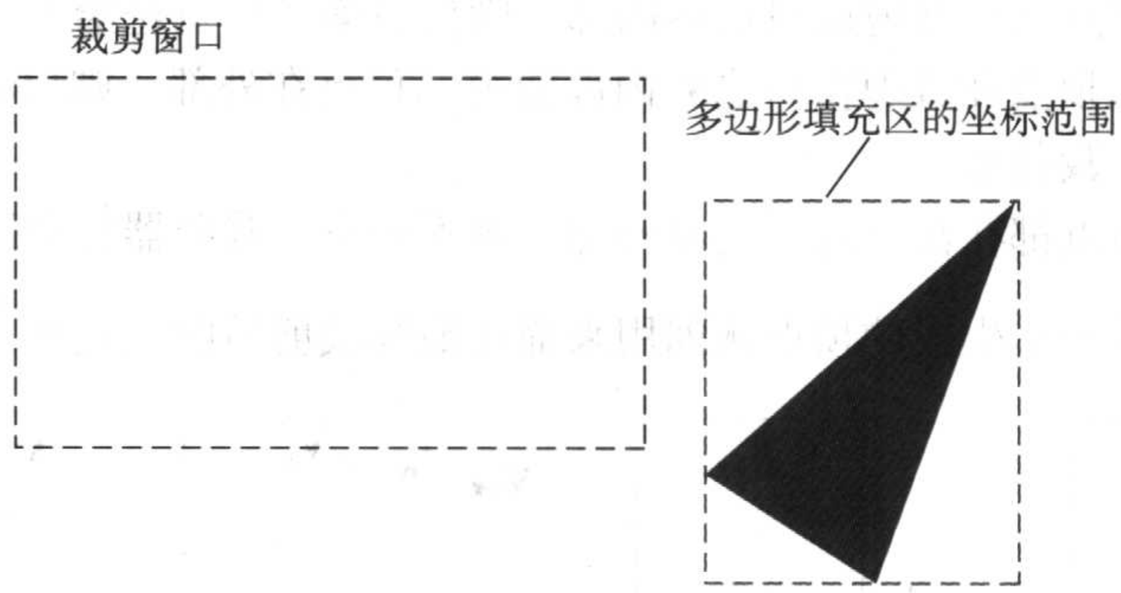


图 6.24　多边形填充区的坐标范围完全在裁剪窗口的右边界外部

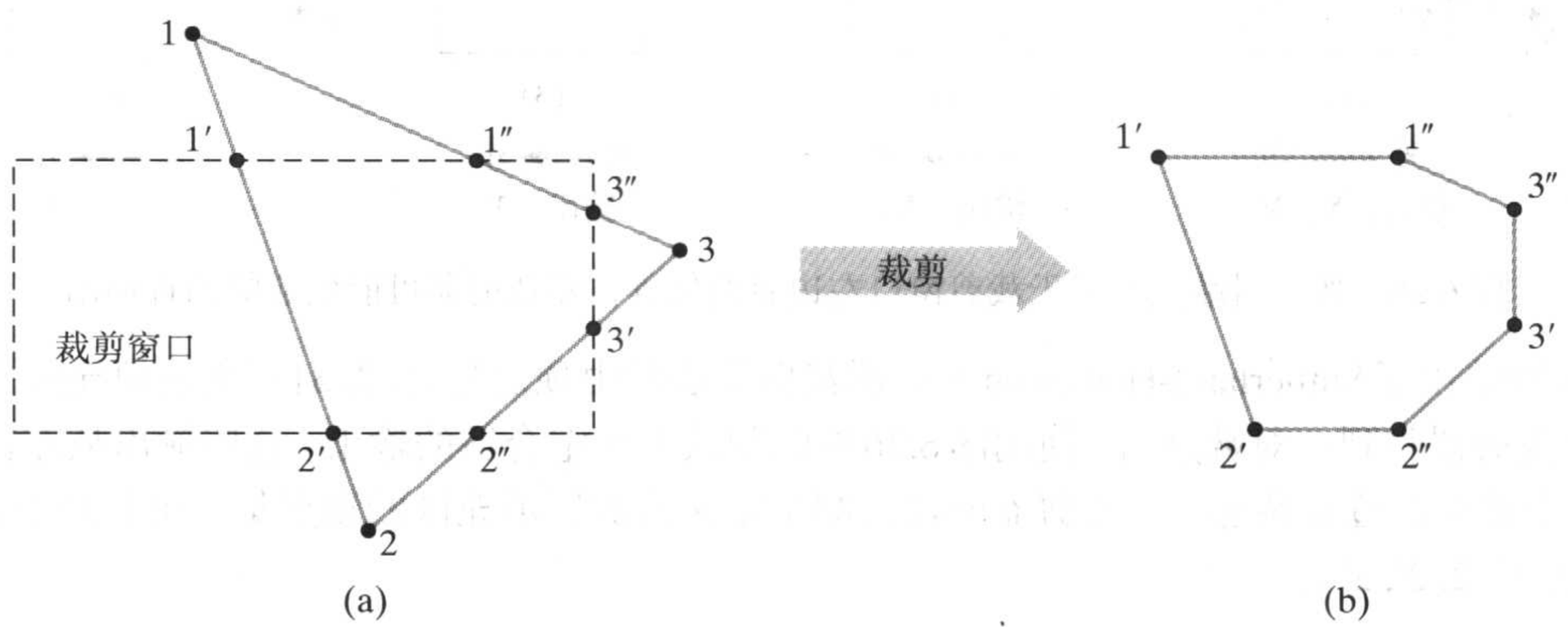


图 6.25　由顶点{1, 2, 3}定义的凸多边形填充区(a)被裁剪出
(b)中由输出顶点{1′, 2′, 2″, 3′, 3″,1″}定义的填充区

6.8.1　Sutherland-Hodgman 多边形裁剪

由Sutherland和 Hodgman提出的裁剪凸多边形填充区的高效算法是将多边形顶点依次传递给每一裁剪阶段，每一个裁剪后的顶点可立即传递给下一阶段。这取消了每一裁剪阶段对成组顶点的需求，从而允许边界裁剪子程序并行地执行。最终的输出是描述裁剪后的多边形填充区边界的顶点队列。

由于Sutherland-Hodgman算法只生成一个输出顶点队列，它不能生成裁剪图6.22(a)中凹多边形边界的结果，即显示在图6.22(b)中的两个输出多边形。但可以为之添加一个处理来获得多个输出顶点队列，从而完成一般凹多边形的裁剪。而基本的Sutherland-Hodgman算法处理其裁剪结果能用单个顶点队列描述的凹多边形。

这一方法的总体策略是顺序地将每一多边形线段的一对顶点送给一组裁剪器（左、右、下、上）。一个裁剪器完成一对顶点的处理后，该边裁剪后留下的坐标值立即送给下一个裁剪器。然后第一个裁剪器处理下一对端点。这样，各个裁剪器可以并行地工作。

在用裁剪边界对多边形的边裁剪时有四种情况需要考虑。一种可能是多边形边的第一端点在裁剪边界外部而第二端点在内部。或两个端点都在裁剪边界内部。另一种可能是第一端点在裁剪边界的内部而第二端点在外部。最后一种可能，两个端点都在裁剪边界外部。

要实现将顶点从一个裁剪阶段传递给下一个，每个裁剪器的输出要用图6.26的格式进行组织。在每对端点通过四个裁剪器中之一时，按下列测试结果为下一裁剪器生成输出。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

1. 如果第一个输入顶点在裁剪窗口边界外部而第二顶点在内部，则将多边形的边与窗口边的交点和第二顶点一起送给下一裁剪器。
2. 如果两个输入顶点都在裁剪窗口边界内部，则仅将第二顶点送给下一裁剪器。
3. 如果第一个输入顶点在裁剪窗口边界内部而第二顶点在外部，则仅将多边形的边与窗口边的交点送给下一裁剪器。
4. 如果两个输入顶点都在裁剪窗口边界外部，则不向下一裁剪器传递顶点。

一组裁剪器的最后一个生成的顶点队列用来描述最终裁剪后的填充区。

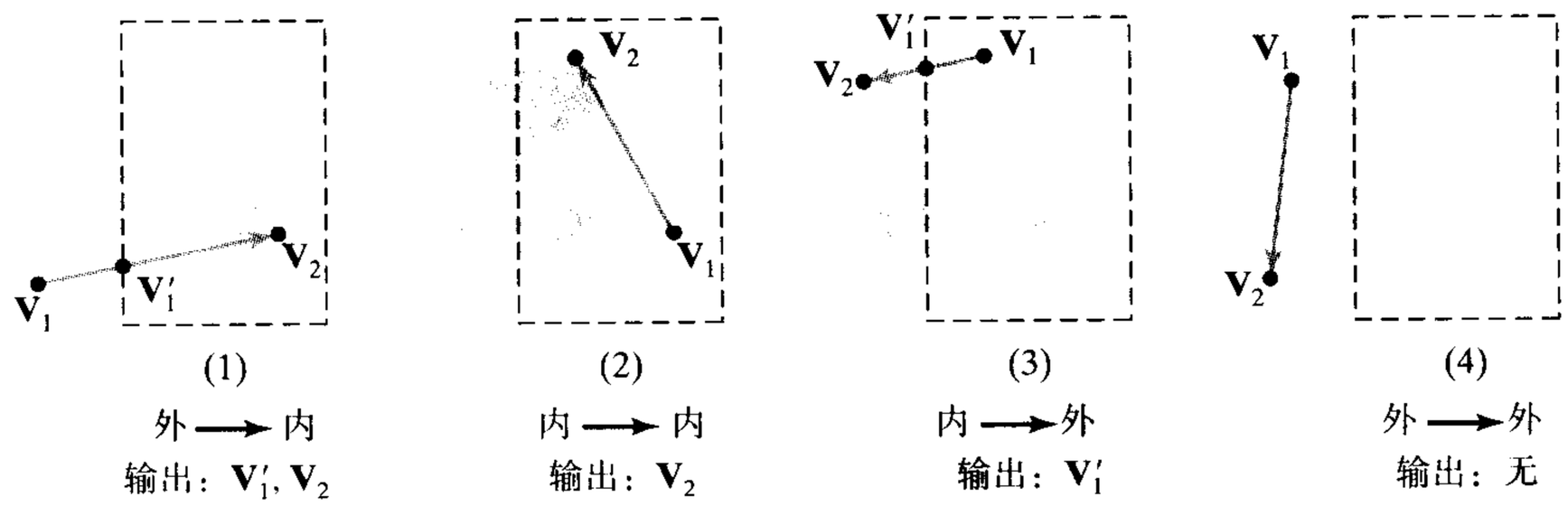


图 6.26 按一对端点相对于裁剪窗口左边界的位置，左裁剪器可能生成的四种输出

图6.27给出了Sutherland-Hodgman多边形裁剪算法对由顶点集{1, 2, 3}定义的填充区处理的例子。一个裁剪器收到一对端点后，使用图6.26中的测试来确定合适的输出。这些输出从左裁剪器向右、下、上裁剪器逐步传递。上裁剪器的输出则是定义裁剪后填充区的顶点集。在本例中，输出顶点队列是{1′, 2, 2′, 2″}。

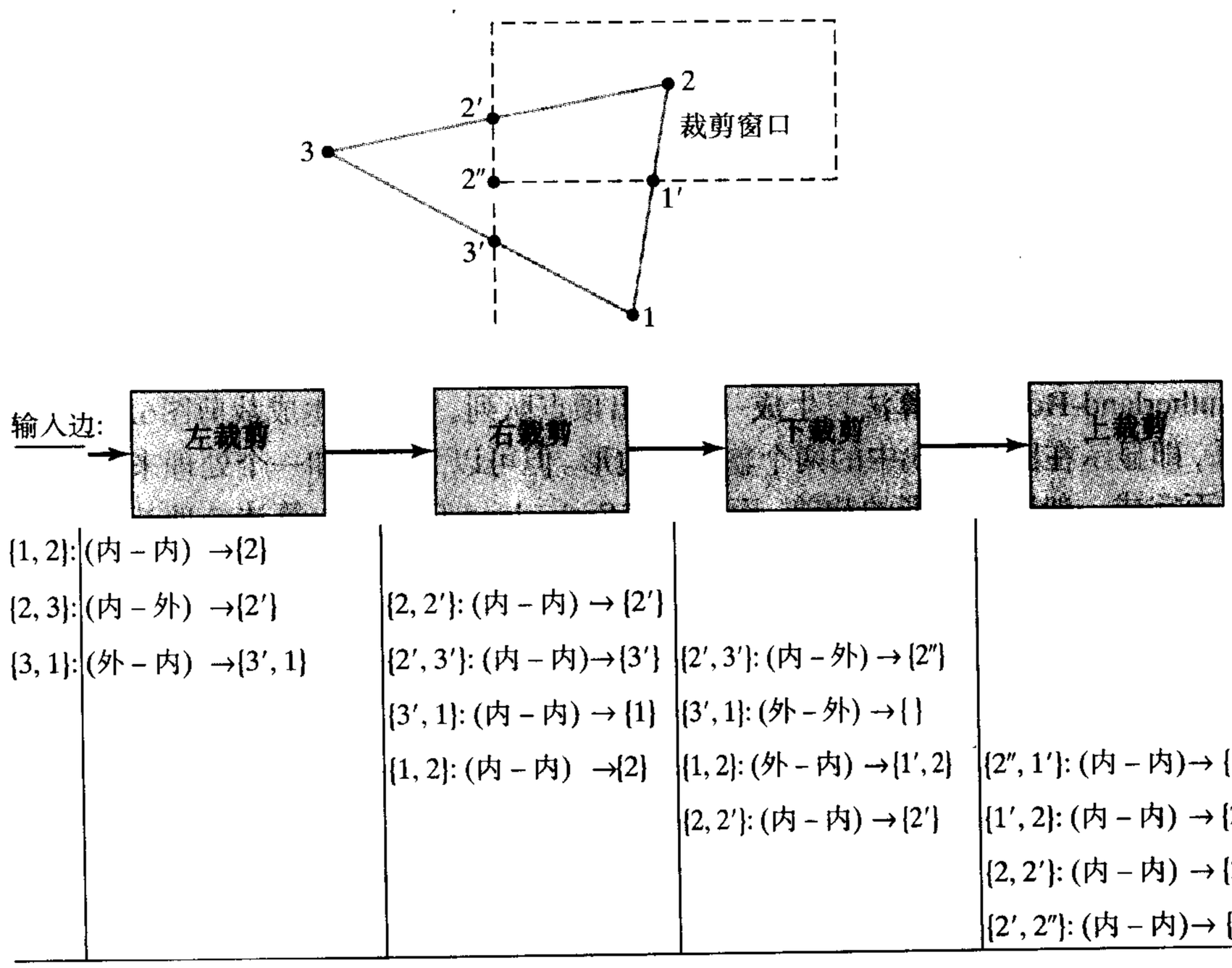


图 6.27 使用 Sutherland-Hodgman 算法对多边形顶点集{1, 2, 3}进行处理。最终裁剪结果是顶点集{1′, 2, 2′, 2″}

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Sutherland-Hodgman多边形裁剪算法的串行实现在下列过程中示出。输入的一组顶点经左、右、下、上裁剪子程序转换为输出顶点队列。

```
typedef enum { Left, Right, Bottom, Top } Boundary;
const GLint nClip = 4;

GLint inside (wcPt2D p, Boundary b, wcPt2D wMin, wcPt2D wMax)
{
  switch (b) {
  case Left:   if (p.x < wMin.x) return (false); break;
  case Right:  if (p.x > wMax.x) return (false); break;
  case Bottom: if (p.y < wMin.y) return (false); break;
  case Top:    if (p.y > wMax.y) return (false); break;
  }
  return (true);
}

GLint cross (wcPt2D p1, wcPt2D p2, Boundary winEdge, wcPt2D wMin, wcPt2D wMax)
{
  if (inside (p1, winEdge, wMin, wMax) == inside (p2, winEdge, wMin, wMax))
    return (false);
  else return (true);
}

wcPt2D intersect (wcPt2D p1, wcPt2D p2, Boundary winEdge,
                                        wcPt2D wMin, wcPt2D wMax)
{
  wcPt2D iPt;
  GLfloat m;

  if (p1.x != p2.x) m = (p1.y - p2.y) / (p1.x - p2.x);
  switch (winEdge) {
  case Left:
    iPt.x = wMin.x;
    iPt.y = p2.y + (wMin.x - p2.x) * m;
    break;
  case Right:
    iPt.x = wMax.x;
    iPt.y = p2.y + (wMax.x - p2.x) * m;
    break;
  case Bottom:
    iPt.y = wMin.y;
    if (p1.x != p2.x) iPt.x = p2.x + (wMin.y - p2.y) / m;
    else iPt.x = p2.x;
    break;
  case Top:
    iPt.y = wMax.y;
    if (p1.x != p2.x) iPt.x = p2.x + (wMax.y - p2.y) / m;
    else iPt.x = p2.x;
    break;
  }
  return (iPt);
}

void clipPoint (wcPt2D p, Boundary winEdge, wcPt2D wMin, wcPt2D wMax,
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
                    wcPt2D * pOut, int * cnt, wcPt2D * first[], wcPt2D * s)
{
  wcPt2D iPt;

  /* If no previous point exists for this clipping boundary,
   * save this point.
   */
  if (!first[winEdge])
    first[winEdge] = &p;
  else
    /*  Previous point exists.  If p and previous point cross
     *  this clipping boundary, find intersection.  Clip against
     *  next boundary, if any.  If no more clip boundaries, add
     *  intersection to output list.
     */
    if (cross (p, s[winEdge], winEdge, wMin, wMax)) {
      iPt = intersect (p, s[winEdge], winEdge, wMin, wMax);
      if (winEdge < Top)
        clipPoint (iPt, b+1, wMin, wMax, pOut, cnt, first, s);
      else {
        pOut[*cnt] = iPt;  (*cnt)++;
      }
    }

  /*  Save p as most recent point for this clip boundary.  */
  s[winEdge] = p;

  /*  For all, if point inside, proceed to next boundary, if any.  */
  if (inside (p, winEdge, wMin, wMax))
    if (winEdge < Top)
      clipPoint (p, winEdge + 1, wMin, wMax, pOut, cnt, first, s);
    else {
      pOut[*cnt] = p;  (*cnt)++;
    }
}

void closeClip (wcPt2D wMin, wcPt2D wMax, wcPt2D * pOut,
                GLint * cnt, wcPt2D * first [ ], wcPt2D * s)
{
  wcPt2D pt;
  Boundary winEdge;

  for (winEdge = Left; winEdge <= Top; winEdge++) {
    if (cross (s[winEdge], *first[winEdge], winEdge, wMin, wMax)) {
      pt = intersect (s[winEdge], *first[winEdge], winEdge, wMin, wMax);
      if (winEdge < Top)
        clipPoint (pt, winEdge + 1, wMin, wMax, pOut, cnt, first, s);
      else {
        pOut[*cnt] = pt;  (*cnt)++;
      }
    }
  }
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
GLint polygonClipSuthHodg (wcPt2D wMin, wcPt2D wMax, GLint n, wcPt2D * pIn,
wcPt2D * pOut)
{
  /*  Parameter "first" holds pointer to first point processed for
   *  a boundary; "s" holds most recent point processed for boundary.
   */
  wcPt2D * first[nClip] = { 0, 0, 0, 0 }, s[nClip];
  GLint k, cnt = 0;

  for (k = 0; k < n; k++)
    clipPoint (pIn[k], Left, wMin, wMax, pOut, &cnt, first, s);

  closeClip (wMin, wMax, pOut, &cnt, first, s);
  return (cnt);
}
```

凹多边形使用Sutherland-Hodgman算法裁剪时，可能显示出一条多余的直线。图6.28给出了这样的例子。这种情况在裁剪后的多边形有两个或者多个分离部分的时候将会出现。因为只有一个输出顶点队列，所以队列中最后一个顶点总是连着第一个顶点。

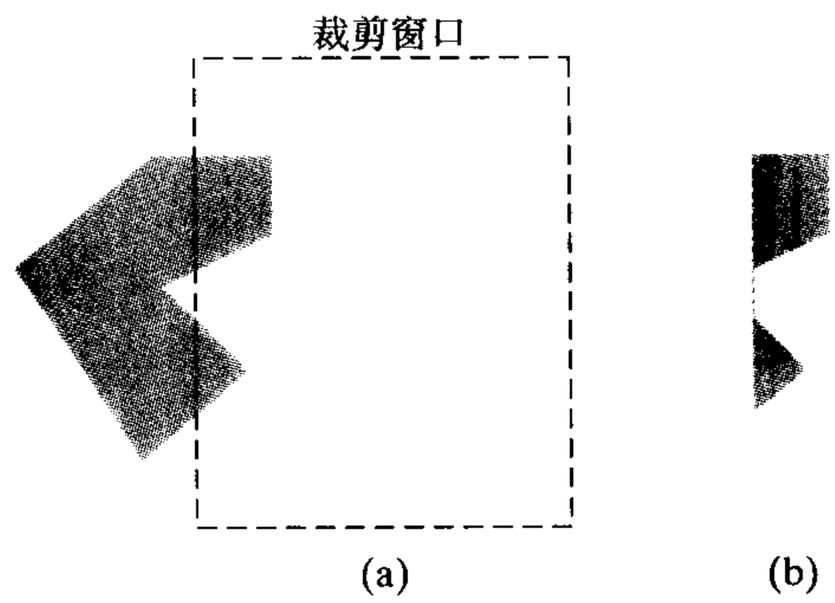


图6.28 使用Sutherland-Hodgman算法裁剪(a)中的凹多边形，产生(b)中的两个连接的区域

为了正确地裁剪凹多边形，我们可以从几个方面入手。一种方法是将凹多边形分割成两个或者更多的凸多边形（参见3.15节），然后使用Sutherland-Hodgman算法分别处理各个凸多边形。另一种方法是修改Sutherland-Hodgman算法，沿着任何一个裁剪窗口边界检查顶点队列。如果发现任一裁剪边界上有两个以上顶点位置则将顶点队列分成两个或多个，从而正确地分割裁剪后的填充区。这可能要求更多的分析来确定裁剪边界上的某些顶点是否应配对或代表裁掉的单个顶点。第三种方法是使用更一般的处理凹多边形的裁剪算法。

6.8.2 Weiler-Atherton算法

这个算法是一种通用的多边形裁剪方法，因而可用于裁剪凸多边形或凹多边形。开始，这个裁剪过程是作为识别可见面的方法而提出的，因此该过程可以用任意形状的裁剪窗口去裁剪任意的多边形填充区域。

Sutherland-Hodgman算法仅仅对填充区的各条边进行裁剪，而Weiler-Atherton算法还要跟踪填充多边形，并寻找使裁剪后的填充区封闭的边界。这样，图6.28(b)中的多个填充区可以作为分开的、不相连的多边形来识别和显示。为了找到使裁剪后的填充区封闭的边，必须沿填充区边界（顺时针或逆时针）在每次离开裁剪窗口时沿裁剪窗口边界走。在裁剪窗口边界上的路线方向与填充区边界上的方向一致。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

处理方向是逆时针还是顺时针依赖于定义多边形填充区的顶点顺序。多数情况下，作为前向面的定义，顶点队列采用逆时针方向描述。因此，形成凸角的两条相邻边向量的叉积确定法向量的方向，即从后向面到前向面的方向。在不知道顶点顺序的时候，可以计算其法向量，或用3.15节中的任一方法找到填充区一个内点。然后，如果顺序地处理各边，而内点始终位于左侧，就可确定逆时针方向。否则，如内点始终在右侧，则确定顺时针方向。

对于逆时针的多边形填充区顶点顺序，可应用下列 Weiler-Atherton 过程。

1. 按逆时针方向处理多边形填充区，直到一对内－外顶点与某裁剪边界相遇；即多边形边的第一顶点在裁剪窗口内而第二顶点在裁剪窗口外。
2. 在窗口边界上从出交点沿逆时针方向到达另一个与多边形的交点。如该点是处理边的点，则走向下一步。如果是新交点，则继续按逆时针方向处理多边形直到遇见已处理的顶点。
3. 形成裁剪后该区域的顶点队列。
4. 回到出交点并继续按逆时针处理多边形的边。

图6.29给出了用标准矩形裁剪窗口对逆时针排列方向与凹多边形进行的Weiler-Atherton裁剪。

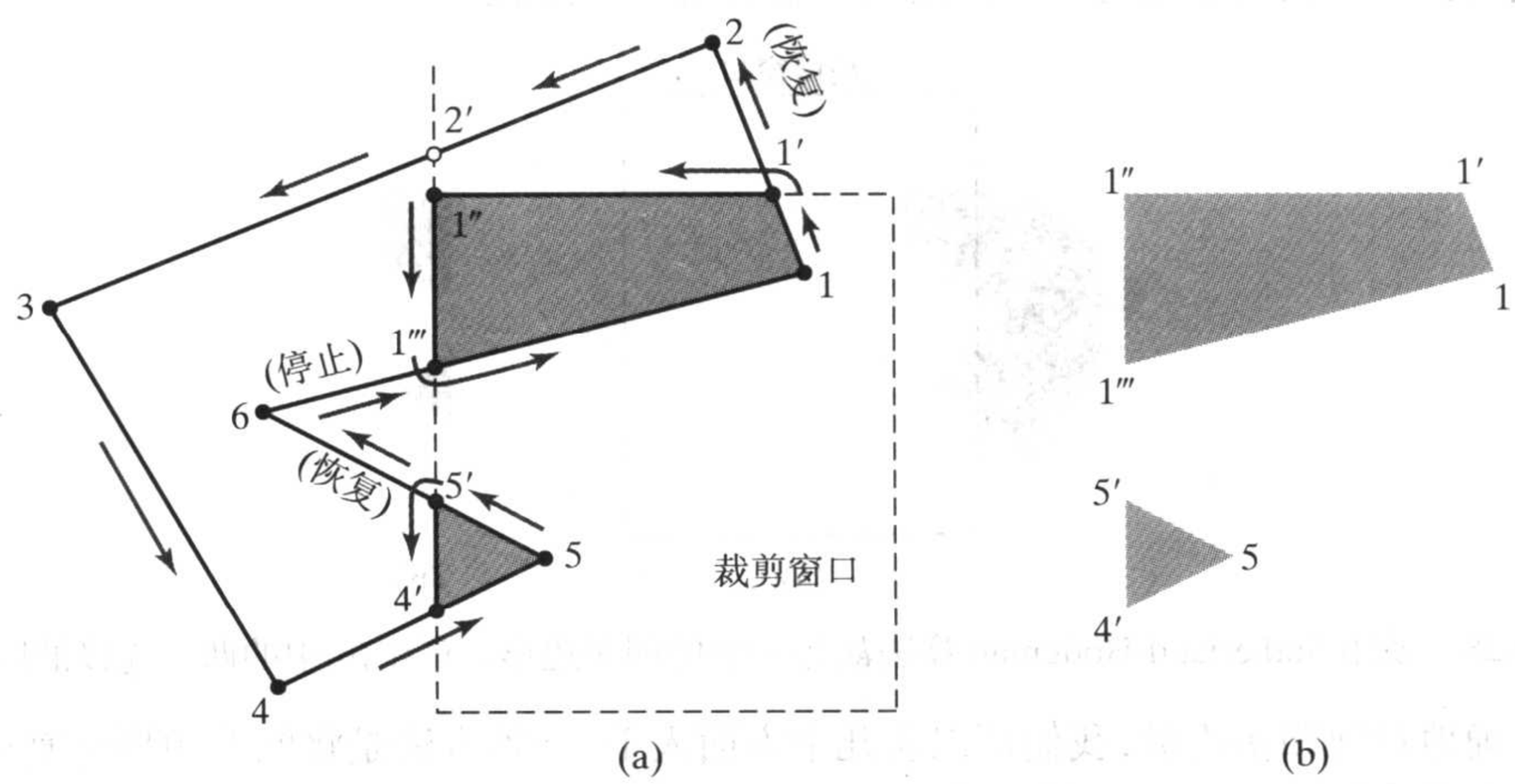


图 6.29 使用 Weiler-Atherton 算法裁剪(a)中由顶点队列{1, 2, 3, 4, 5, 6}定义的凹多边形，产生(b)中两个分离的多边形{1, 1′, 1″, 1‴}和{4′, 5, 5′}

从图6.29(a)中编号为1的顶点开始，按逆时针方向的下一多边形顶点编号为2。因此，该边在窗口的上边界离开裁剪窗口。计算该交点位置（点1′）并在该处向左转按逆时针方向处理窗口边界。在裁剪窗口的上边界上处理时直到左边界都没有与多边形相交。标记转角点为1″并沿左边界到与多边形的交点1‴。接着再沿多边形的边按逆时针方向回到顶点1。这样确认了顶点队列{1, 1′, 1″, 1‴}是原填充区的一个裁剪结果。多边形边的处理重新从点1′开始。由点2和3定义的边穿过左边界的外部，但点2和2′在裁剪窗口上边界的上部且顶点2′和3在裁剪窗口的左侧。同样，端点3和4的边在裁剪窗口左侧。但下一条边（从端点4到5）重新进入裁剪窗口，取得交点4′。端点5、6的边在交点5′处离开窗口，因此向左转向左裁剪边界获得封闭的顶点队列{4′, 5, 5′}。在点5′处重新开始多边形边的处理，回到已处理的点1‴。至此，处理完所有的顶点和边，填充区裁剪完毕。

6.8.3 非矩形的多边形窗口对多边形的裁剪

梁友栋-Barsky算法和其他参数化线裁剪算法特别适合于处理用凸多边形裁剪窗口去裁剪填充区。在该方法中，使用参数法表示填充区和裁剪窗口，且两多边形均用顶点队列描述。先对比填充区和裁

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

剪窗口的包围矩形。如果不能确认填充区完全在裁剪窗口外部，则使用内-外测试来处理参数边方程。完成所有边的测试后，解联立参数线方程来确定窗口交点的位置。

可以通过Weiler-Atherton算法中的边遍历方法用任意多边形裁剪窗口（凸或凹多边形）来处理任意多边形填充区，如图6.30所示。这种情况下，需要如填充区的顶点队列一样按逆时针（或顺时针）方向维护一个裁剪窗口的顶点队列。还需要使用内-外测试来确定填充区顶点是在特定裁剪窗口的边界之内还是之外。这种裁剪方法对在填充区或裁剪窗口中包含已用边界定义的洞时也同样适用。另外，可以使用这种基本方法构造实体几何应用来确认两个多边形的并、交、差。实际上，定位一填充区的裁剪后的区域等价于确定两平面区域的交。

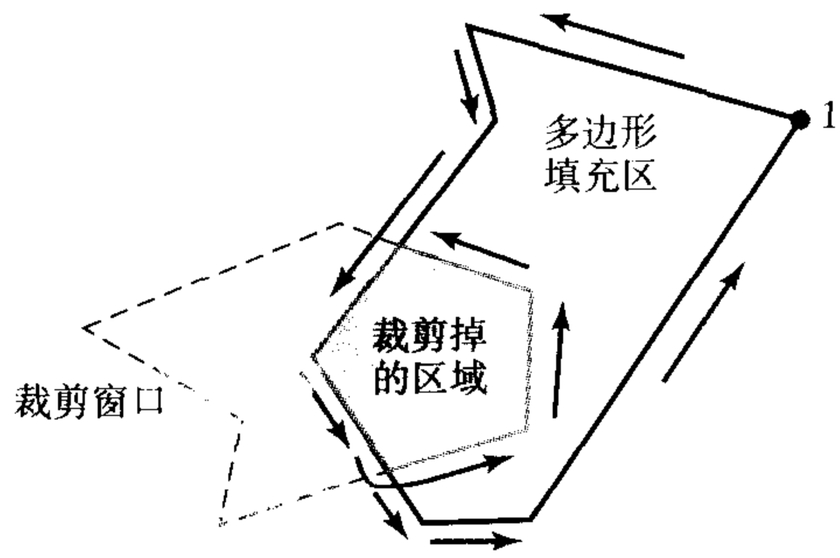


图6.30 使用Weiler-Atherton算法用凹多边形裁剪窗口裁剪多边形

6.8.4 非线性裁剪窗口边界的多边形裁剪

处理用曲线边界裁剪窗口的一种方法是先用直线段逼近边界，然后使用一般的多边形裁剪窗口的裁剪算法对其进行处理。我们还可以使用线段裁剪中讨论的通用算法。首先，比较填充区与裁剪窗口的坐标范围。根据裁剪窗口的形状，可能进行其他一些基于对称考虑的区域测试。对于那些不能确认完全在裁剪窗口内部或外部的填充区，都要计算窗口与填充区的相交位置。

6.9 曲线的裁剪

曲线边界的区域可以使用类似上一节的方法进行裁剪。如果对象已经用直线段逼近则使用多边形裁剪算法。否则，曲线的裁剪过程涉及非线性方程，与线性边界的区域处理相比，需要更多的处理。

可以先测试对象与裁剪窗口的坐标范围来确定是否可以简单地接受或拒绝整个对象。如果不能，则再检查对称性来确定是否可以简单地接受或拒绝。例如，圆有四对称性和八对称性，因此可以检查这些单独的圆区域的坐标范围。仅仅检查整个坐标范围，不能拒绝图6.31中的整个圆。但半个圆在裁剪窗口的右边（或上边界的外部），左上四分之一圆在裁剪窗口的上边，且余下的两个八分之一圆也可用类似的方法排除。

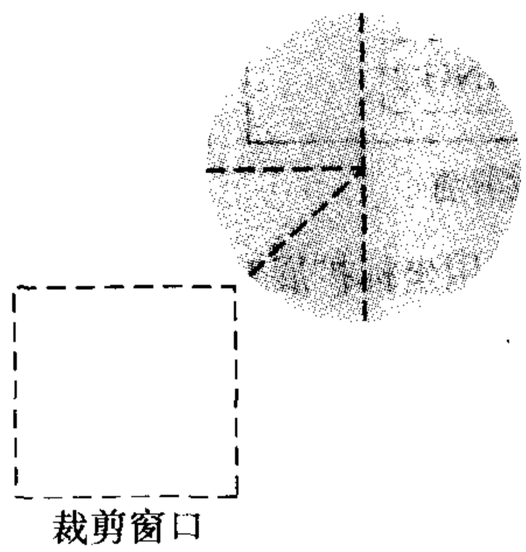


图6.31 一个圆填充区，其四分之一和八分之一位于裁剪窗口外部

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

计算交点时要将裁剪边界位置（xw_{min}, xw_{max}, yw_{min}, yw_{max}）代入对象边界并求解其他坐标值。交点位置求得后，对象的定义位置则可存储以便在扫描转换过程中使用。图6.32给出了用矩形窗口对圆的裁剪。在该例子中，圆的半径和裁剪后的弧端点可用于裁剪后区域的填充，通过引入圆算法定位相交端点之间的弧上位置。

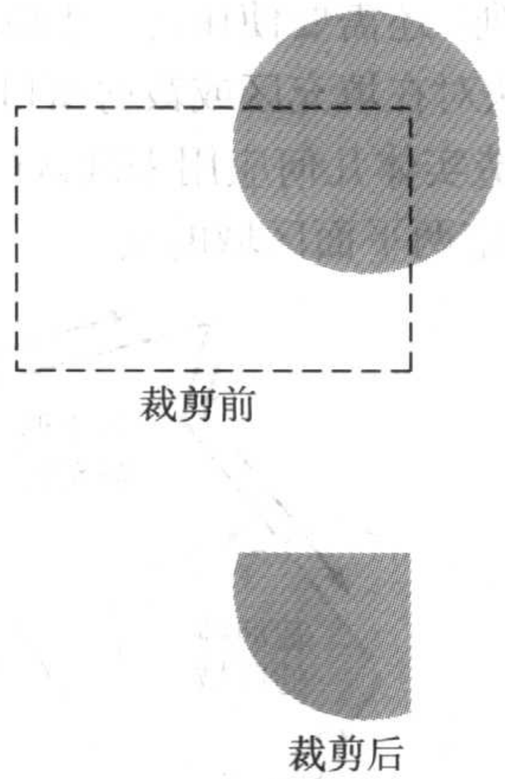


图 6.32　裁剪一个圆填充区

利用一般的多边形裁剪窗口对曲线对象的裁剪可使用类似的方法。第一轮，比较对象包围矩形和裁剪窗口的包围盒。如果不能保存或排除整个对象，则接着求解直线-曲线联立方程来确定裁剪交点。

6.10　文字的裁剪

在图形软件包中有几种对文字裁剪的技术。根据产生文字的方法和具体应用的要求，可以采用不同的技术。

使用窗口边界处理字符串的最简单的方法，是如图6.33所示的全部保留或全部舍弃字符串（all-or-none string-clipping）的裁剪策略。如果字符串中的所有字符都在裁剪窗口内，那么就全部保留这些内容，否则舍弃整个字符串。这种方法通过测试字符串的坐标范围来实现。如果包围矩形的坐标范围并不完全在裁剪窗口内部，则字符串被拒绝。

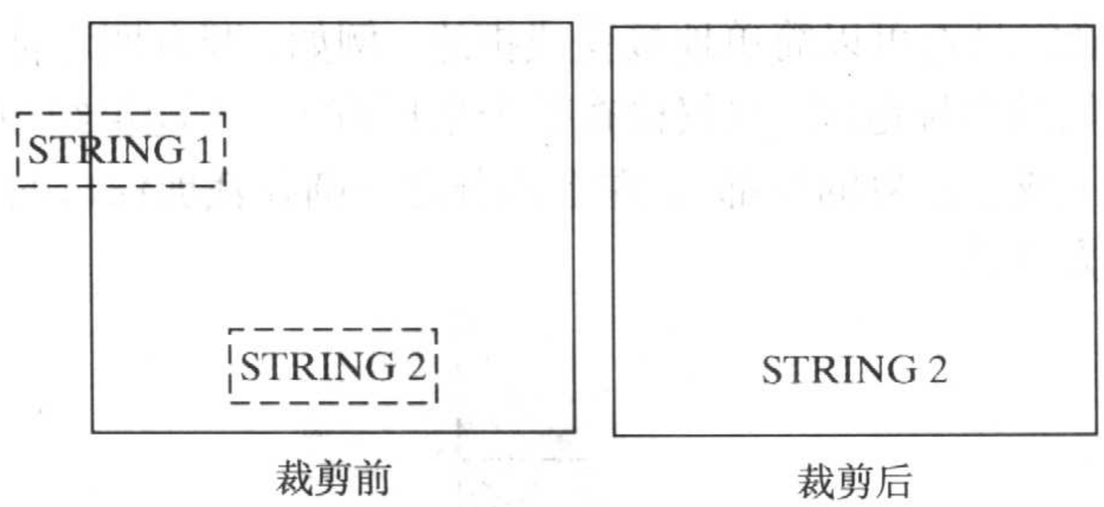


图 6.33　使用坐标扩展对整个字符串进行裁剪

另一种方法是使用全部保留或全部舍弃字符（all-or-none character-clipping）的策略。这里仅舍弃没有完全落在裁剪窗口内的字符（参见图 6.34）。在这种情况下，每一个字符的边界要与窗口进行比较。任何没有完全落在裁剪窗口内的字符都将被裁剪。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

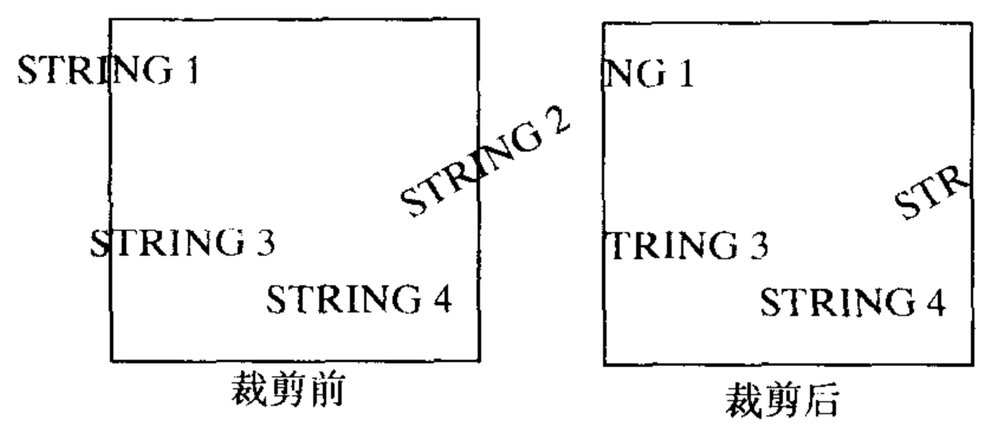


图 6.34　使用字符包围矩形对单个字符进行裁剪

第三种文字裁剪方法是裁剪单个字符的部分。这提供了裁剪后字符串的最精确的显示，但需要较多的处理时间。这里使用了线段裁剪的相同方法对字符进行裁剪。如果一个字符与裁剪窗口边界有重叠，则裁剪掉位于窗口之外的字符部分（参见图 6.35）。由线段构成的轮廓字体可以使用线段裁剪算法进行处理。而使用位图定义的字符，则通过比较字符点阵中的各个像素对于裁剪边界的相对位置而进行裁剪。

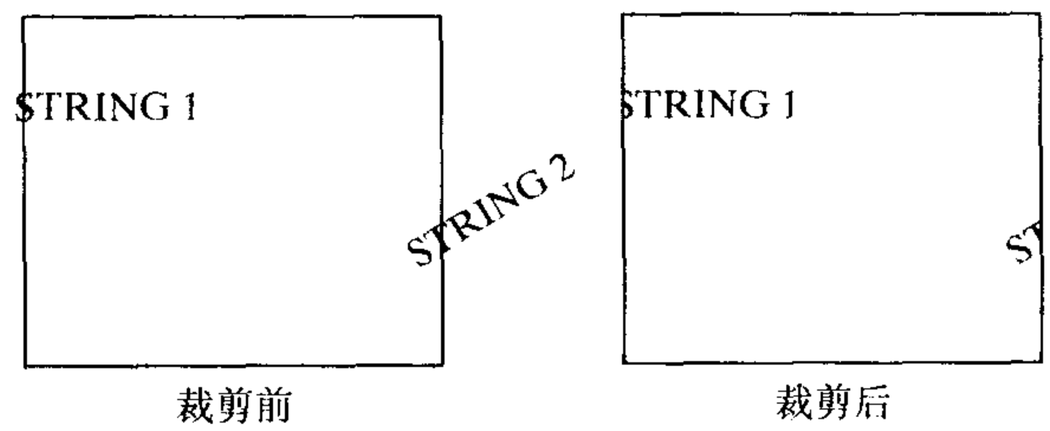


图 6.35　对字符本身进行裁剪

6.11　小结

二维观察变换流水线是将在 xy 平面上定义的世界坐标系图形显示出来的一系列操作。在构造场景后，它被映射到观察坐标系，再到规范化坐标系去裁剪。最后，场景变换到设备坐标系显示。规范化坐标系取值范围为 0 到 1 或从 -1 到 1，它使图形软件包不依赖于具体的输出设备。

我们用裁剪窗口选择场景的一部分在输出设备上显示，裁剪窗口在世界坐标系或参考世界坐标系定义的观察坐标系中描述。裁剪窗口中的内容变换到输出设备的视口去显示。在有些系统中，视口在规范化坐标系中指定。其他系统则在设备坐标系中指定。一般情况下，裁剪窗口和视口是标准位置的矩形，其边界平行于坐标轴。对象映射到视口后其在视口中的相对位置与其在裁剪窗口中的相对位置一样。为了保持对象的比例，视口与对应的裁剪窗口必须具有相同的纵横比。对同一场景可以建立任意数量的裁剪窗口和视口。

裁剪算法通常在规范化坐标系中执行，因此所有不依赖于设备坐标的几何变换和观察变换都可以合并成一个变换矩阵。使用设备坐标系中指定的视口时，可在将规范化、对称正方形中的内容变换到视口之前，使用规范化的、对称的、其坐标范围从 -1 到 1 的正方形来裁剪二维场景。

各种图形软件包都包含裁剪直线段和多边形填充区的子程序。包含描述点和文字串函数的软件包也会包含这些图元的裁剪子程序。由于裁剪计算耗时，改进裁剪算法仍然是计算机图形学的一个重点考虑的领域。Cohen 和 Sutherland 提出了使用区域码标识线端点相对于裁剪窗口边界的位置的裁剪算法。端点区域码用来快速确认那些完全在裁剪窗口内部或外部的线段。对余下的线段必须计算边界上的交点位置。梁友栋和 Barsky 提出了与 Cyrus-Beck 算法类似的、用参数方程表示线段的快速线裁剪算法。该方法在求交前进行更多的测试。Nicholl-Lee-Nicholl 算法通过在 xy 平面上使用

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

更多的区域测试而进一步减少了求交操作。参数化的线裁剪算法较容易扩充到凹裁剪窗口和三维场景。因为Cohen-Sutherland算法是第一个提出的算法，所以得到了广泛的应用，其中的区域码用于识别线段端点与矩形裁剪窗口边界的相对位置。对于不能立刻确定为完全落在窗口内或者完全落在窗口外的线段，要按照窗口边界进行裁剪。梁友栋-Barsky算法采用参数化线段表达式，类似于早期的Cyrus-Beck算法，该算法减少了求交的次数，从而使裁剪算法的效率得到提高。Nicholl-Lee-Nicholl算法通过对xy平面进行更多的区域测试来减少求交运算。参数化的线段裁剪法便于扩充到凸多边形裁剪窗口和三维裁剪窗口。然而，Nicholl-Lee-Nicholl算法仅能用于二维线裁剪。

用于凹多边形裁剪窗口的线段裁剪算法也已提出。一种方法是将凹多边形分割为几个凸多边形并使用参数化线裁剪方法。另一种方法是在凹裁剪窗口上增加边使其修改为凸窗口。然后进行一系列的外裁剪和内裁剪来获得裁剪后的线段。

尽管曲线边界的裁剪窗口较少使用，不过我们可以使用类似的方法。但这里的求交计算需要求解非线性方程。

多边形填充区由边的队列定义，而多边形裁剪过程必须保留关于在各个处理阶段中裁剪后的边如何连接成多边形的信息。在Sutherland-Hodgman算法中，填充区的一对顶点经由每个边界裁剪器处理，获得的边裁剪信息立即传递给下一裁剪例程，因而四个裁剪例程（左、右、下、上）并行工作。该方法是裁剪凸多边形填充区的高效方法。但在裁剪包含不相连部分的凹多边形来说，Sutherland-Hodgman算法会生成多余的连接线段。参数化线段裁剪器的扩充，如梁友栋-Barsky方法，也可用来裁剪凸多边形填充区。Weiler-Atherton算法使用边界遍历方法，对凸多边形和凹多边形都能正确地裁剪。

使用参数化直线表示方法的扩充使填充区可被凸裁剪窗口裁剪。Weiler-Atherton算法可以使用任意多边形裁剪窗口裁剪任意多边形填充区域。通过窗口的多边形逼近或用曲线窗口边界处理填充区，可以实现利用具有非线性边界的裁剪窗口对填充区的裁剪。

全部保留或全部舍弃策略是最快速的文字裁剪方法，它在字符串有一部分在任一裁剪边界之外时裁去整个字符串。我们也可只裁掉那些不完全在裁剪窗口内的字符。最精确的文字裁剪方法是使用点、线段、多边形或曲线等裁剪算法对字符串中的每一字符按其由点阵还是轮廓方式进行裁剪。

尽管OpenGL是为三维应用而定义的，但是它也提供了二维GLU函数以描述世界坐标系中的标准矩形裁剪窗口。OpenGL中裁剪窗口的坐标是投影变换的参数。因此，我们要先引入投影矩阵模式。接着用OpenGL基本库的函数指定视口，用GLUT函数指定显示窗口。可以使用多种GLUT函数设定各种显示窗口的参数。表6.1总结了OpenGL的二维观察函数，其中也列出了与观察相关的函数。

表6.1 OpenGL二维观察函数小结

函数	描述
gluOrtho2D	指定作为二维正交投影参数的裁剪窗口坐标
glViewport	为视口指定屏幕坐标参数
glGetIntegerv	使用变量GL_VIEWPORT和vpArray获取当前视口的参数
glutInit	初始化GLUT函数库
glutInitWindowPosition	指定显示窗口左上角的坐标
glutInitWindowSize	指定显示窗口的宽和高
glutCreateWindow	创建一显示窗口（赋予一整数标识）并指定显示窗口标题
glutInitDisplayMode	为显示窗口选择缓存和颜色模式等参数
glClearColor	为显示窗口指定背景色
glClearIndex	使用颜色索引模式为显示窗口指定背景色

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

（续表）

函数	描述
glutDestroyWindow	指定显示窗口的标识码，删除该窗口
glutSetWindow	指定显示窗口的标识码，使该窗口成为当前窗口
glutPositionWindow	重新设定当前显示窗口的屏幕定位
glutReshapeWindow	重新设定当前显示窗口的宽度和高度
glutFullScreen	设定显示窗口的尺寸为整个屏幕
glutReshapeFunc	指定在显示窗口尺寸改变时要引用的函数
glutIconifyWindow	当前显示窗口转变为一个图标
glutSetIconTitle	为显示窗口图标指定一标号
glutSetWindowTitle	为当前显示窗口指定新标题
glutPopWindow	将当前显示窗口移到“顶部”；即在所有其他窗口之前
glutPushWindow	将当前显示窗口移到“底部”；即在所有其他窗口之后
glutShowWindow	让当前显示窗口回到屏幕上
glutCreateSubWindow	在一显示窗口中建立下一层的窗口
glutSetCursor	为屏幕光标选择形状
glutDisplayFunc	在当前显示窗口中引入函数来建立图形
glutPostRedisplay	重画当前窗口的内容
glutMainLoop	执行计算机图形程序
glutIdleFunc	指定在系统为空闲时要执行的函数
glutGet	按指定状态参数询问系统

参考文献

线段裁剪算法的讨论请参见 Sproull and Sutherland(1968)、Cyrus and Beck(1978)、Liang and Barsky(1984)和 Nicholl, Lee, and Nicholl(1987)。对 Cohen-Sutherland 线段裁剪算法的改进请参见 Duvanenko(1990)。

基本的多边形裁剪算法请参见 Sutherland and Hodgman(1974)和 Liang and Barsky(1983)。任意形状多边形相互裁剪的通用技术请参见 Weiler and Atherton(1977)和 Weiler(1980)。

OpenGL 的观察操作在 Woo, Neider, Davis, and Shreiner(1999)中讨论。显示窗口 GLUT 函数在 Kilgard(1996)中讨论，另外的关于 GLUT 的信息可从网站：http://reality.sgi.com/opengl/glut3/glut3.html 中获得。

练习题

6.1　给定观察坐标原点 $\mathbf{P}_0$ 和观察向量 $\mathbf{V}$，编写一个计算从二维世界坐标系到观察坐标系的变换矩阵（6.1）的元素的程序。

6.2　推导出将裁剪窗口内容变换到视口的变换矩阵（6.8），要求首先将窗口缩放到视口的尺寸，然后平移到视口的位置。以裁剪窗口的中心作为缩放和平移操作的参考点。

6.3　编写一个计算裁剪窗口到对称的规范化正方形的变换矩阵（6.9）元素的程序。

6.4　编写一组程序，实现不含裁剪操作的观察流水线。要求使用模型坐标变换、特定的观察系统、向对称的规范化正方形的变换等。

6.5　写出一个完整实现 Cohen-Sutherland 线裁剪算法的程序。

6.6　讨论梁友栋 -Barsky 线裁剪算法中各种测试和交点参数 u_1 和 u_2 计算方法的原理。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

6.7 比较若干条相对于裁剪窗口的不同方向线段的Cohen-Sutherland和梁友栋-Barsky裁剪算法的算术运算次数。

6.8 写出一个完整实现梁友栋-Barsky线裁剪算法的程序。

6.9 设计将图6.16中 xy 平面上三个区域的交点计算映射到其他六个区域的对称变换。

6.10 建立任意输入一对顶点的Nicholl-Lee-Nicholl线段裁剪的详细算法。

6.11 比较若干条相对于裁剪窗口的不同方向线段的NLN算法、Cohen-Sutherland和梁友栋-Barsky裁剪算法的算术运算次数。

6.12 将梁友栋-Barsky裁剪算法改成多边形裁剪算法。

6.13 假设裁剪窗口为标准位置的矩形，设计Weiler-Atherton多边形裁剪的详细算法。

6.14 设计Weiler-Atherton多边形裁剪算法，其中裁剪窗口为任意多边形。

6.15 设计Weiler-Atherton多边形裁剪算法，其中裁剪窗口为任意特殊的多边形（凸或凹）。

6.16 编写使用矩形窗口裁剪标准位置椭圆的算法。

6.17 假设字符串的所有字符具有同样的宽度，按照“全部保留或全部舍弃字符”的裁剪策略，设计字符串裁剪算法。

6.18 使用特定尺寸的像素网格定义字符，设计裁剪单个字符的算法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第7章 三维观察

在二维图形应用中，观察操作将世界坐标平面上的点变换到输出设备平面中的像素位置。利用矩形的裁剪窗口和视口，二维图形软件包裁剪场景并将其映射到设备坐标系。因为我们在如何构造场景、如何在输出设备上生成视图等方面有更多的选择，所以使三维图形操作涉及了更多的方面。

7.1 三维观察概念综述

在对一场景建模时，场景中每一对象一般由包围该对象、形成封闭边界的一组面来定义。对于有些应用需指定对象的内部结构信息。除生成对象表面特征视图的过程外，图形软件包有时还提供显示实体对象的内部组成或剖面。观察函数通过一组将对象的指定视图投影到显示设备表面上的过程来处理对象的描述。三维观察中的许多处理，如裁剪子程序，与二维观察流水线中的类似。但三维观察包含一些在二维观察中没有的任务。例如，需要有投影子程序将场景变换到平面视图、必须识别可见部分、对逼真显示要考虑光照效果和表面特征。

7.1.1 三维场景观察

要获得三维世界坐标系场景的显示，必须先建立观察用的坐标系，或“照相机”参数。该坐标系定义与照相机胶片平面对应的观察平面或投影平面（view plane 或 projection plane）的方向（参见图7.1）。然后将对象描述转换到观察坐标系并投影到观察平面上。我们可以用线框图形式在输出设备上生成如图 7.2 所示的对象视图，或应用光照和面绘制技术获得可见面的真实感图形。

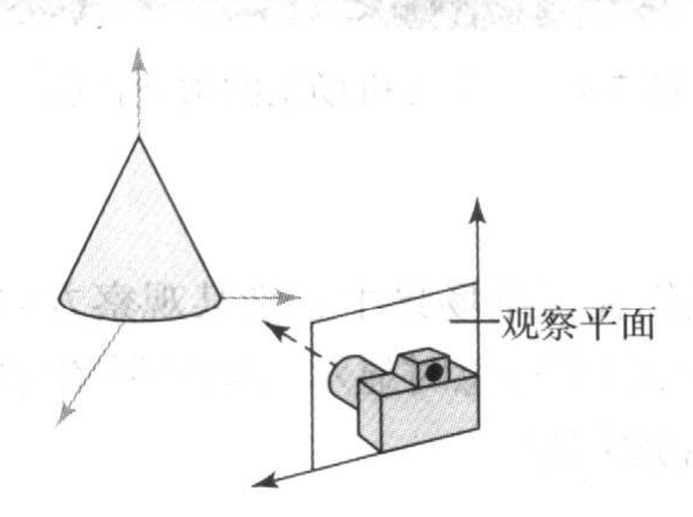


图 7.1 为获得一个选定的三维场景视图的坐标参考系

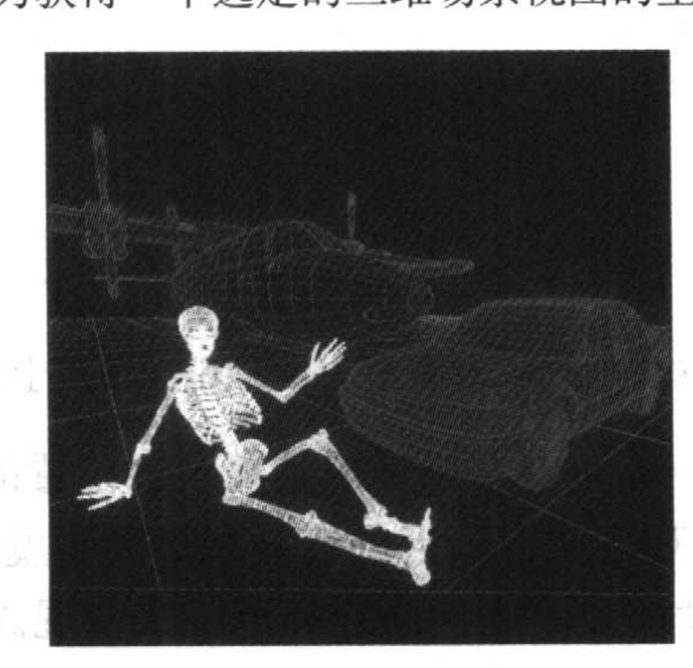


图 7.2 一个三维对象的消除隐藏线的线框图显示，选自一个商用的对象形体数据库

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

7.1.2 投影

与照相机不同的是，我们可以选择多种不同方法将场景投影到观察平面上。一种将实体描述投影到观察平面的方法是沿平行方向投影每个点。这种称为平行投影（parallel projection）的技术用于工程和建筑制图通过显示对象实际尺寸的一组视图来表达对象，如图 7.3 所示。

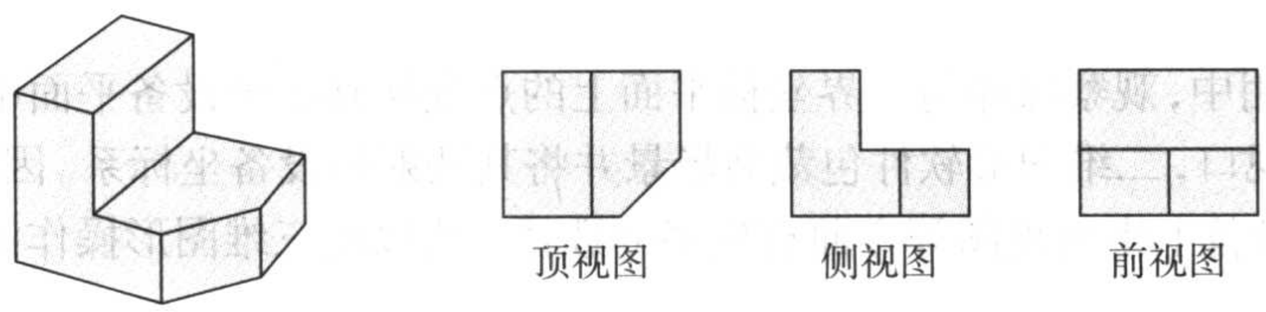


图 7.3 一对象的三个平行视图，在不同的观察位置给出相关比例

另一种观察三维场景的方法是沿会聚路径投影每一点。这种称为透视投影（perspective projection）的处理造成离视点远的对象比同样大小但离视点较近的对象显示得小。利用透视投影方法生成的图形看起来更真实，因为它遵循了人眼和照相机镜头获得图像的原理。在图7.4给出的透视投影图中，沿观察方向的平行线会聚到背景远方的一点，背景中的飞机比前景中的飞机小。

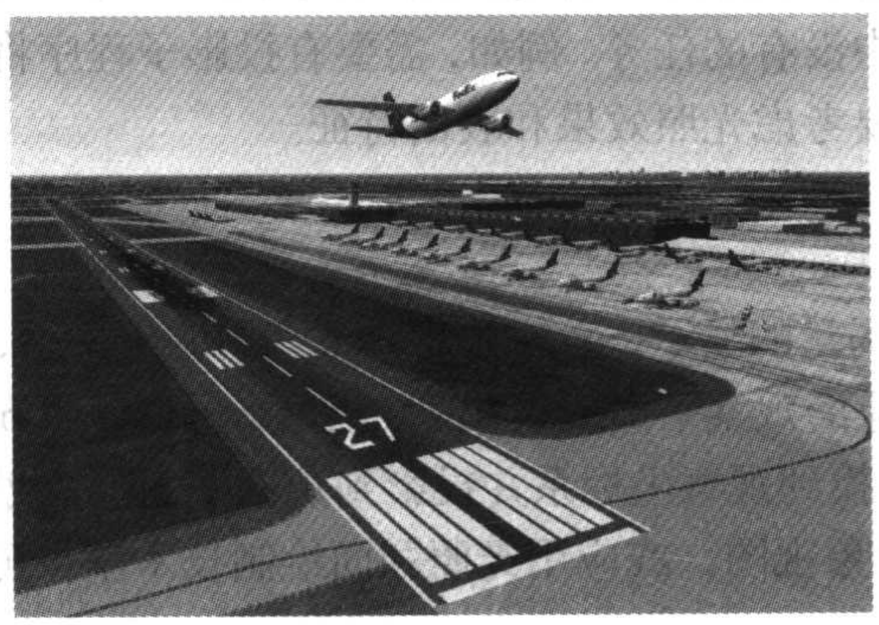


图 7.4 一架飞机场景的透视投影

7.1.3 深度提示

除少数特例外，深度信息对我们在三维场景中鉴别某观察方向显示的对象中哪个在前、哪个在后是很重要的。图7.5给出了在没有深度信息时显示对象线框图可能出现的模糊性。在实体对象的二维表示中可以有多种方法引入深度信息。

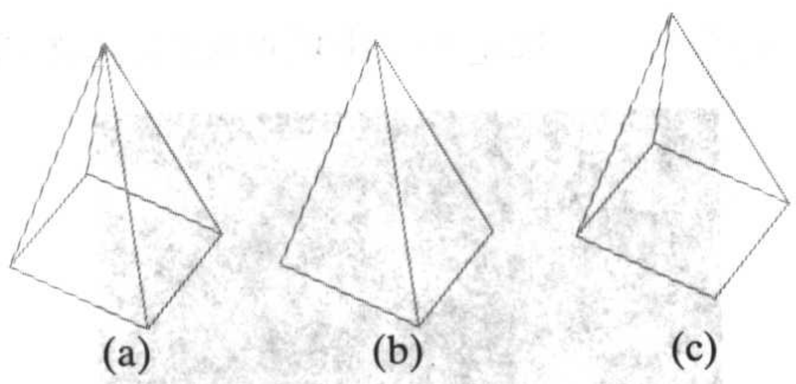


图 7.5 棱锥体的线框图：(a)不包含深度提示及观察方向；(b)在顶点上方向下看；(c)在底面之下往上看

在线框图显示中指示深度的一种简单方法是按其离观察位置的距离来改变线段的亮度。图7.6给出了带深度提示的线框对象。离观察位置最近的线段用最高亮度显示，较远的线段在显示时逐步减弱亮度。通过选定最大最小强度值及其变化的距离范围来实现深度提示。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

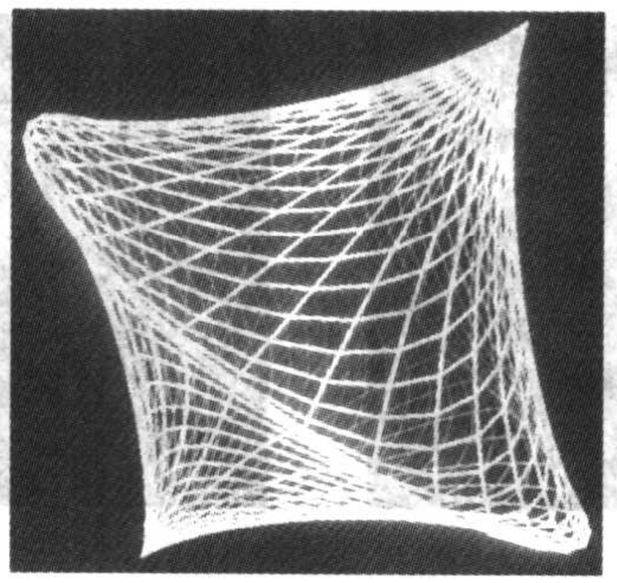


图 7.6 带深度提示的线框图，线段的亮度由前向后逐步减弱

深度提示的另一应用是在感觉对象的强度上模拟雾气效果。因为在光线投射中有灰尘、雾和烟等造成较远的对象比近的对象模糊。

7.1.4 可见线和可见面的判定

我们还可以在线框图中用不同于深度提示的技术来说明深度关系。一种方法是简单地醒目显示可见线或用不同颜色显示它们。另一种技术通常用于工程制图中，是用虚线显示不可见线条。也可如图7.5(b)和图7.5(c)所示从显示中隐去不可见线条。但隐去不可见线条的同时也隐去了关于对象背面的形状信息，而线框图表示一般用于表现一个对象从前往后的整体外貌。

在生成场景的真实感图形时，对象的背后部分全部消除而仅显示可见面。此时，使用面绘制过程使屏幕像素仅包含前向面的颜色图案。

7.1.5 面绘制

通过使用场景中的光照条件和赋予的表面特征绘制对象表面，可以增加显示的真实感。通过指定光源的颜色和位置来设定光照条件，也可以设定背景光照效果。对象的表面特征包括表面是否透明及是否光滑。我们可设定不同的参数值来模拟玻璃、塑料、木纹图案及橘子的皱皮。在图7.7中，面绘制方法与透视投影及可见面识别方法一起生成了有一定真实感的场景显示。

图 7.7 通过透视投影、光照效果和选择表面特征生成的真实感房间显示

7.1.6 拆散和剖切面视图

许多图形软件包允许按层次结构方式定义对象，因此可以存储内部细节。这种对象的剖切面视图可用来展示其内部结构及对象各部分之间的关系。图7.8给出了机械设计中的多种拆散显示。剖切面视图是另一种展示对象组成部分的方法，如图7.9所示，消去了部分可见面以展示内部结构。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

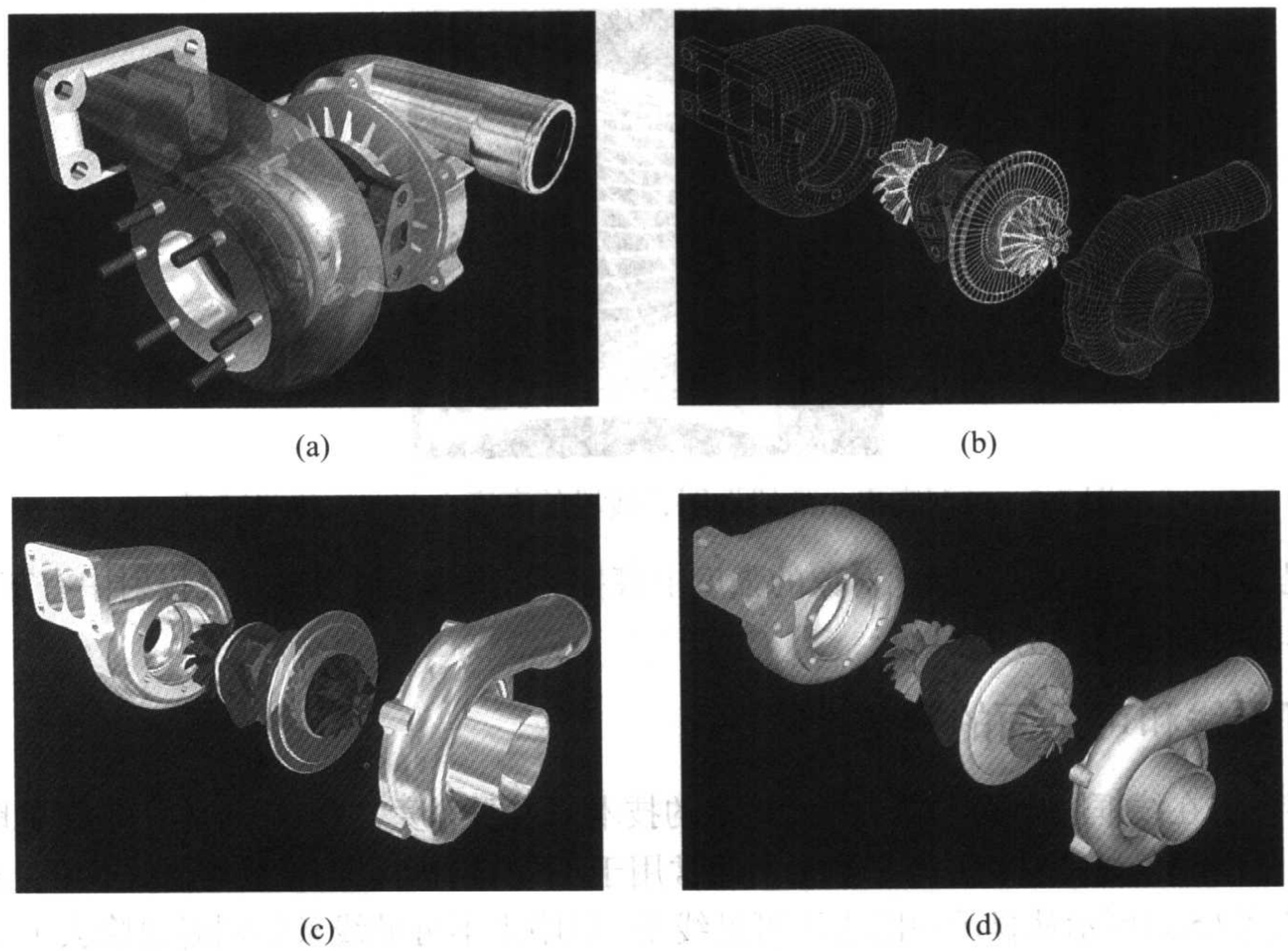


图 7.8 (a)一个完成绘制和安装的涡轮；(b)按拆散、线框图方式显示；(c)面绘制的拆散式显示；(d)面绘制、颜色的拆散显示

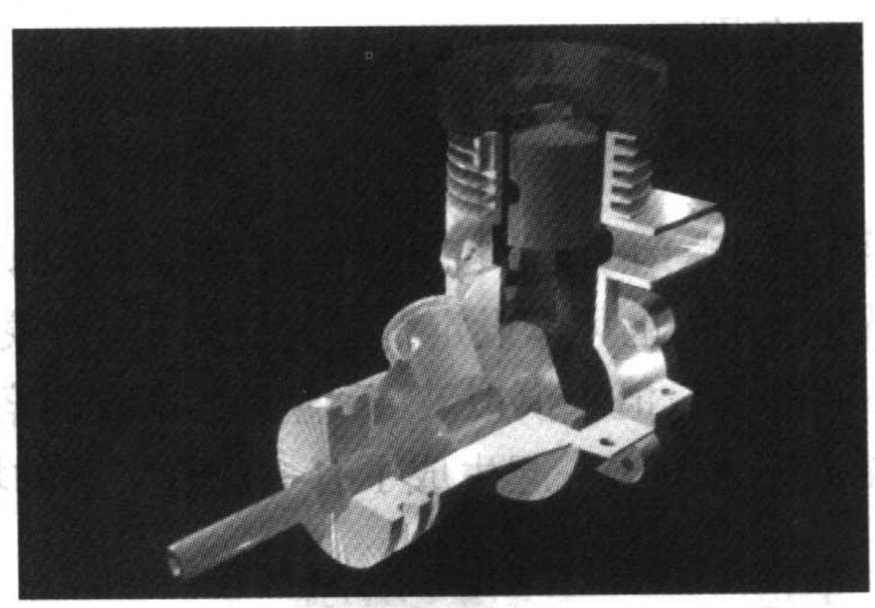


图 7.9 割草机的彩色剖切面视图，展示内部各部分的结构和关系

7.1.7 三维和立体视图

为计算机生成的场景增加真实感的另外一些方法包括三维显示和立体视图。如第2章所述，三维视图可通过从一个振动的柔性镜子中反射出光栅图像来实现。镜子的振动与CRT上场景的显示同步。镜子振动时，焦距改变使场景中每一点反射到与其深度对应的空间位置。

立体设备给出场景的两个视图：一个为左眼，另一个为右眼。观察位置与观察者的眼睛位置对应。这两个视图交替地在光栅监视器的刷新周期内显示。当我们使用特殊的眼镜观看监视器时，该眼镜与监视器同步地交替关闭两个镜头，即可看到三维效果的场景显示。

7.2 三维观察流水线

三维场景视图的计算机生成步骤有点类似于拍一张照片的过程。首先，与安放照相机相对应，需要在场景中确定一个观察位置。根据要显示场景的前、后、侧、上或底来选择观察位置。也可

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在一组对象的中间、甚至在一个对象如一个建筑物或一个分子的内部选择一个观察位置。然后需要确定照相机的方向（参见图7.10）。照相机朝哪个方向照及如何绕视线旋转照相机以确定相片的向上方向？最后，当按下快门时，按照相机“裁剪窗口”（镜头）的大小来修剪场景，光线从可视表面投影到照相机胶片上。

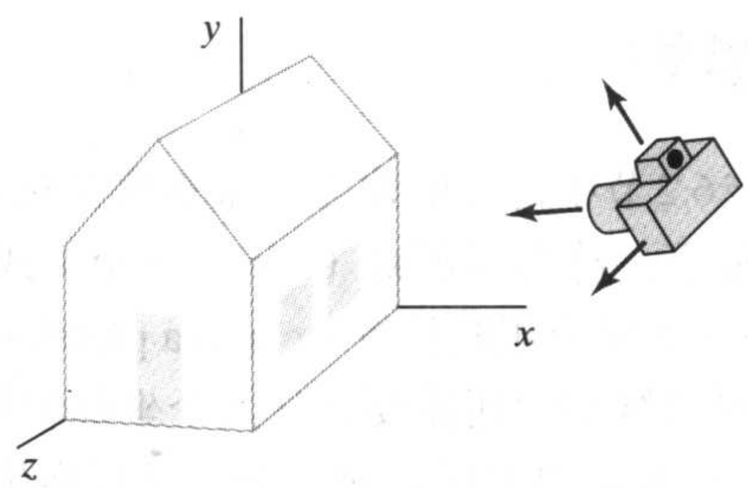


图7.10 对场景拍照需要选择照相机的位置和方向

要记住，与使用照相机相比，利用图形软件包生成场景的视图有更大的灵活性和更多的选择。我们可以选择平行投影或透视投影，还可以有选择地沿视线消除一些场景部分。可以将投影平面移出“照相机”位置，甚至在我们的人造照相机背后获得对象的图片。

三维场景的某些观察操作与二维观察流水线（参见6.1节）相同或类似。二维视口用来确定三维场景投影视图在输出设备上的位置，而二维裁剪窗口用来选择向视口映射的视图。我们在屏幕坐标系中建立显示窗口，就像在二维应用中所做的那样。裁剪窗口、视口和显示窗口通常指定为其边平行于坐标轴的矩形。而在三维观察中，裁剪窗口位于所选择的观察平面上，场景相对于用一组裁剪平面（clipping plane）定义的封闭空间体来裁剪。观察位置、观察平面、裁剪窗口和裁剪平面都在观察坐标系中指定。

图7.11给出了对建立三维场景以及将场景变换到设备坐标的一般处理步骤。一旦在世界坐标系中建好场景模型，就将场景描述转换到选择的观察坐标系。观察坐标系定义了观察参数，包括投影平面（观察平面）的位置和方向，我们可以把投影平面看做是照相机胶片平面。然后在投影平面上定义与照相机镜头对应的二维裁剪窗口，并建立三维裁剪区域。该裁剪区域称为**观察体**（view volume），其形状和大小依赖于裁剪窗口的尺寸、投影方式和所选的观察方向的边界位置。投影操作将场景的观察坐标描述转换为投影平面的坐标位置。对象映射到规范化坐标系，所有在观察体外的部分被裁剪掉。裁剪操作可以在所有与设备无关的坐标变换（从世界坐标系到规范化坐标系）完成之后进行。这样，坐标变换可以合并以便最大限度地提高效率。

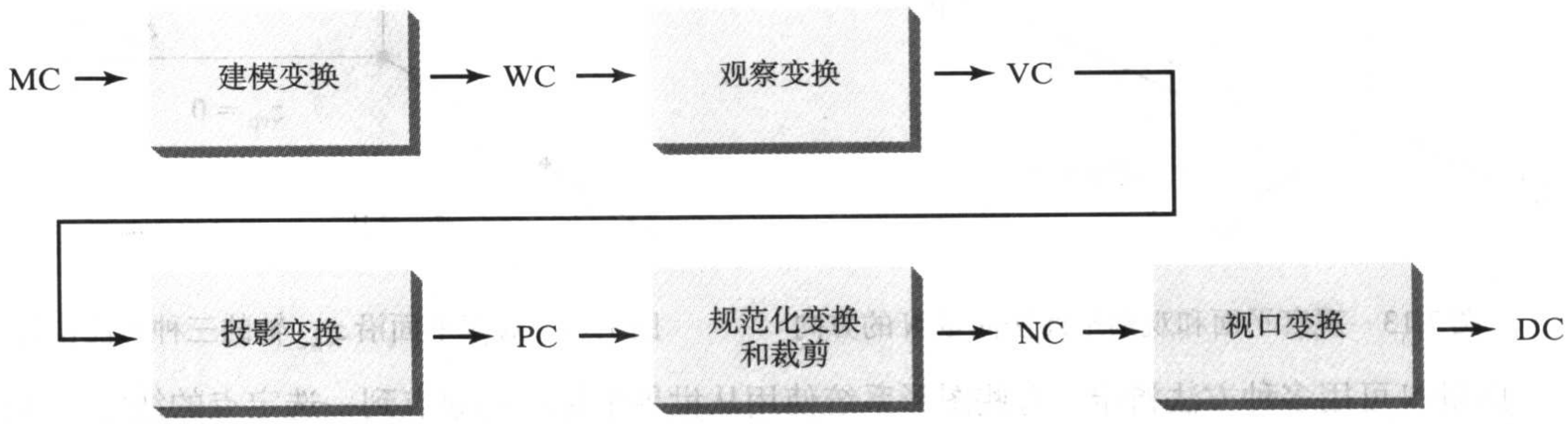


图7.11 一般的三维变换流水线，从建模坐标到世界坐标到观察坐标到投影坐标到规范化坐标最后到设备坐标

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

与在二维观察中一样，视口边界可在规范化坐标系或设备坐标系中指定。依赖于观察算法，我们假定视口在设备坐标系中指定，而规范化坐标已在裁剪后转换为视口坐标。还有另外一些必须完成的任务，如识别可见面和表面绘制。最后一步是将观察坐标映射到设备坐标系的指定显示窗口中。由于设备坐标系有时使用左手系方式，所以从显示屏幕出发的正向距离可用做场景的深度。

7.3 三维观察坐标系参数

建立一个三维观察坐标系与6.2节中讨论的建立二维观察坐标系类似。首先在世界坐标系中选定一点 $\mathbf{P}_0 = (x_0, y_0, z_0)$作为观察原点，称为**观察点**（view point）或**观察位置**（view position）。有时观察点也称为视点（eye position）或照相机位置（camera position）。再指定定义 y_{view} 方向的观察**向上向量**（view-up-vector）**V**。对三维空间还需要另外两个坐标轴的方向。这一般通过将观察方向用做 z_{view} 轴来实现。图 7.12 给出了在世界坐标系对一个三维观察坐标系统的定位。

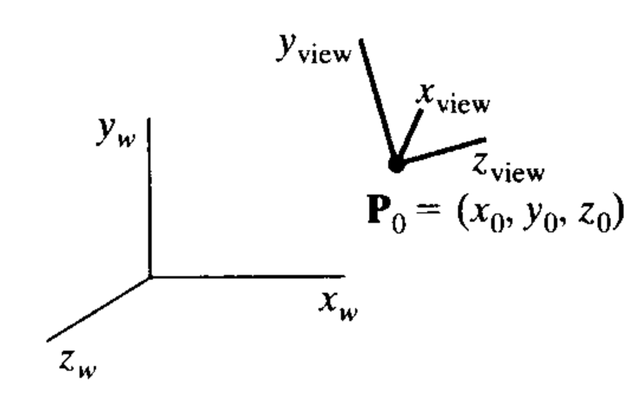


图 7.12 右手观察坐标系，其轴 x_{view}、y_{view}、z_{view} 相对于右手世界坐标系

7.3.1 观察平面法向量

因为观察方向通常沿着 z_{view} 轴，因此**观察平面**（view plane）有时也称为**投影平面**（projection plane），一般假设为与该轴垂直。这样，观察平面的方向及正 z_{view} 轴可定义为**观察平面法向量**（view-plane normal vector）**N**，如图 7.13 所示。

另外使用一个标量参数设定观察平面在沿 z_{view} 轴方向的位置 z_{vp}，如图 7.14 所示。该参数通常指定为从观察原点沿观察方向到观察平面的距离，它常常在负 z_{view} 方向。这样，观察平面总是与 $x_{view}y_{view}$ 平面平行，对象到观察平面的投影与场景在输出设备上的显示相对应。

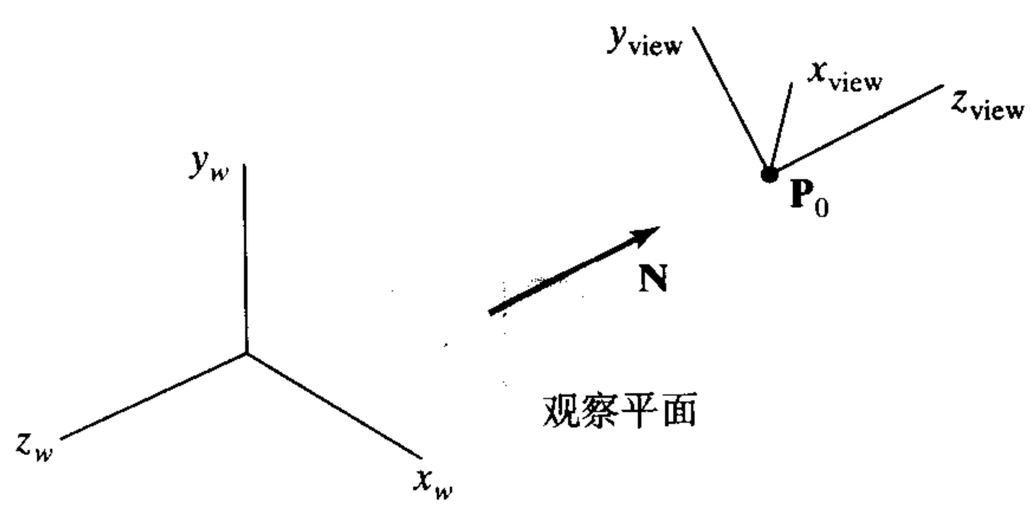


图 7.13 观察平面和观察平面法向量 **N** 的方向

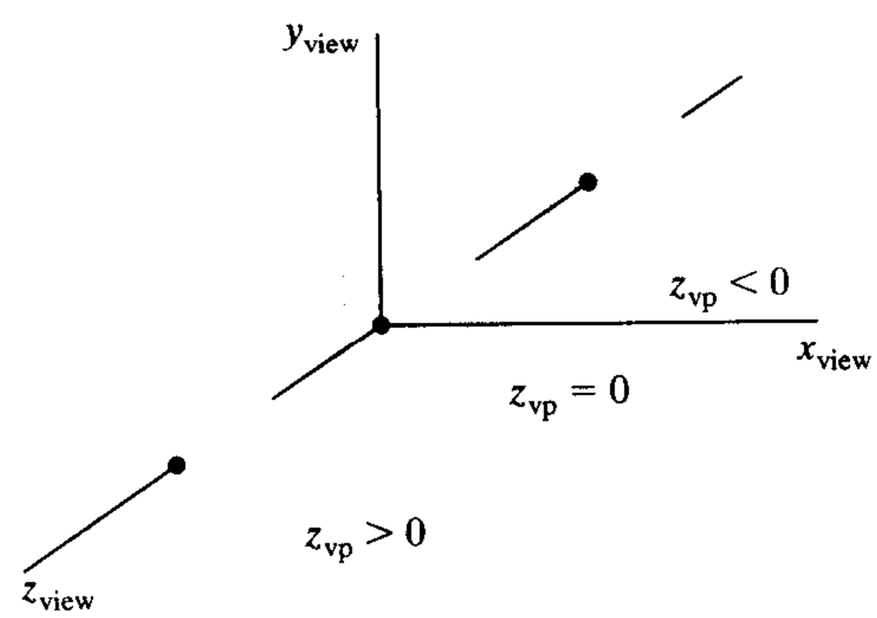


图 7.14 观察平面沿 z_{view} 轴的三种可能位置

向量 **N** 可用多种方法指定。有些图形系统使用从世界坐标系的原点到一选定点的线定义 **N** 的方向。另一些系统以参考点 $\mathbf{P}_{ref}$ 到观察原点 $\mathbf{P}_0$ 的方向赋给 **N**（参见图 7.15）。此时，参考点称为场景中的注视点（look-at point），观察方向为 **N** 的反向。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

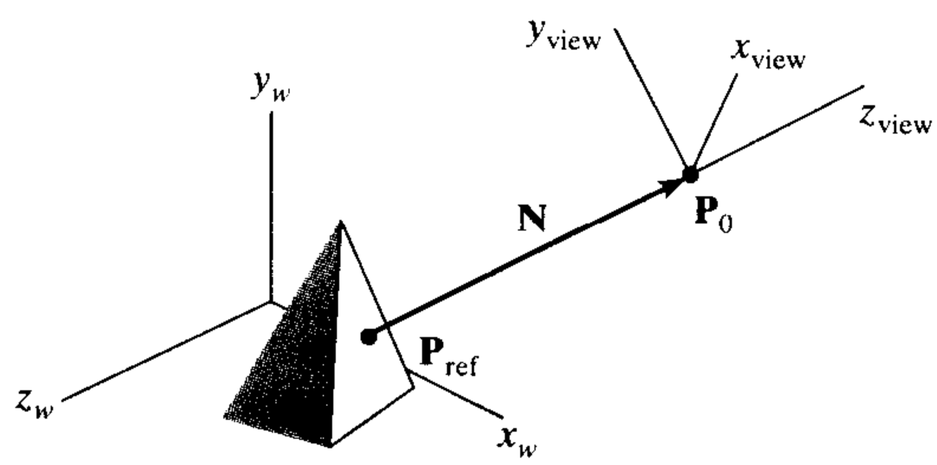


图 7.15 用选定的参考点 $\mathbf{P}_{ref}$ 到观察坐标系统原点 $\mathbf{P}_0$ 的方向指定观察平面法向量 **N**

也可以使用方向角来定义观察平面法向量和其他向量方向。即使用一直线分别与 x、y 和 z 轴间的三个夹角 α、β 和 γ。但指定场景中的两个点的位置通常比指定角度更容易。

7.3.2 观察向上向量

在选定观察平面法向量 **N** 后，可接着选择观察向上向量 **V**。该向量用来确定 y_{view} 轴正向。

通常，通过选定世界坐标系中的一个相关位置来定义 **V**，因此观察向上向量是从世界坐标系原点到该选定点。由于观察平面法向量 **N** 定义 z_{view} 轴方向，向量 **V** 应该与之垂直。但是，确定与 **N** 精确垂直的方向 **V** 是很困难的。因此，观察子程序一般要如图 7.16 所示那样调整向量 **V** 的方向，将其投影到观察平面上得到与观察平面法向量垂直的向量。

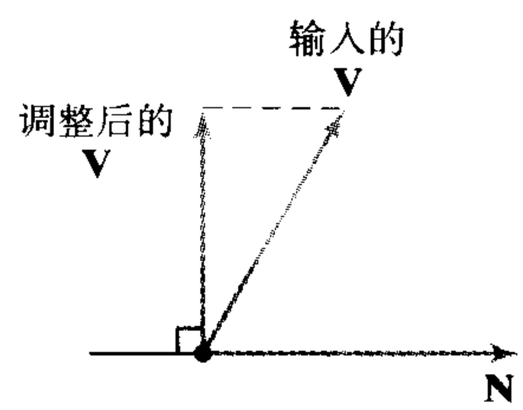


图 7.16 调整观察向上向量 **V** 的输入位置使其与法向量 **N** 垂直

我们可以选择任意的观察向上向量 **V**，只要它与 **N** 不平行。一种方便的选择是使用平行于世界坐标系轴 y_w 的方向，即设 $\mathbf{V} = (0, 1, 0)$。

7.3.3 uvn 观察坐标系统

有些图形软件包中使用观察方向为 z_{view} 正向的左手观察坐标系统。使用左手系统时，z_{view} 的增加方向看成是沿视线远离观察位置。但右手观察系统也很普遍，因为它与世界坐标系有相同的方向。这使得图形软件包只要按同一个坐标系方向处理世界坐标系和观察坐标系。尽管早期的图形软件包在左手系中定义观察坐标，但现在的图形标准中使用右手观察坐标系。而左手坐标轴常常用于描述屏幕坐标系统及规范化变换。

由于观察平面法向量 **N** 定义 z_{view} 轴方向且观察向上向量 **V** 用来获得 y_{view} 轴方向，我们只要再确定 x_{view} 轴方向。使用 **N** 和 **V** 的输入值，可以计算与 **N** 和 **V** 都垂直的第三个向量 **U**。向量 **U** 就定义了正 x_{view} 轴的方向。计算 **V** 和 **N** 的叉积得到正确的 **U** 方向并形成右手观察系统。**N** 和 **U** 的叉积可生成与 **N** 和 **U** 都正交的调整后沿正向 y_{view} 轴的 **V** 值。执行这些过程后，我们可得右手观察坐标系统的一组单位轴向量。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_x, n_y, n_z)$$

$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V}|} = (u_x, u_y, u_z) \tag{7.1}$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x, v_y, v_z)$$

使用这些单位向量形成的坐标系统常称为uvn**观察坐标参考系统**（uvn viewing-coordinate reference frame），请参见图7.17。

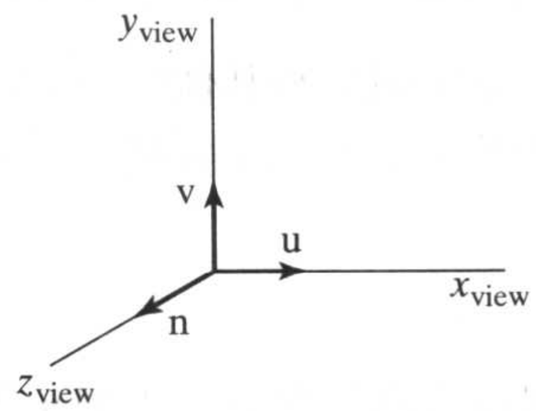


图7.17 使用单位向量**u**、**v**和**n**定义的右手观察坐标系统

7.3.4 生成三维观察效果

通过改变观察参数，可以得到场景中对象的多种视图。例如，固定观察位置时，通过改变**N**的方向可以显示围绕观察坐标原点的对象。改变**N**还可以建立包含从固定照相机位置的多视图的组合显示。图7.18为虚拟环境给出了一个广角显示。该广角视图由场景的同一视点但观察方向略有偏移的7个视图组合而成。同样，移动观察方向可生成立体视图。但此时还需要移动观察点来模拟两个眼睛的位置。

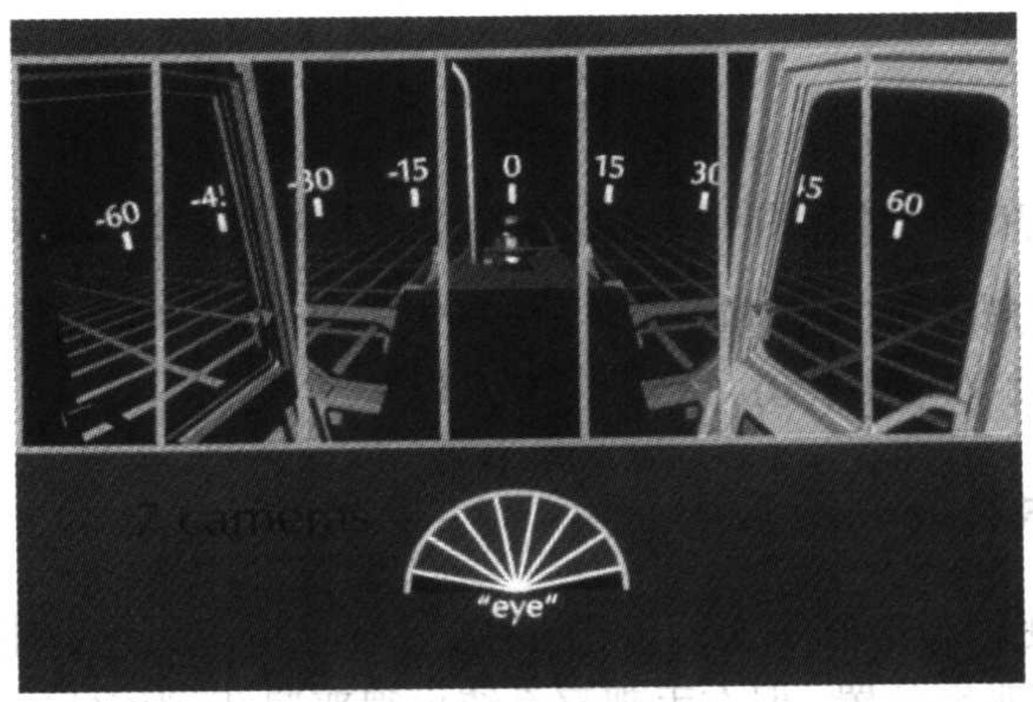


图7.18 用7个部分组合生成的虚拟环境广角显示图，每个视图的观察方向有少许差别（伊利诺伊大学UIUC巨型机应用国家中心提供）

在交互式应用中，法向量**N**是一个经常变动的观察参数。当然，在修改**N**的方向时，也要修改其他轴向量从而维护右手观察坐标系统。

如果要模拟运动的移镜效果，就像照相机移过场景或跟随一个移过场景的对象一样，需要在移动观察点时保持**N**的方向固定，如图7.19所示。为了显示一个对象的不同视图如侧视图和前视图，必须绕对象移动观察点，如图7.20所示。作为替代方法，一个对象或一组对象的多个视图可使用几何变换来生成而不需改变观察参数。

图7.21给出了为选择观察参数值而开发的一个界面。该软件包括一个观察编辑器、多显示窗口的选择、菜单和其他接口工具。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

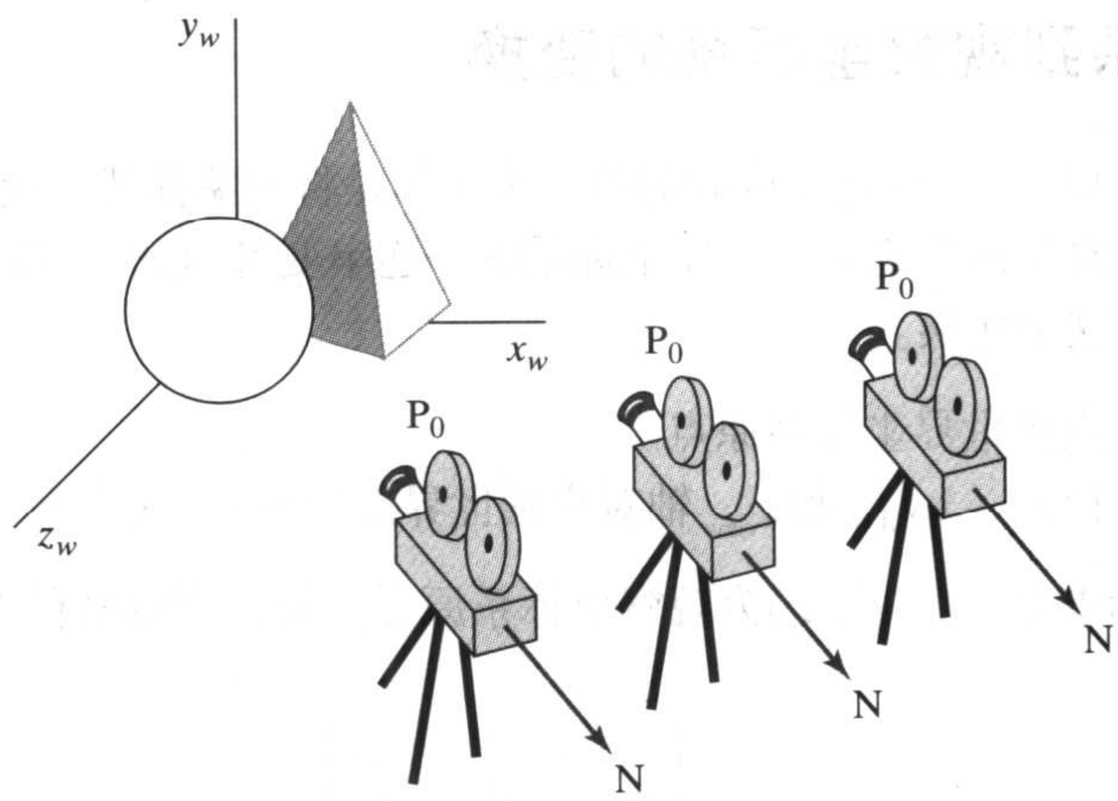


图 7.19 通过固定 **N** 的方向、改变观察参考点位置而生成移镜效果

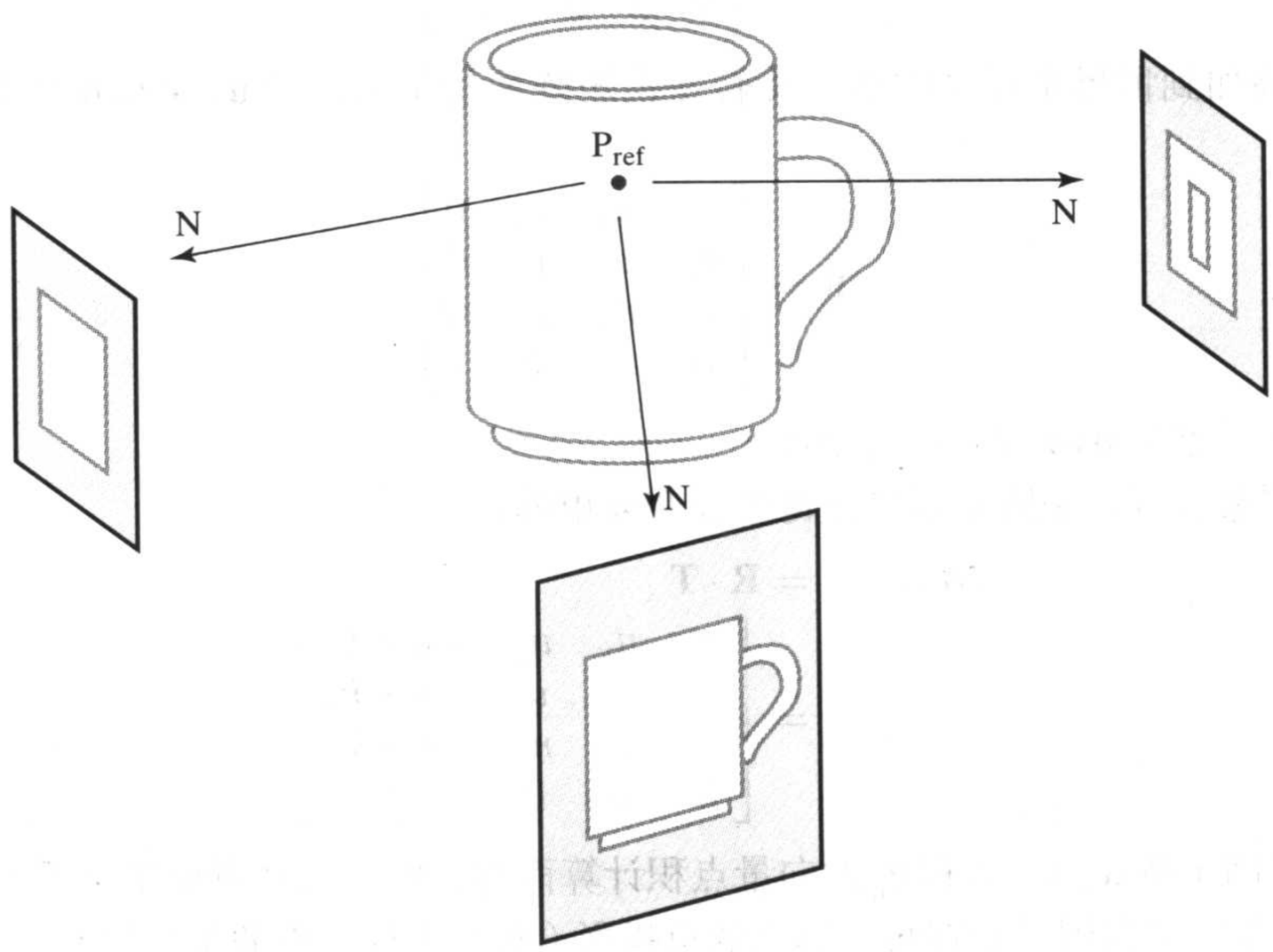


图 7.20 以固定观察参考点从不同方向观察一场景

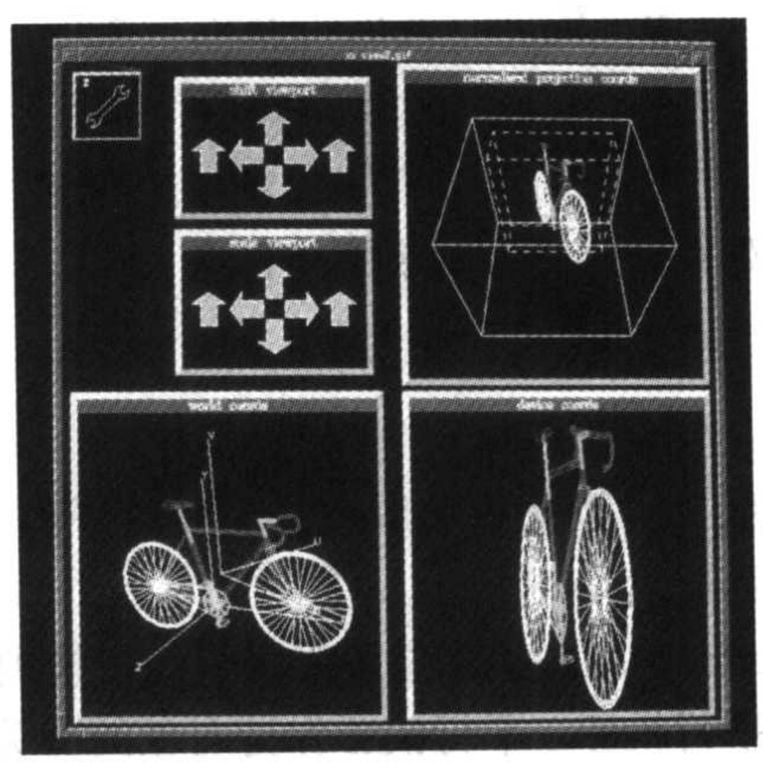


图 7.21 曼彻斯特大学使用PHIGS工具包开发的用于控制观察参数的交互界面（T. L. J. Howard、J. G. Williams 和 W. T. Hewitt 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

7.4 世界坐标系到观察坐标系的变换

在三维观察流水线中，场景构造完成后的第一步工作是将对象描述变换到观察坐标系中。对象描述的转换等价于将观察坐标系叠加到世界坐标系的一连串变换。我们可以使用5.15节叙述的坐标系间的变换方法来实现这个转换。

1. 平移观察坐标原点到世界坐标系原点。
2. 进行旋转，分别让 x_{view}、y_{view} 和 z_{view} 轴对应到世界坐标的 x_w、y_w、z_w 轴。

如果指定世界坐标点 $\mathbf{P} = (x_0, y_0, z_0)$ 为观察坐标系原点，则将观察坐标系原点移到世界坐标系原点的变换是

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.2}$$

将观察坐标系叠加到世界坐标系的组合旋转变换矩阵使用单位向量 **u**、**v** 和 **n** 来形成。该变换矩阵为

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.3}$$

这里，矩阵 **R** 的元素是 **uvn** 轴向量的分量。

将前面的平移和旋转矩阵乘起来获得坐标变换矩阵：

$$\begin{aligned} \mathbf{M}_{WC,\,VC} &= \mathbf{R} \cdot \mathbf{T} \\ &= \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_0 \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_0 \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{7.4}$$

该矩阵中的平移因子按 **u**、**v**、**n** 和 $\mathbf{P}_0$ 的向量点积计算而得，$\mathbf{P}_0$ 代表从世界坐标系原点到观察原点的向量。换句话说，平移因子是在每一轴上的负投影（观察坐标系中的负分量 $\mathbf{P}_0$）。这些矩阵元素的取值为

$$\begin{aligned} -\mathbf{u} \cdot \mathbf{P}_0 &= -x_0 u_x - y_0 u_y - z_0 u_z \\ -\mathbf{v} \cdot \mathbf{P}_0 &= -x_0 v_x - y_0 v_y - z_0 v_z \\ -\mathbf{n} \cdot \mathbf{P}_0 &= -x_0 n_x - y_0 n_y - z_0 n_z \end{aligned} \tag{7.5}$$

矩阵（7.4）将世界坐标系中的对象描述变换到观察坐标系。

7.5 投影变换

对象描述变换到观察坐标后，下一阶段是将其投影到观察平面上。图形软件一般都支持平行投影和透视投影两种方式。

在**平行投影**（parallel projection）中，坐标位置沿平行线变换到观察平面上。图 7.22 给出了用端点坐标 $\mathbf{P}_1$ 和 $\mathbf{P}_2$ 描述的线段的平行投影。平行投影保持对象的有关比例不变，这是三维对象计算机辅助绘图和设计中产生成比例工程图的方法。场景中的平行线在平行投影中显示成平行的。一般有两种获得对象平行视图的方法：沿垂直于观察平面的直线投影，或沿某倾斜角度投影到观察平面。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

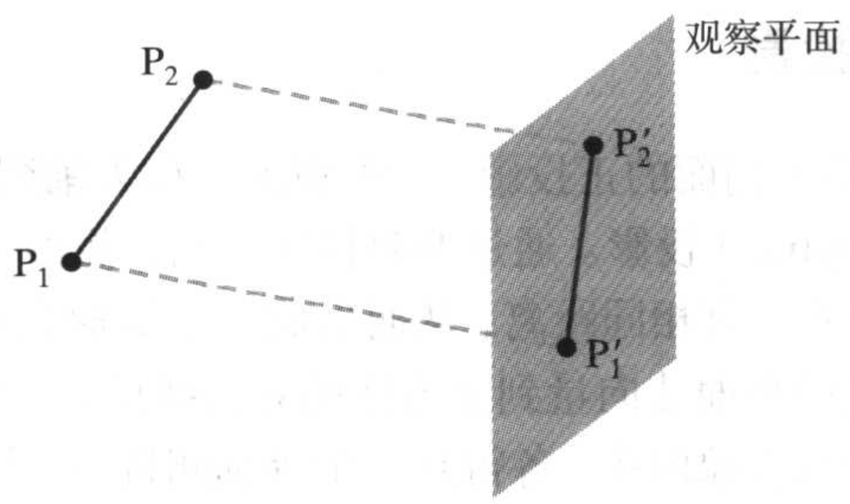


图 7.22 线段到观察平面的平行投影

在**透视投影**（perspective projection）中，对象位置沿会聚到观察平面后一点的直线变换到投影坐标系。图7.23给出了用端点坐标$\mathbf{P}_1$和$\mathbf{P}_2$描述的线段的透视投影。与平行投影不同的是，透视投影不保持对象的相关比例。但场景的透视投影真实感较好，因为在透视显示中较远的对象减小了尺寸。

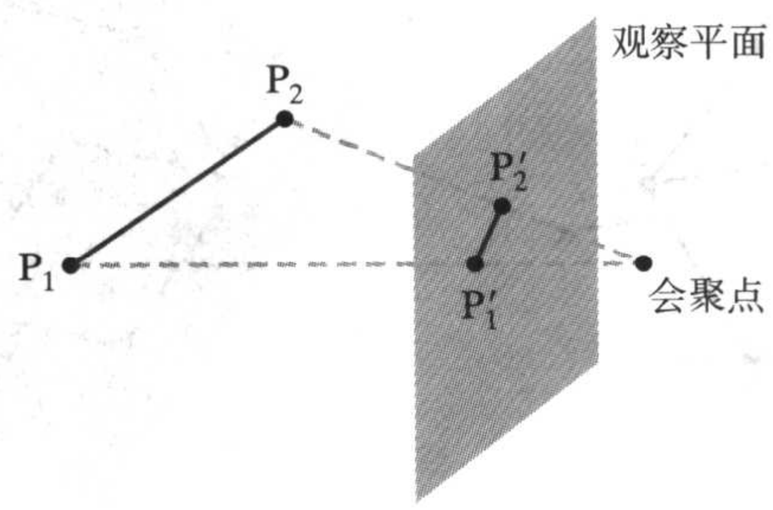


图 7.23 线段到观察平面的透视投影

7.6 正投影

对象描述沿与投影平面法向量平行的方向到投影平面上的变换称为**正投影**（orthogonal projection；或**正交投影**，orthographic projection）。这生成一个平行投影变换，其中投影线与投影平面垂直。正投影常常用来生成对象的前视图、侧视图和顶视图，如图7.24所示。前、侧和后方向的正投影称为立面图（elevation），顶部正投影称为平面图（plan view）。工程和建筑绘图通常使用正投影，因为可以精确地绘出长度和角度，并能从图中测量出这些值。

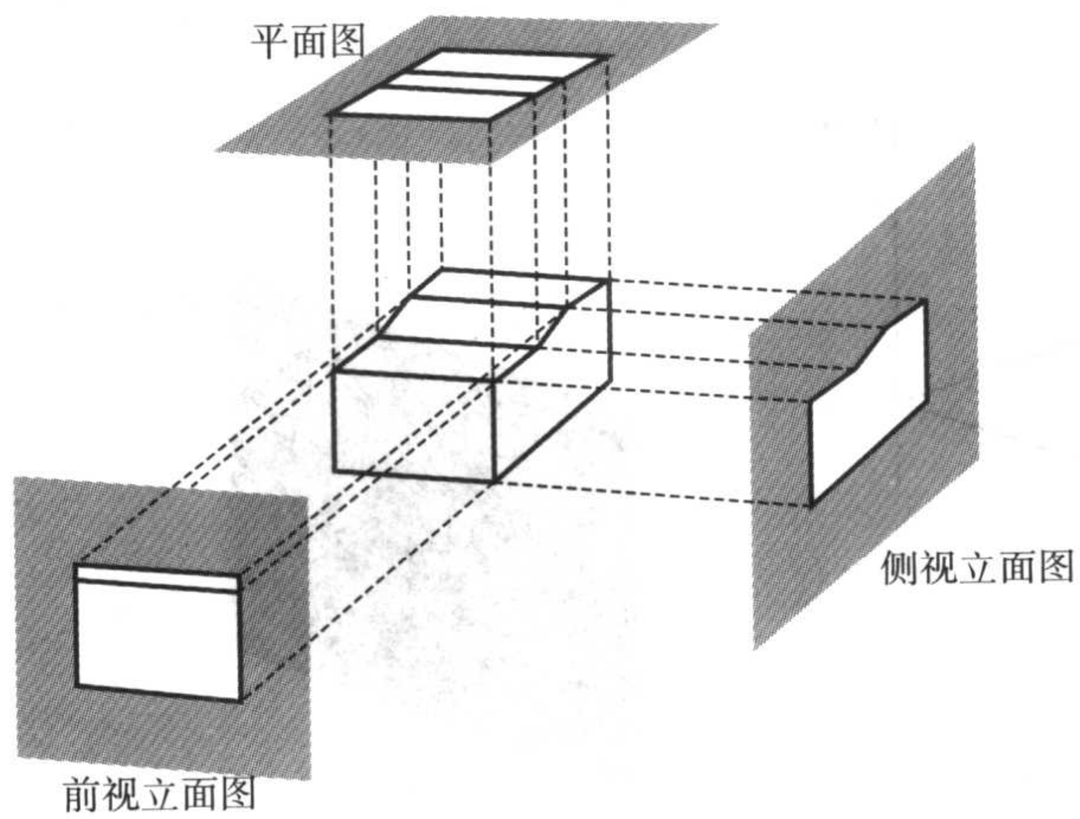


图 7.24 对象的正投影，显示了平面图和立面图

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

7.6.1 轴测和等轴测正投影

我们也能形成显示对象多个侧面的正投影。这些视图也称为**轴测**（axonometric）正投影。最常用的轴测投影是**等轴测**（isometric）投影。通过调整投影平面，使得与每个坐标轴（其中定义了对象，该轴称为主轴）的交点离原点有相同距离，从而生成一个等轴测投影。图7.25表示了立方体的一个等轴测投影。通过调整投影平面法向量到立方体的对角线位置，可以得到等轴测投影。一共有八个位置，每一个位置在一个八分象限中，各得到一个等轴测视图。所有的三个主轴在等轴测投影中缩短了相同的等级，从而保持相应的比例不变。不过，在三个主轴方向的缩放因子不同的一般轴测投影中，则不是这种情况。

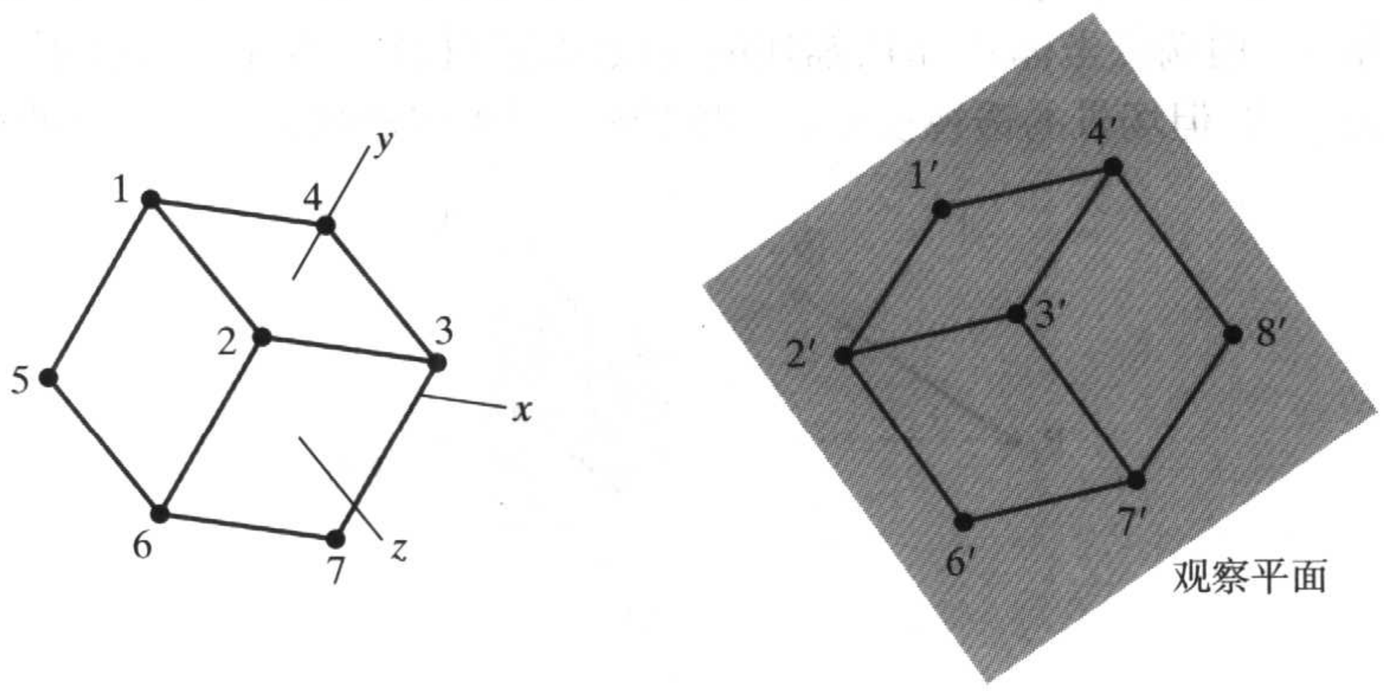


图 7.25 立方体的等轴测投影

7.6.2 正投影坐标系

如果投影方向平行于z_{view}轴，则正投影的变换公式很简单。如图7.26所示，观察坐标系中任一位置(x, y, z)的投影坐标为

$$x_p = x, \qquad y_p = y \tag{7.6}$$

任何投影变换中z坐标值被保存，用于可见性检测过程。而场景中每一个三维坐标点转换到了规范化空间。

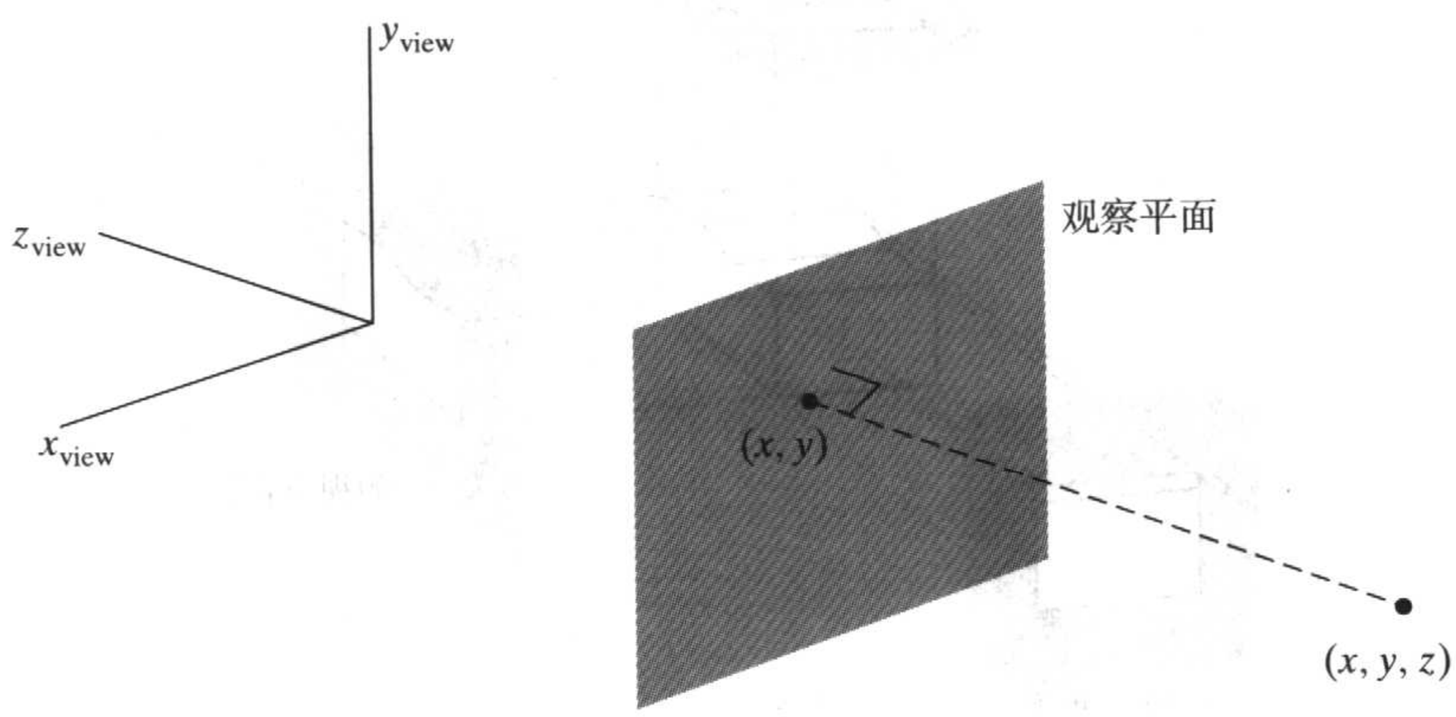


图 7.26 一个空间点到观察平面的正交投影

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

7.6.3　裁剪窗口和正投影观察体

模拟照相机时，镜头的类型是确定有多少场景变换到胶片上的一个因素。广角镜头摄入比一般镜头更多的场景。对于计算机图形应用而言，使用矩形的裁剪窗口来实现这一目标。在二维应用中，图形软件包一般仅允许标准位置的裁剪矩形。因此，我们像在二维观察中一样，在三维观察中通过选择二维坐标位置左下角和右上角来建立裁剪窗口。在三维观察坐标系中，裁剪窗口定位在观察平面上，其边 x_{view} 与 y_{view} 和轴平行，如图 7.27 所示。如果要使用其他形状和方向的裁剪窗口，则必须开发自己的观察过程。

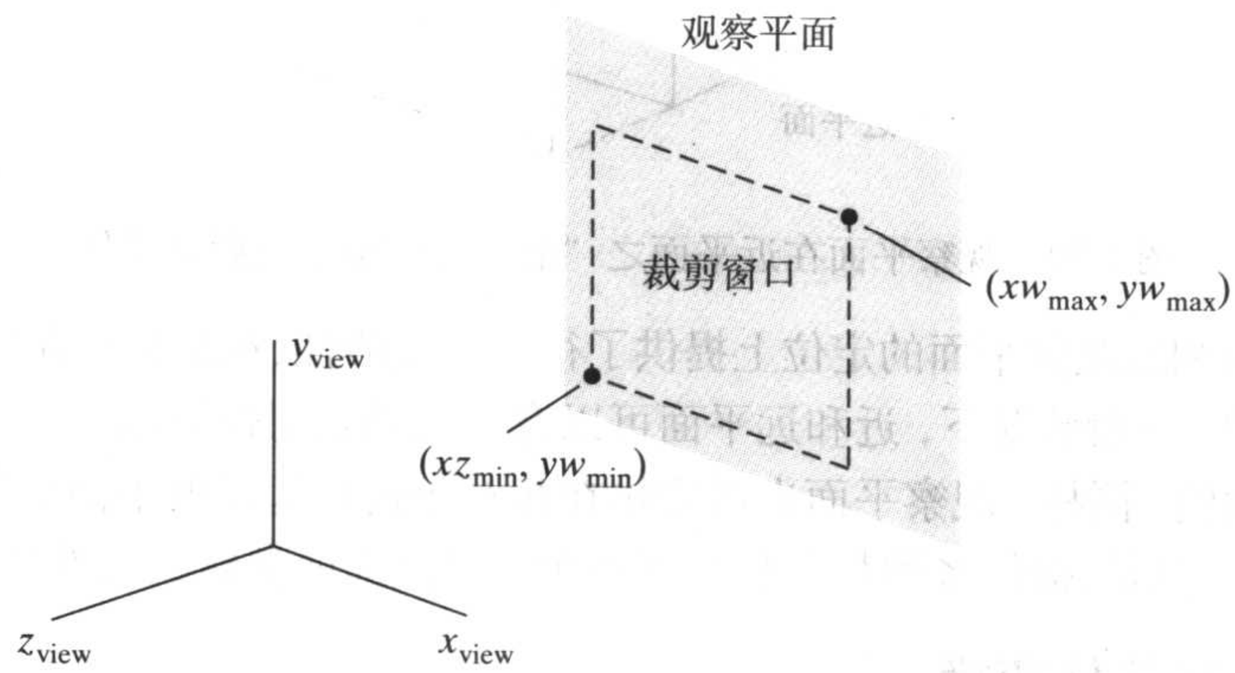
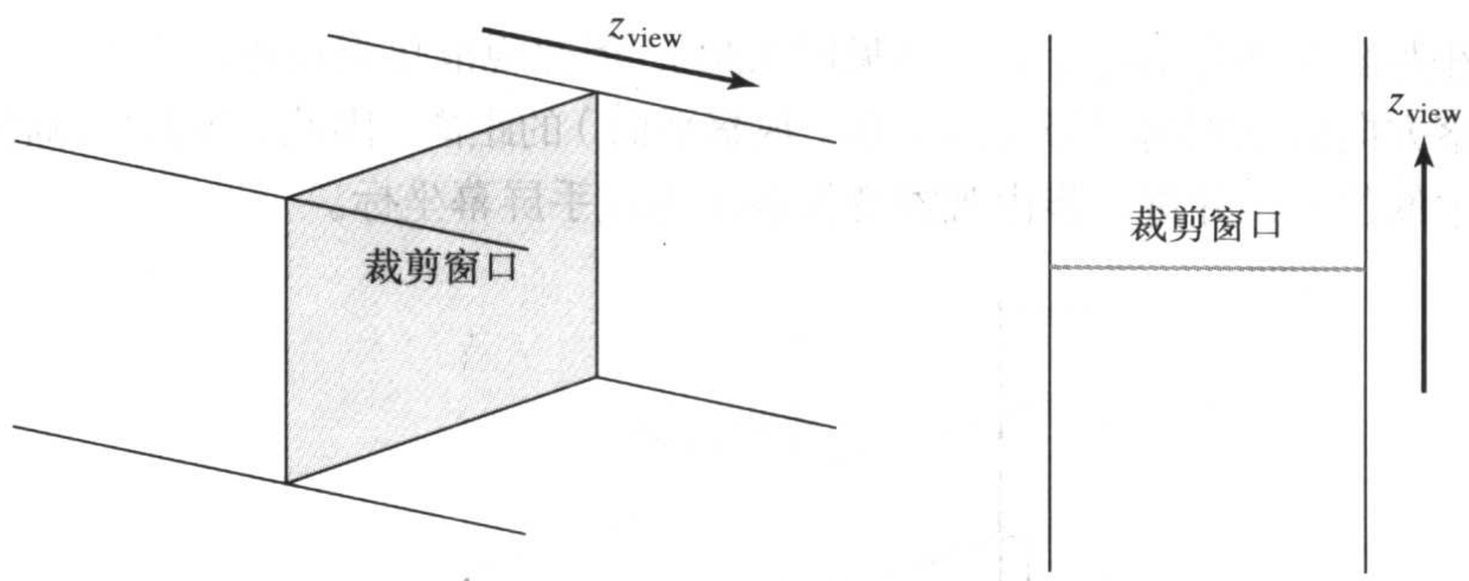


图 7.27　在观察平面上用观察坐标给出其最小和最大坐标值的裁剪窗口

裁剪窗口的边指定了要显示的场景部分的 x 和 y 范围限制。该限制形成了称为**正投影观察体**（orthogonal-projection view volume）的裁剪区域的上、下和两侧。由于投影线与观察平面垂直，因此这四个边界是和观察平面也垂直的平面，并且它们经过裁剪窗口的边而形成无限的裁剪区域，如图 7.28 所示。



(a) 侧视图　　(b) 顶视图

图 7.28　无限的正投影观察体

我们可以通过选择平行于观察平面的一个或两个边界平面来为正交观察体的 z_{view} 方向限定边界。这两个平面称为**近–远裁剪平面**（near-far clipping plane），或**前–后裁剪平面**（front-back clipping plane）。近和远平面使我们能把要显示场景前面的和后面的对象排除。当观察方向是沿 z_{view} 轴负向时，通常有 $z_{far} < z_{near}$，因此远平面在负 z_{view} 轴的远处。某些图形库将这两个平面作为选择项，而其他库则要求它们不能为默认的。指定了近和远平面，就有了一个有限的正交观察体，它是一个如图7.29所示的和观察平面一起定位的矩形平行管道（rectangular parallelepiped）。我们观察到的场景是在该观察体中的那些对象，在观察体外的场景部分均被裁剪掉。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

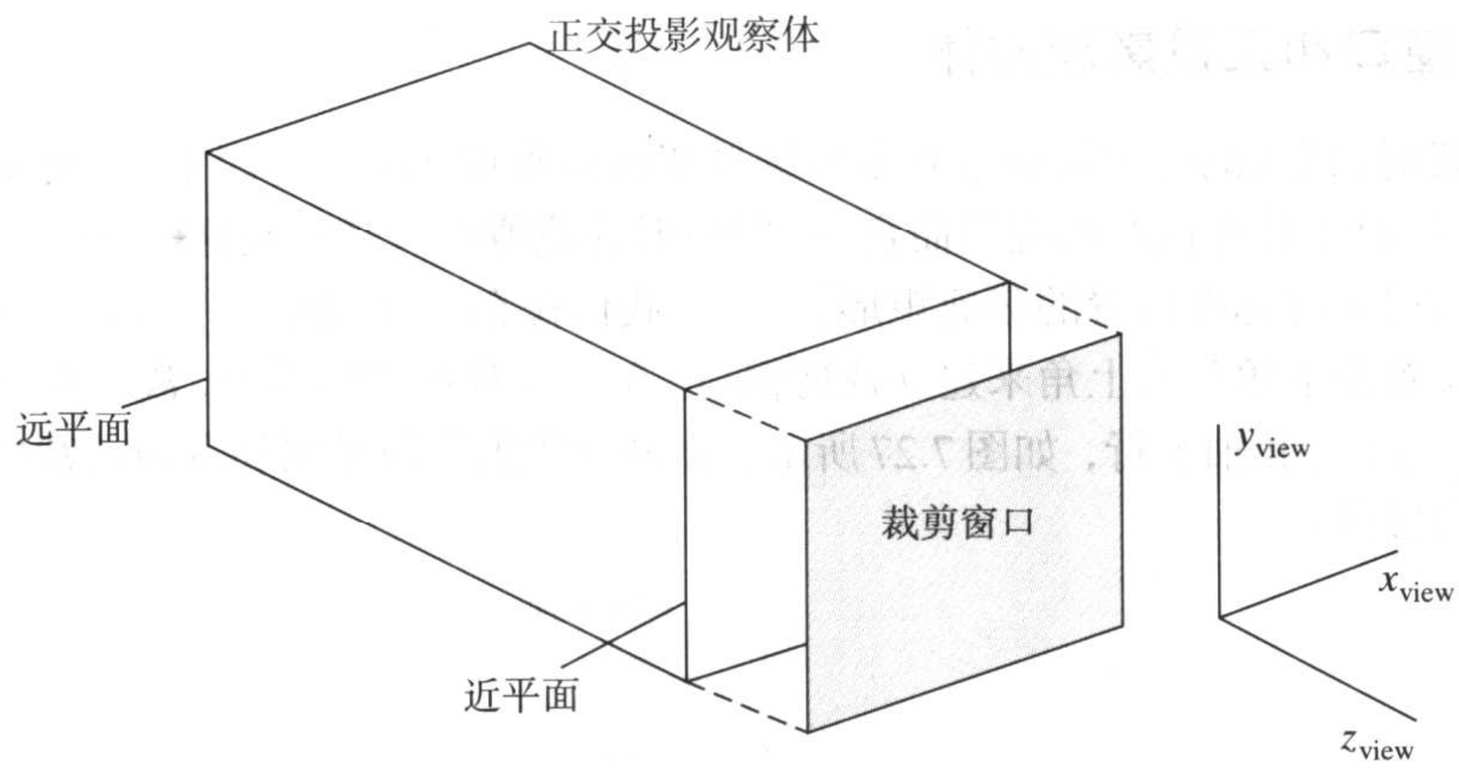


图 7.29　观察平面在近平面之“前”的有限正投影观察体

图形软件包在近和远裁剪平面的定位上提供了很大的灵活性，包括提供在场景其他位置指定附加的裁剪平面的选项。一般情况下，近和远平面可以有任意的相对位置以获得各种观察效果，包括安排在观察平面的两侧。同样，观察平面也可安排在相对于近和远裁剪平面的任意位置，尽管它常常与近裁剪平面一致。但是，提供多种裁剪和观察平面的定位选项常导致三维场景处理效率的下降。

7.6.4　正投影的规范化变换

使用从坐标位置到观察平面的正投影变换，任意一点(x, y, z)的投影位置是(x, y)。因此，在建立观察体的范围后，该矩形平行管道内部的坐标描述即为投影坐标，它们不需要另外的投影处理就可直接映射到**规范化观察体**（normalized view volume）。有些图形软件包使用单位立方体作为规范化观察体，其x、y和z坐标规范成0到1之间。另外的规范化变换方法使用坐标范围从–1到1的对称立方体。

由于屏幕坐标经常指定为左手系（参见图7.30），因此规范化观察体也常指定为左手系统。这样就可以将观察方向的正距离解释为离屏幕（观察平面）的距离。因此，我们可以将投影坐标转换为左手坐标系中的位置，并进一步由观察变换转换为左手屏幕坐标。

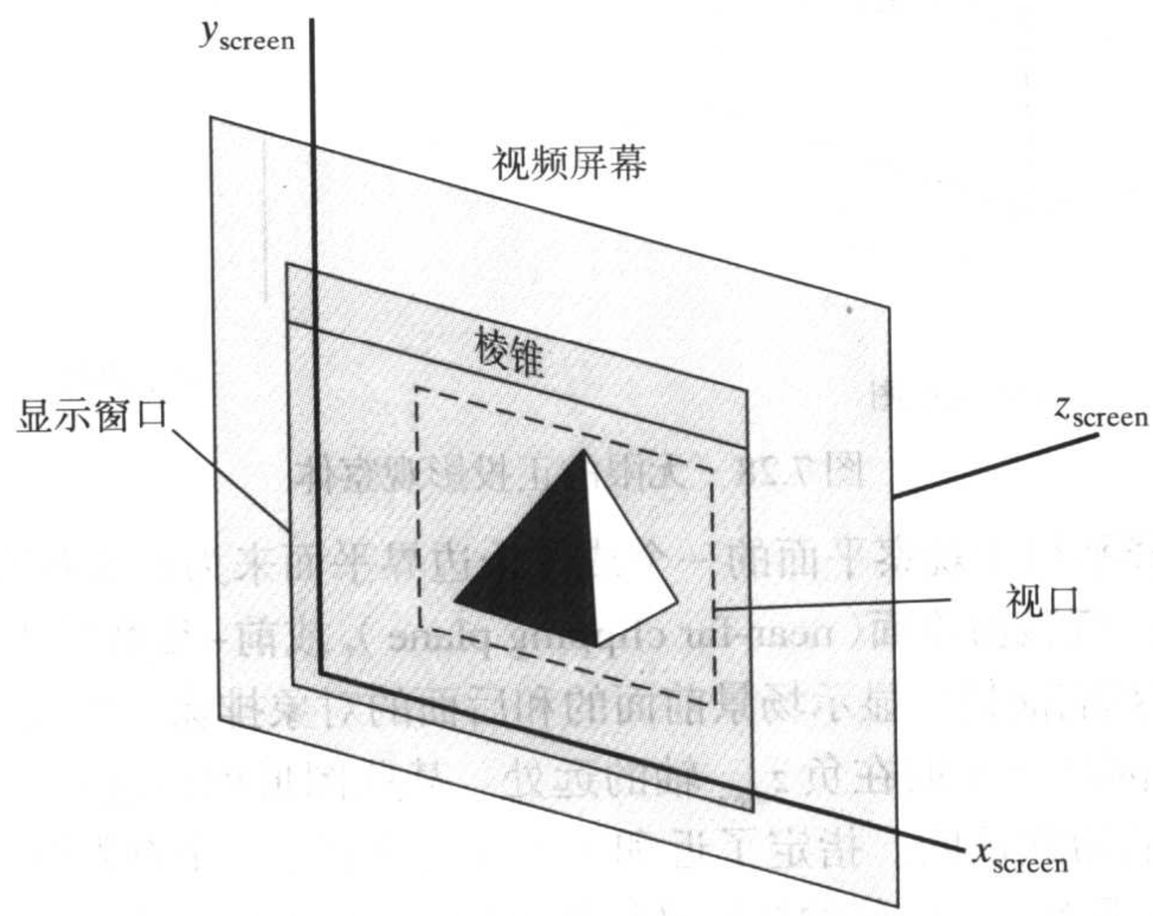


图 7.30　一个左手坐标系统

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

为了展示规范化变换，我们假定正投影观察体映射到左手坐标系的对称的规范化立方体。同样，近和远平面的z坐标分别用z_{near}和z_{far}来表示。图7.31给出了这个规范化变换。位置$(x_{min}, y_{min}, z_{near})$映射到规范化位置(−1, −1, −1)，而位置$(x_{max}, y_{max}, z_{far})$映射到(1, 1, 1)。

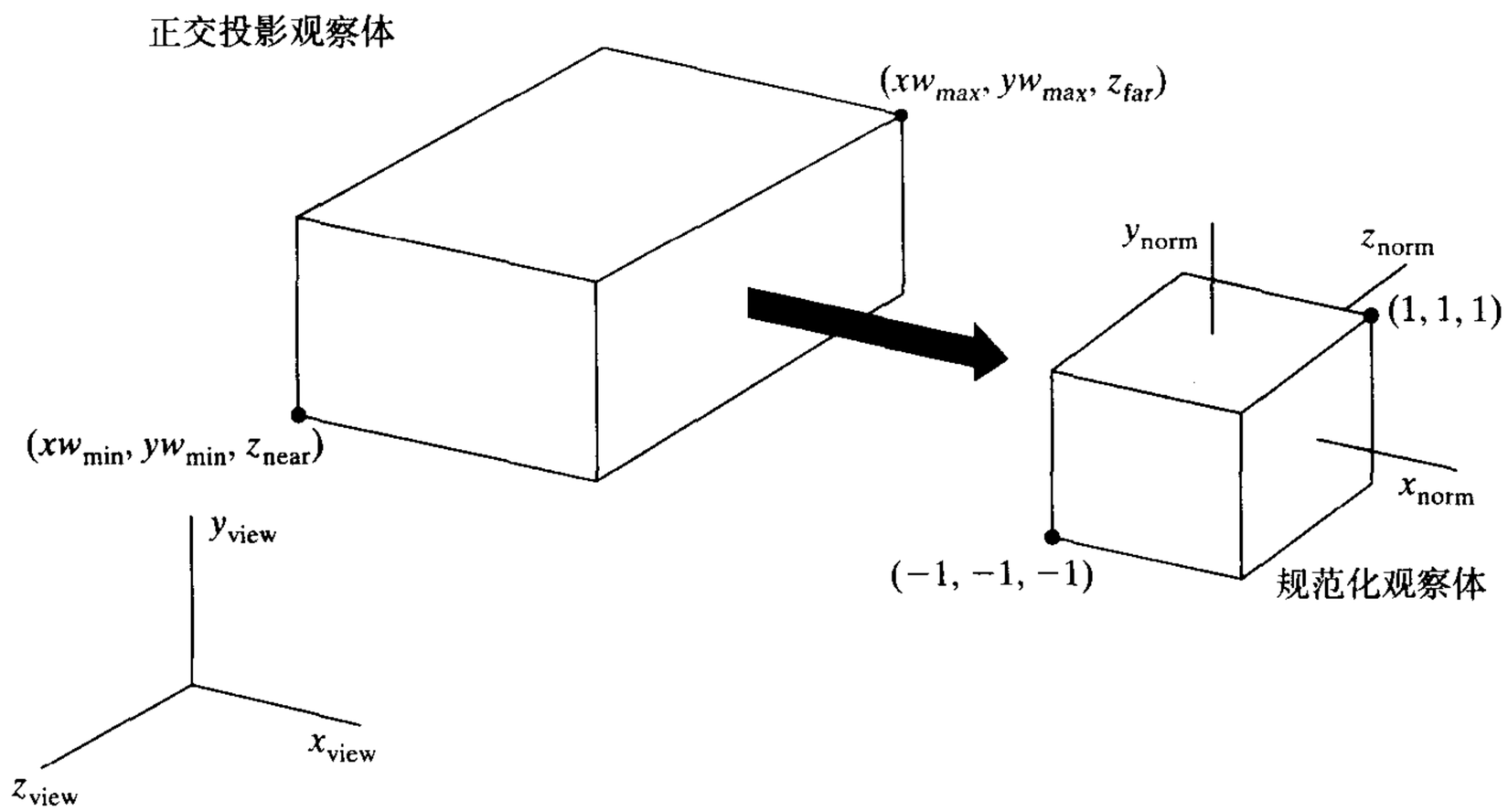


图 7.31　从正投影观察体到左手坐标系统中对称立方体的规范化变换

把矩形平行管道观察体变换到规范化立方体与6.3节中将裁剪窗口转换到规范化对称正方形的工作相类似。正投影观察体中的x和y位置的规范化变换由规范化矩阵（6.9）给出。另外，我们要用相同的计算将从z_{near}到z_{far}的z坐标变换成−1到1之间。因此，正投影观察体的规范化变换是

$$\mathbf{M}_{\text{ortho,norm}} = \begin{bmatrix} \frac{2}{xw_{max} - xw_{min}} & 0 & 0 & -\frac{xw_{max} + xw_{min}}{xw_{max} - xw_{min}} \\ 0 & \frac{2}{yw_{max} - yw_{min}} & 0 & -\frac{yw_{max} + yw_{min}}{yw_{max} - yw_{min}} \\ 0 & 0 & \frac{-2}{z_{near} - z_{far}} & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.7}$$

该矩阵从左边和组合观察变换 **R · T** 相乘（参见 7.4 节），得到完整的从世界坐标系到规范化正投影坐标系的变换。

因此，在规范化变换之后可高效地进行裁剪处理。在观察流水线的这个阶段，所有与设备无关的变换均已完成，并且已经合并到一个组合矩阵中。裁剪后，再应用可见性测试、表面绘制和视口变换来生成最后的场景屏幕显示。

7.7　斜投影

通常，场景的平行投影视图通过将对象描述沿投影线变换到观察平面来获得，投影线的方向和观察平面法向量之间的关系为任意的。当投影路径与观察平面不垂直时，该映射称为**斜平行投影**（oblique parallel projection）。使用这样的投影可生成对象的前视、顶视等视图的混合视图，如图 7.32 所示。斜平行投影用投影线的向量方向定义，它可用多种方法指定。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

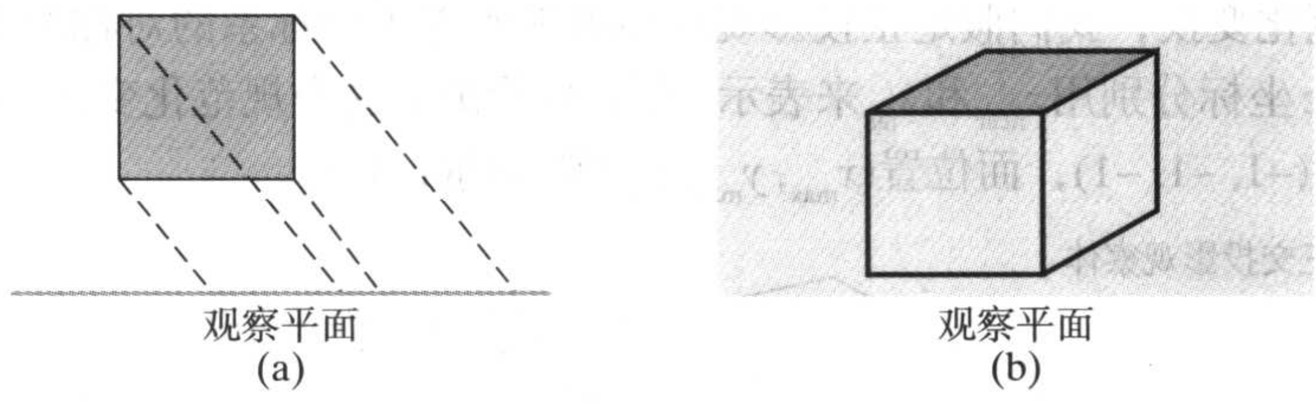


图 7.32　一个立方体的斜平行投影：(a)顶视图；(b)包含立方体的多个表面

7.7.1　绘图和设计中的斜平行投影

在工程与建筑设计应用中，斜平行投影常使用两个角度来描述，如图7.33中的α和ϕ。其中的空间位置(x, y, z)投影到位于观察z轴z_{vp}处的观察平面的(x_p, y_p, z_{vp})。位置(x, y, z_{vp})是相应的正投影点。从(x, y, z)到(x_p, y_p, z_{vp})的斜平行投影线与投影平面上连接(x_p, y_p, z_{vp})和(x, y, z_{vp})的线之间的夹角为α。观察平面上这条长度为L的线与投影平面水平方向的夹角为ϕ。角α可赋以0°到90°之间的值，而ϕ可以从0°到360°。用x、y、L和ϕ来表示投影坐标，如下所示：

$$\begin{aligned} x_p &= x + L\cos\phi \\ y_p &= y + L\sin\phi \end{aligned} \tag{7.8}$$

长度L依赖于角α及点(x, y, z)到观察平面的距离：

$$\tan\alpha = \frac{z_{vp} - z}{L} \tag{7.9}$$

因此

$$\begin{aligned} L &= \frac{z_{vp} - z}{\tan\alpha} \\ &= L_1(z_{vp} - z) \end{aligned} \tag{7.10}$$

这里$L_1 = \cot\alpha$，当$z_{vp} - z = 1$时就是L。斜平行投影方程（7.8）可写成

$$\begin{aligned} x_p &= x + L_1(z_{vp} - z)\cos\phi \\ y_p &= y + L_1(z_{vp} - z)\sin\phi \end{aligned} \tag{7.11}$$

在$L_1 = 0$时（在$\alpha = 90°$时发生）得到正投影。

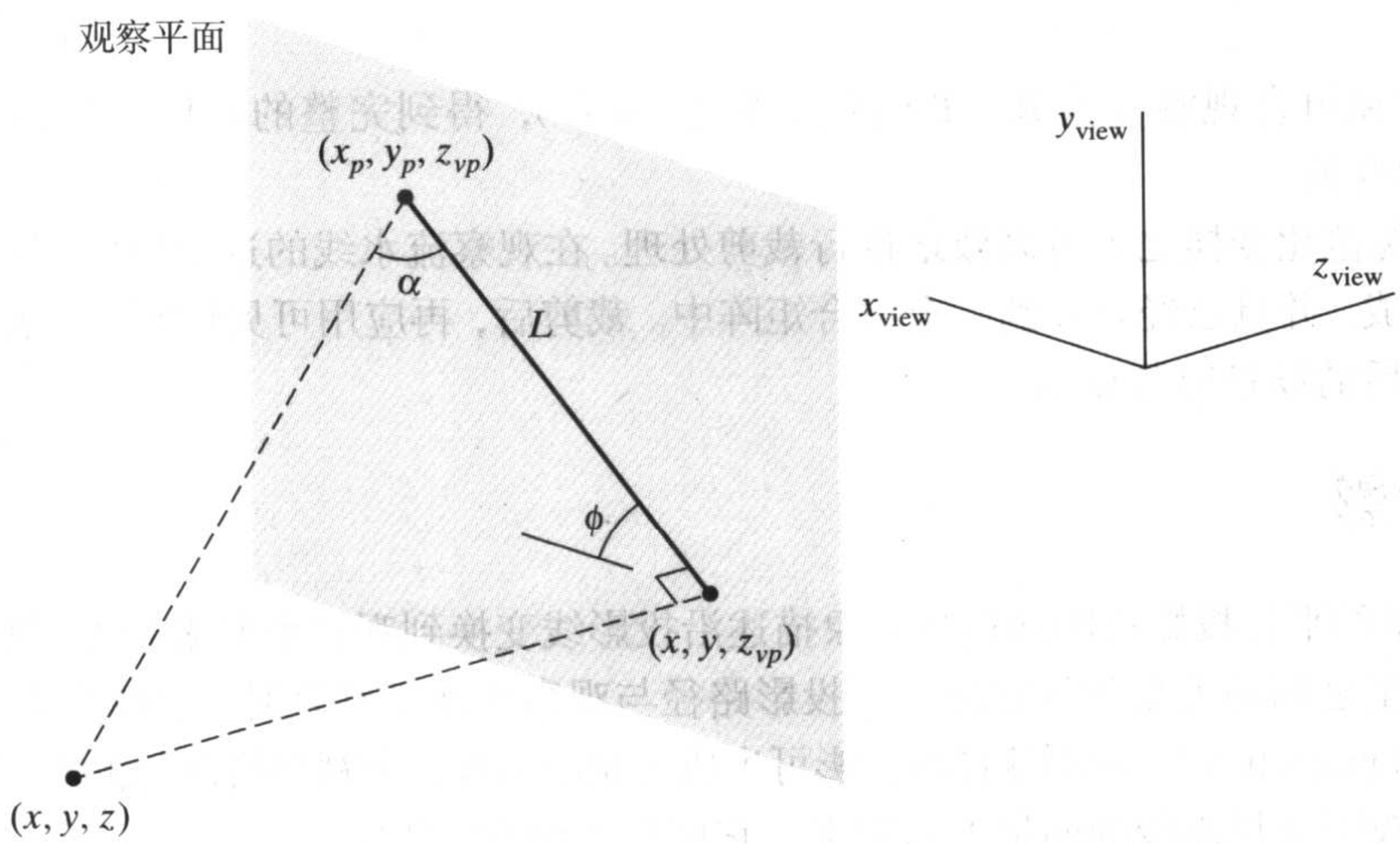


图 7.33　坐标位置(x, y, z)到z_{view}轴z_{vp}位置的投影平面上位置(x_p, y_p, z_{vp})的斜平行投影

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

方程（7.11）表达了 z 轴的错切变换（参见 5.14 节）。实际上，斜平行投影的效果是常数 z 的错切平面并将它们投影到观察平面上。每个常数 z 平面上位置 (x, y) 按该平面到观察平面距离的比例来移动，因此该平面上的角度、距离及平行线都将精确地投影。图 7.34 给出了这一效果，其中观察平面安排在立方体的前平面。立方体的后平面错切并与前平面在观察面上的投影重叠。立方体上连接前平面和后平面的侧棱线投影成长度为 L_1 的线，它与投影平面的水平线夹角为 ϕ。

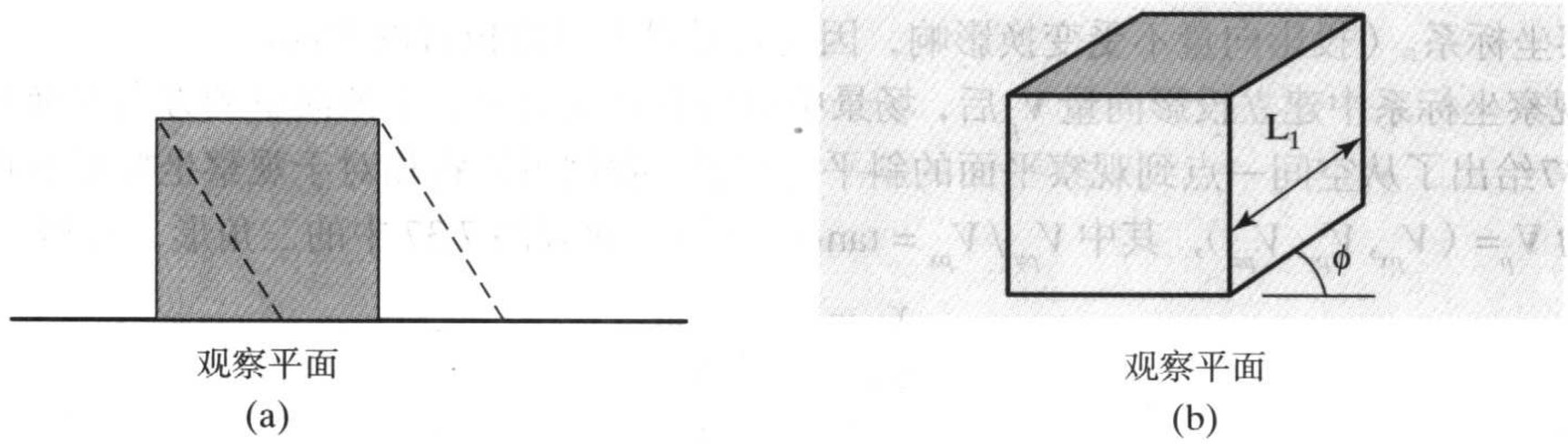


图 7.34　观察平面与立方体前平面一致时生成的：(a)立方体的斜投影（顶视图）；(b)前、侧和顶视图的混合显示

7.7.2　斜等测和斜二测斜平行投影

角度 ϕ 一般使用 30° 和 45°，显示对象前、侧和顶（或前、侧和底）视图的组合。α 的常用值为满足 $\tan\alpha = 1$ 和 $\tan\alpha = 2$ 的值。第一种情况，$\alpha = 45°$，获得的视图称为**斜等测**（cavalier）投影图。所有垂直于投影平面的线条投影后长度不变。图 7.35 给出了一个立方体的斜等测投影图的例子。

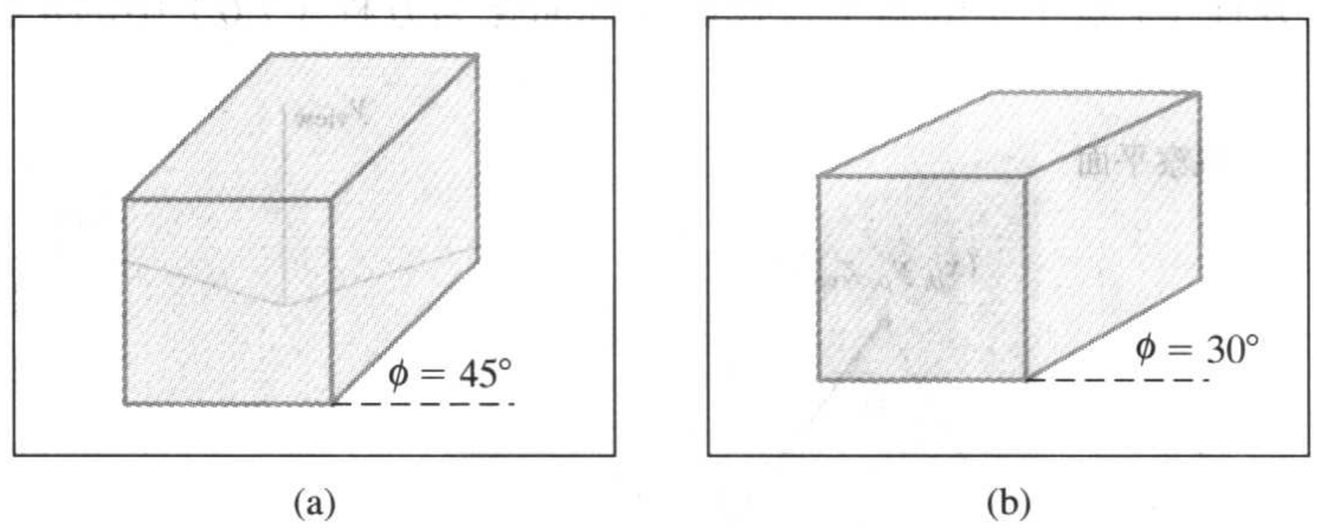


图 7.35　两种角度的斜等测立方体投影图。立方体深度投影与宽度及高度相等

当投影角 α 满足 $\tan\alpha = 2$ 时，生成的视图称为**斜二测**（cabinet）投影图。对于这样的角度（$\approx 63.4°$），垂直于观察平面的线条投影后得到一半长度。因为在垂直方向长度减半，使得斜二测投影看起来比等轴测的真实感好一些。图 7.36 给出了立方体的斜二测投影图的例子。

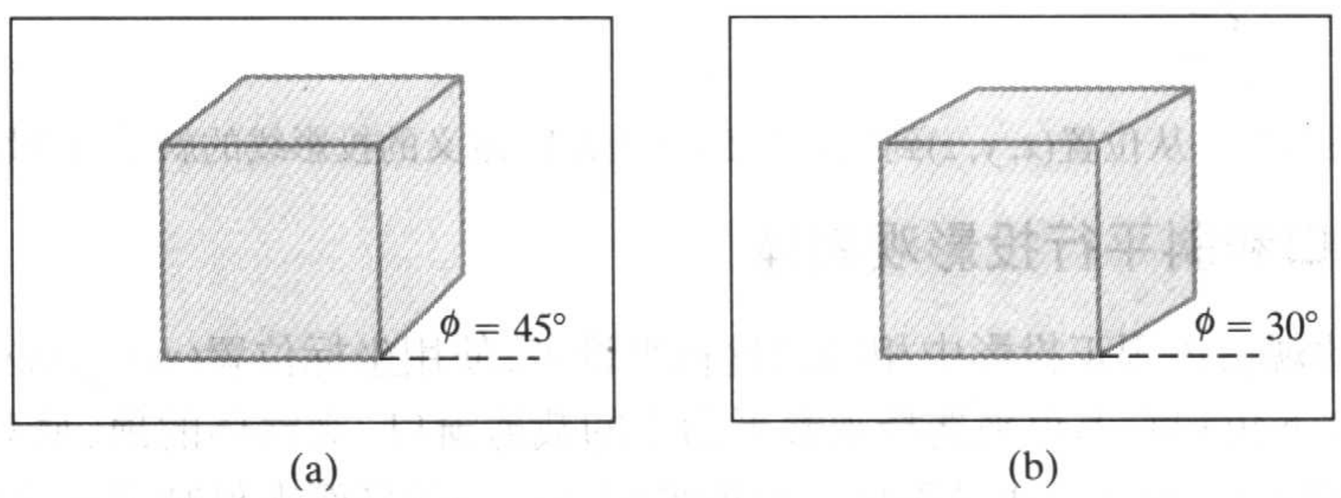


图 7.36　两种角度的斜二测立方体投影图。立方体的深度投影是宽度及高度的一半

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

7.7.3 斜平行投影向量

在支持斜平行投影的图形编程软件包中，到观察平面的投影方向用**平行投影向量**（parallel-projection vector）$\mathbf{V}_p$来描述。该方向向量可以由相对于观察点的一个参考位置来指定，如同指定观察平面法向量一样，或使用任何其他两点来指定。有些软件包使用相对于裁剪窗口中心的参考点来定义平行投影方向。如果投影向量在世界坐标系中指定，则必须先使用7.4节中的旋转矩阵将它变换到观察坐标系。（投影向量不受变换影响，因为它是没有固定位置的方向。）

在观察坐标系中建立投影向量$\mathbf{V}_p$后，场景中的所有点均沿平行于该向量的方向变换到观察平面。图7.37给出了从空间一点到观察平面的斜平行投影。我们可以将相对于观察坐标系的投影向量分量记为$\mathbf{V}_p=(V_{px}, V_{py}, V_{pz})$，其中$V_{py}/V_{px}=\tan\phi$。然后，比较图7.37中的三角形，可得

$$\frac{x_p - x}{z_{vp} - z} = \frac{V_{px}}{V_{pz}}$$

$$\frac{y_p - y}{z_{vp} - z} = \frac{V_{py}}{V_{pz}}$$

斜平行投影方程（7.11）可通过投影向量写出等价的公式：

$$\begin{aligned} x_p &= x + (z_{vp} - z)\frac{V_{px}}{V_{pz}} \\ y_p &= y + (z_{vp} - z)\frac{V_{py}}{V_{pz}} \end{aligned} \tag{7.12}$$

当$V_{px}=V_{py}=0$时，方程（7.12）中的斜平行投影坐标简化为方程（7.6）中的正投影坐标。

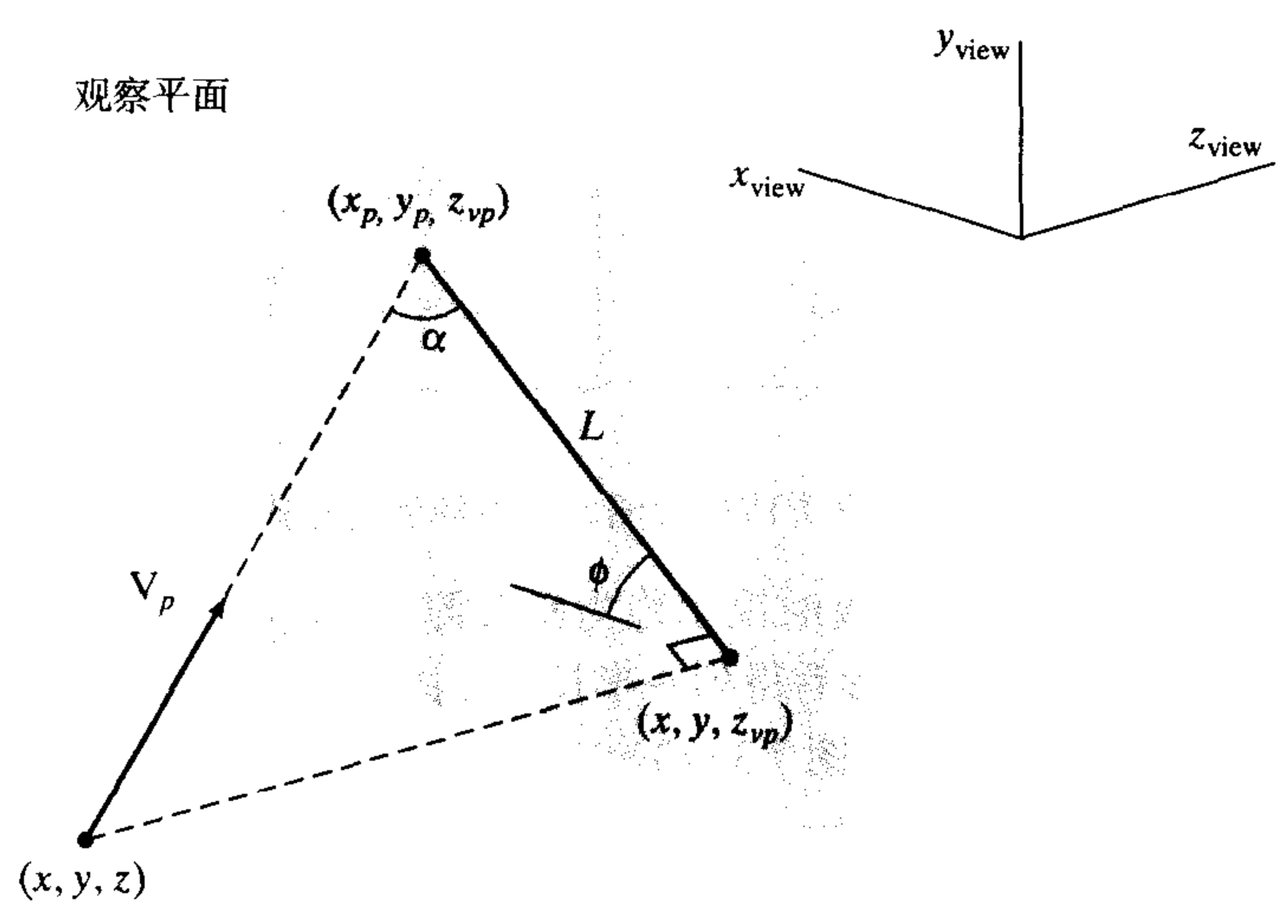


图7.37 从位置(x, y, z)到观察平面沿向量$\mathbf{V}_p$定义的投影线的斜平行投影

7.7.4 裁剪窗口和斜平行投影观察体

斜平行投影的观察体用正投影中相同的过程来设定。使用坐标位置(xw_{min}, yw_{min})和(xw_{max}, yw_{max})作为裁剪矩形的左下角和右上角来选择观察平面上的裁剪窗口。观察体的顶、底和两侧由投影方向和裁剪窗口的边来定义。另外，通过添加一个近平面和一个远平面来限制观察体的范围，如图7.38所示。有限的斜平行投影观察体是一个斜平行管道。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

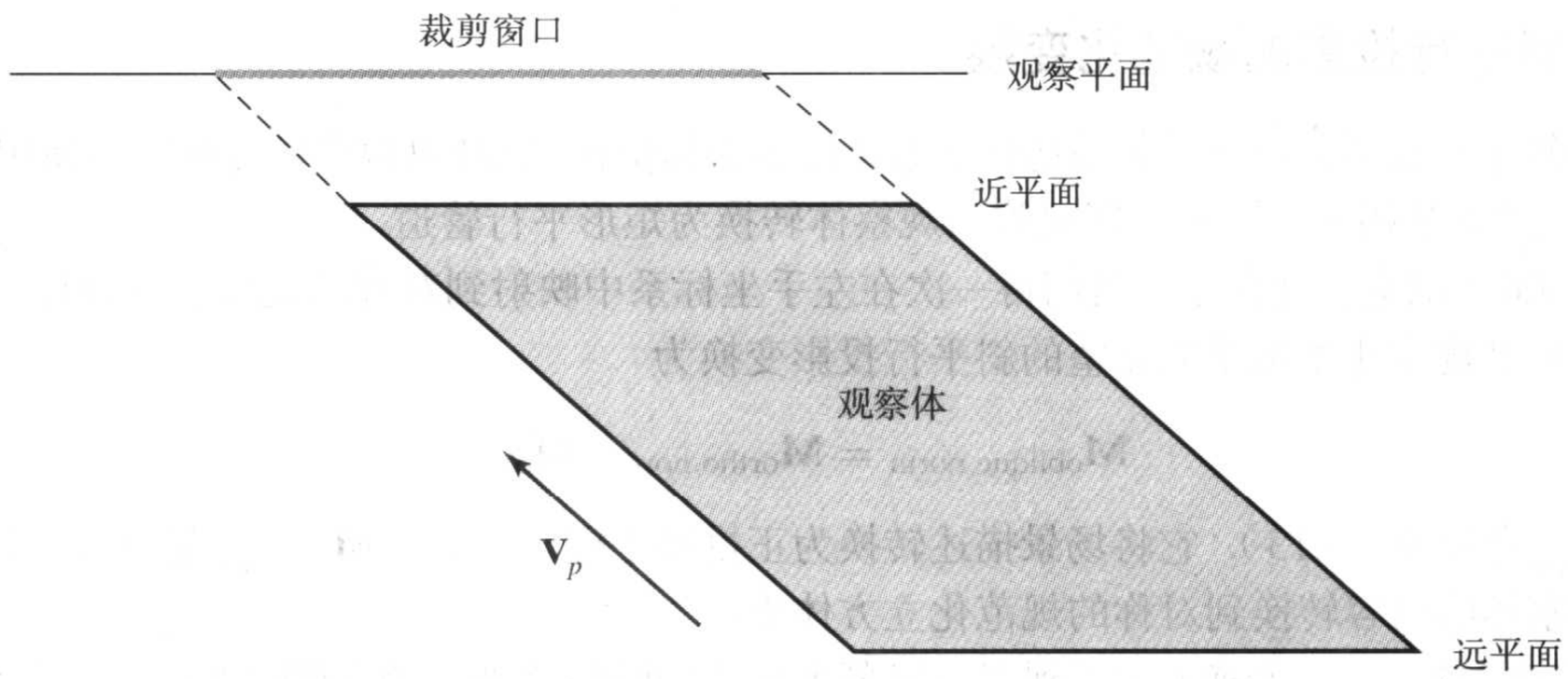


图 7.38　向量 $\mathbf{V}_p$ 方向的斜平行投影的有限观察体的顶视图

改变观察平面位置可能会影响斜平行投影，这依赖于怎样描述投影方向。在有些系统中，斜平行投影方向与连接参考点和裁剪窗口的中心的线条平行。因此，移动观察平面或裁剪窗口而不调整参考点会改变观察体的形状。

7.7.5　斜平行投影变换矩阵

使用方程（7.12）中的投影向量参数，我们可将斜平行投影变换矩阵的元素表达如下：

$$\mathbf{M}_{\text{oblique}} = \begin{bmatrix} 1 & 0 & -\dfrac{V_{px}}{V_{pz}} & z_{vp}\dfrac{V_{px}}{V_{pz}} \\ 0 & 1 & -\dfrac{V_{py}}{V_{pz}} & z_{vp}\dfrac{V_{py}}{V_{pz}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.13}$$

该矩阵将 x 和 y 坐标按其与在 z_{view} 轴上观察点 z_{vp} 的距离成比例地移动。空间点的 z 坐标是不改变的。如果 $V_{px} = V_{py} = 0$，则得到一个正投影且矩阵（7.13）简化为一个单位矩阵。

对于一般的斜平行投影，矩阵（7.13）表达 z 轴的错切变换。所有在斜观察体中的坐标位置按其与观察点的距离成比例地错切，其效果是将斜观察体错切成如图7.39所示的矩形的平行管道。这样，观察体中的位置被斜平行投影变换错切到正投影坐标。

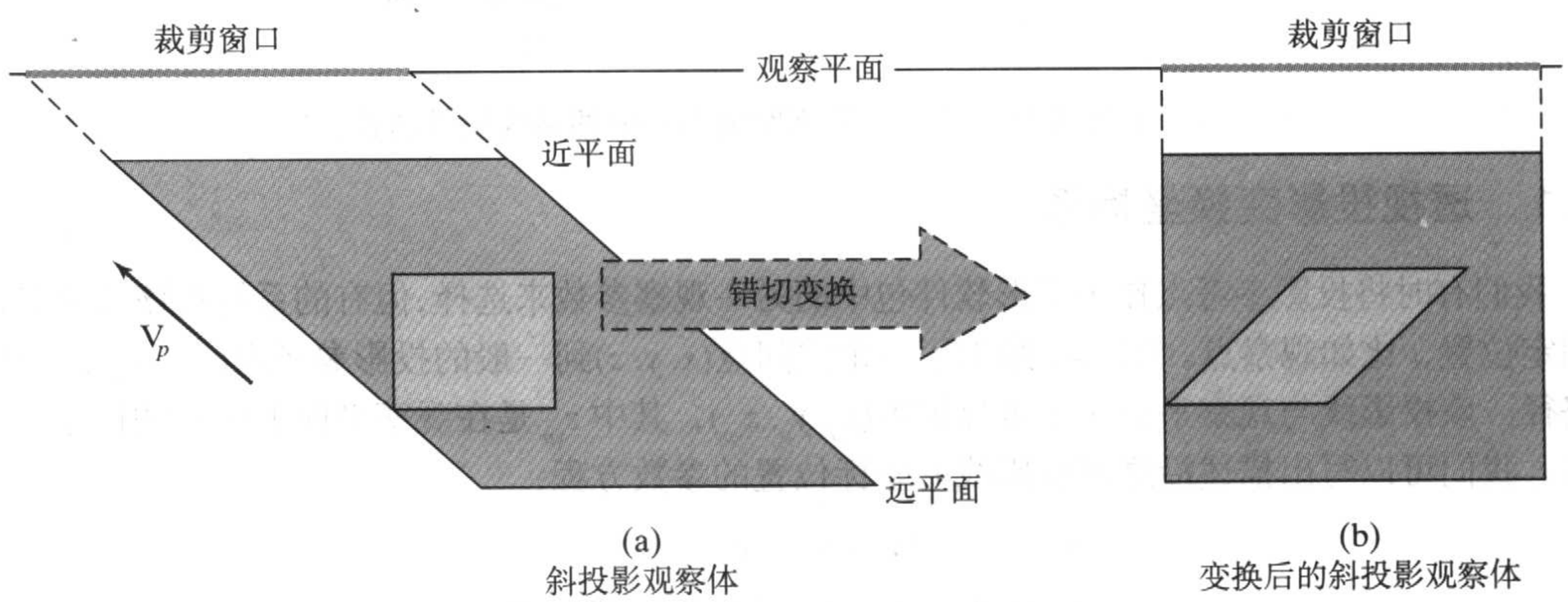


图 7.39　斜平行投影变换的顶视图。斜观察体转换成矩形平行管道，观察体中的对象（如绿色的方块）映射到正投影坐标系

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

7.7.6 斜平行投影的规范化变换

由于斜平行投影方程将对象描述转变为正投影坐标位置，因此可以在该变换之后使用规范化过程。使用与7.6节相同的过程，即可将斜观察体转换为矩形平行管道。

参照7.6节的规范化例子，我们再一次在左手坐标系中映射到对称的规范化立方体。这样，从观察坐标系到规范化坐标系的完整的斜平行投影变换为

$$\mathbf{M}_{\text{oblique,norm}} = \mathbf{M}_{\text{ortho,norm}} \cdot \mathbf{M}_{\text{oblique}} \tag{7.14}$$

变换 $\mathbf{M}_{\text{oblique}}$ 是矩阵（7.13），它将场景描述转换为正投影坐标。而变换 $\mathbf{M}_{\text{ortho,norm}}$ 是矩阵（7.7），它将正投影观察体的内容转换到对称的规范化立方体中。

为了完成整个观察变换（除了到视口屏幕坐标系的映射之外），将矩阵（7.14）合并到7.4节的变换 $\mathbf{M}_{WC,VC}$ 的左边。然后可以在规范化观察体中应用裁剪子程序，并接着进行确定可见对象、表面绘制和视口变换等工作。

7.8 透视投影

尽管场景的平行投影视图较易生成并且能保持对象的相对比例，但它不提供真实感表达。如果要模仿照片，必须考虑场景中会聚到照相机胶片平面的对象反射光线。可以通过将对象沿会聚到**投影参考点**（projection reference point）或**投影中心**（center of projection）的路径投影到观察平面来逼近这种几何-光学效果。对象按透视缩短效果显示，距离远的对象比相同大小的接近观察平面的对象的投影小（参见图7.40）。

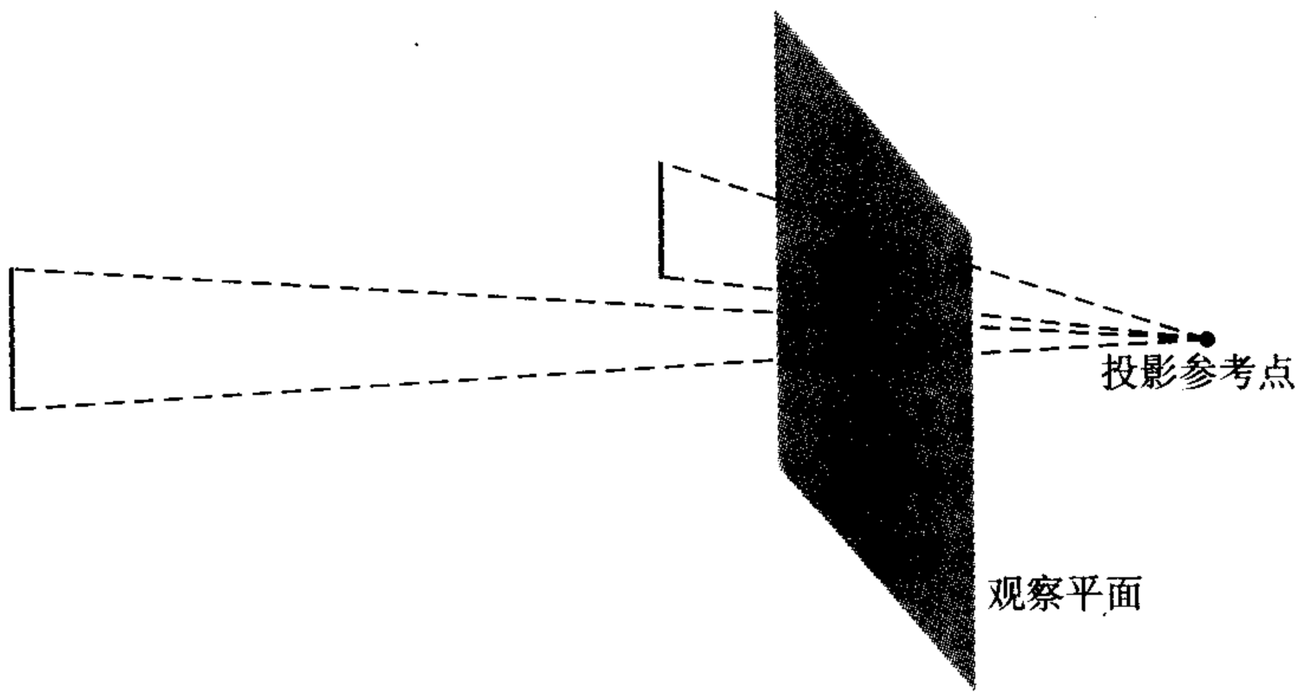


图7.40 长度相同、离观察平面距离不同的两条线段的透视投影

7.8.1 透视投影变换坐标系

我们有时将投影参考点作为图形软件包中的另一观察参数来选择，但有的系统将该会聚点设定在固定位置，比如观察点。图7.41给出了一个空间点(x, y, z)到一般的投影参考点$(x_{prp}, y_{prp}, z_{prp})$的投影路径。该投影线与观察平面交于坐标位置$(x_p, y_p, z_{vp})$，其中 z_{vp} 是在观察平面上选择的位于 z_{view} 轴的点。我们可以写出描述沿透视投影线的坐标位置的参数方程：

$$\begin{aligned} x' &= x - (x - x_{prp})u \\ y' &= y - (y - y_{prp})u \qquad 0 \leqslant u \leqslant 1 \\ z' &= z - (z - z_{prp})u \end{aligned} \tag{7.15}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

坐标位置(x', y', z')代表沿投影线的任意一点。当 $u = 0$ 时指位置 $\mathbf{P} = (x, y, z)$。$u = 1$ 时则位于线段的另一端且投影参考点坐标为$(x_{prp}, y_{prp}, z_{prp})$。在观察平面上有 $z' = z_{vp}$，可求解 z' 方程得到沿投影线的该位置的 u 参数：

$$u = \frac{z_{vp} - z}{z_{prp} - z} \tag{7.16}$$

将此 u 值代入 x' 和 y' 的方程，可得一般的投影变换公式：

$$\begin{aligned} x_p &= x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right) \\ y_p &= y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + y_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right) \end{aligned} \tag{7.17}$$

透视映射的计算比平行投影公式复杂，因为投影计算式（7.17）中的分母是空间位置 z 坐标的函数。因此，我们现在需要形成有一点区别的投影变换过程，以使该映射能与其他观察变换合并。但我们先看一下等式（7.17）的一些特点。

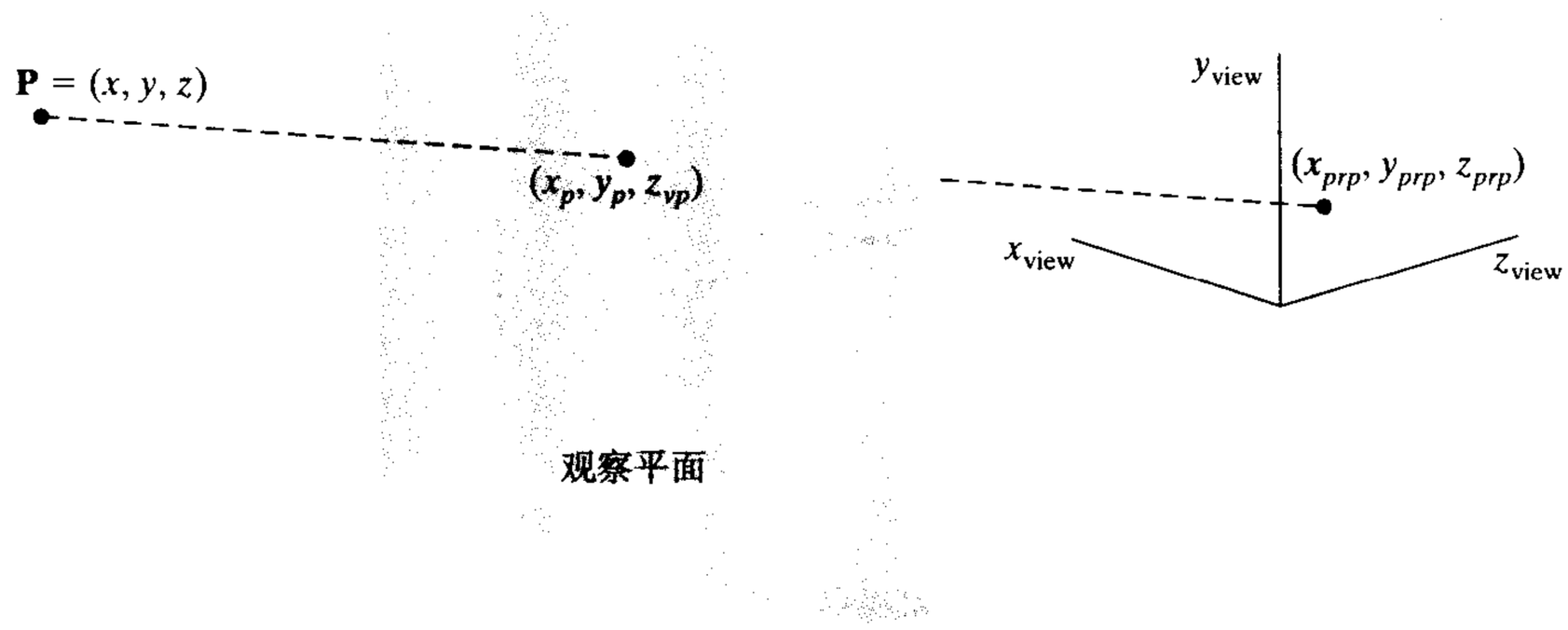


图 7.41　坐标为(x, y, z)的点到选定投影参考点的投影。在观察平面上的交点为(x_p, y_p, z_{vp})

7.8.2　透视投影公式：特殊情况

对于透视投影常常会在参数上加上一些限制。投影参考点或观察平面可能不完全任选，这依赖于特定的图形软件包而定。

为了简化透视投影的计算，投影参考点可能被限定在 z_{view} 轴上，然后

(1) $x_{prp} = y_{prp} = 0$:

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right), \qquad y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) \tag{7.18}$$

而有时将投影参考点固定于坐标系原点，即

(2) $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$:

$$x_p = x\left(\frac{z_{vp}}{z}\right), \qquad y_p = y\left(\frac{z_{vp}}{z}\right) \tag{7.19}$$

如果观察平面是 uv 平面且对投影参考点的位置不加限制，则有

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

(3) $z_{vp}=0$:

$$x_p = x\left(\frac{z_{prp}}{z_{prp}-z}\right) - x_{prp}\left(\frac{z}{z_{prp}-z}\right)$$
$$y_p = y\left(\frac{z_{prp}}{z_{prp}-z}\right) - y_{prp}\left(\frac{z}{z_{prp}-z}\right) \tag{7.20}$$

如果把 uv 平面作为观察平面且投影参考点在 z_{view} 轴上，则透视投影公式为

(4) $x_{prp}=y_{prp}=z_{vp}=0$:

$$x_p = x\left(\frac{z_{prp}}{z_{prp}-z}\right), \qquad y_p = y\left(\frac{z_{prp}}{z_{prp}-z}\right) \tag{7.21}$$

当然，投影参考点不能位于投影平面上。因为那样会导致所有的场景投影到一点。观察平面通常安排在投影参考点和场景之间，但一般来说，观察平面可以安排在除投影点外的任意位置。如果投影参考点在观察平面和场景之间，则对象在观察平面上成倒置状态（参见图 7.42）。如果场景位于观察平面与投影点之间，则对象从观察位置向观察平面投影时简单地放大。

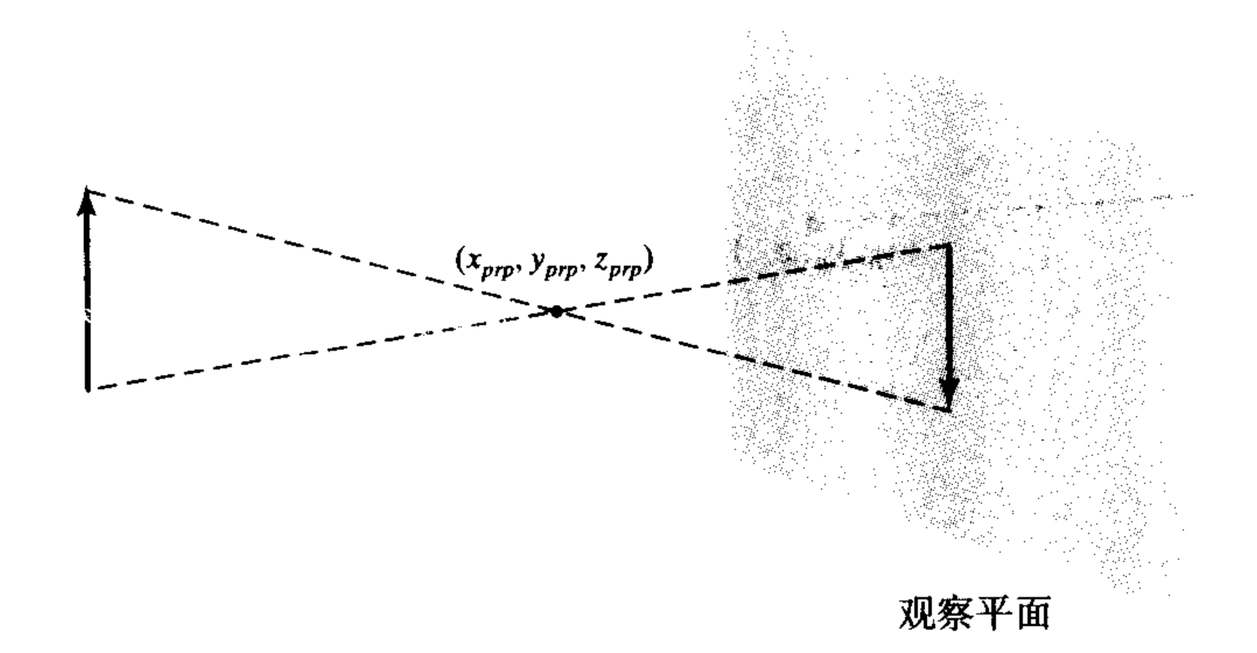


图 7.42　投影参考点位于对象和观察平面之间时对象的透视投影成倒置状态

投影效果还依赖于从投影参考点到观察平面的距离，如图 7.43 所示。如果投影参考点接近观察平面，投影效果将被强化：即较近的对象的显示比相同大小的较远对象大得多。同样，当投影参考点离观察平面较远时，近和远的对象的大小差别会减少。当投影参考点离观察平面很远时，透视投影向平行投影靠拢。

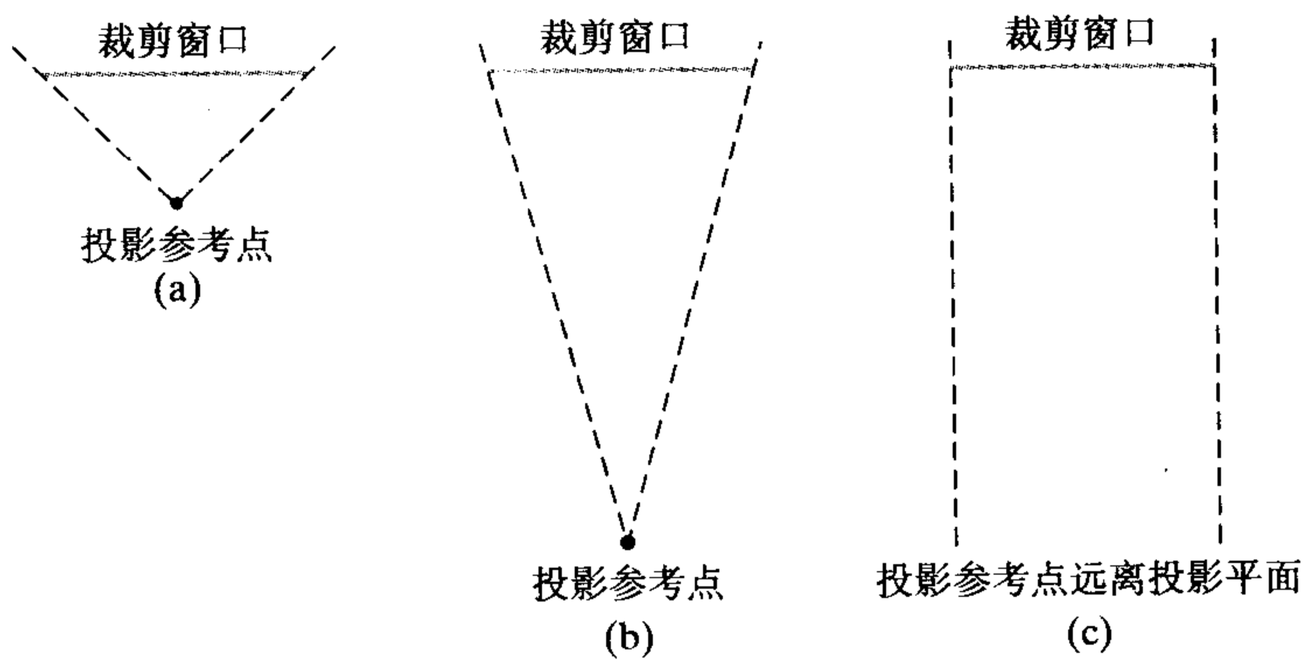


图 7.43　通过移动投影参考点使之远离观察平面来改变透视投影效果

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

7.8.3　透视投影的灭点

当一个场景使用透视映射投影到观察平面上时，平行于观察平面的线条投影后仍然平行。但是任何与观察平面不平行的平行线组投影后成为一组会聚线条，如图7.4所示。一组投影平行线会聚的点称为**灭点**（vanishing point）。每一组平行线有自己单独的灭点。

对象上一组平行于一个主轴的平行线，其灭点称为**主灭点**（principal vanishing point）。通过投影平面的方向可控制主灭点的数量（一个、两个或三个），相应地透视投影分为一点、两点或三点投影。主灭点的数量和与观察平面相交的主轴数量相同。图7.44给出了一个立方体的一点和两点透视投影的外貌。在投影图(b)中，观察平面与对象的 xy 平面平行，因而只有 z 轴与之相交。该方向导致一点透视投影，其主灭点在 z 轴上。而(c)中的视图，投影平面与 x 和 z 轴相交，但与 y 轴不相交。导致的两点透视投影包含 x 轴和 z 轴两个灭点。三点透视投影比两点透视投影在真实感上并未增加许多，因此三点投影在建筑和工程制图中使用得并不很多。

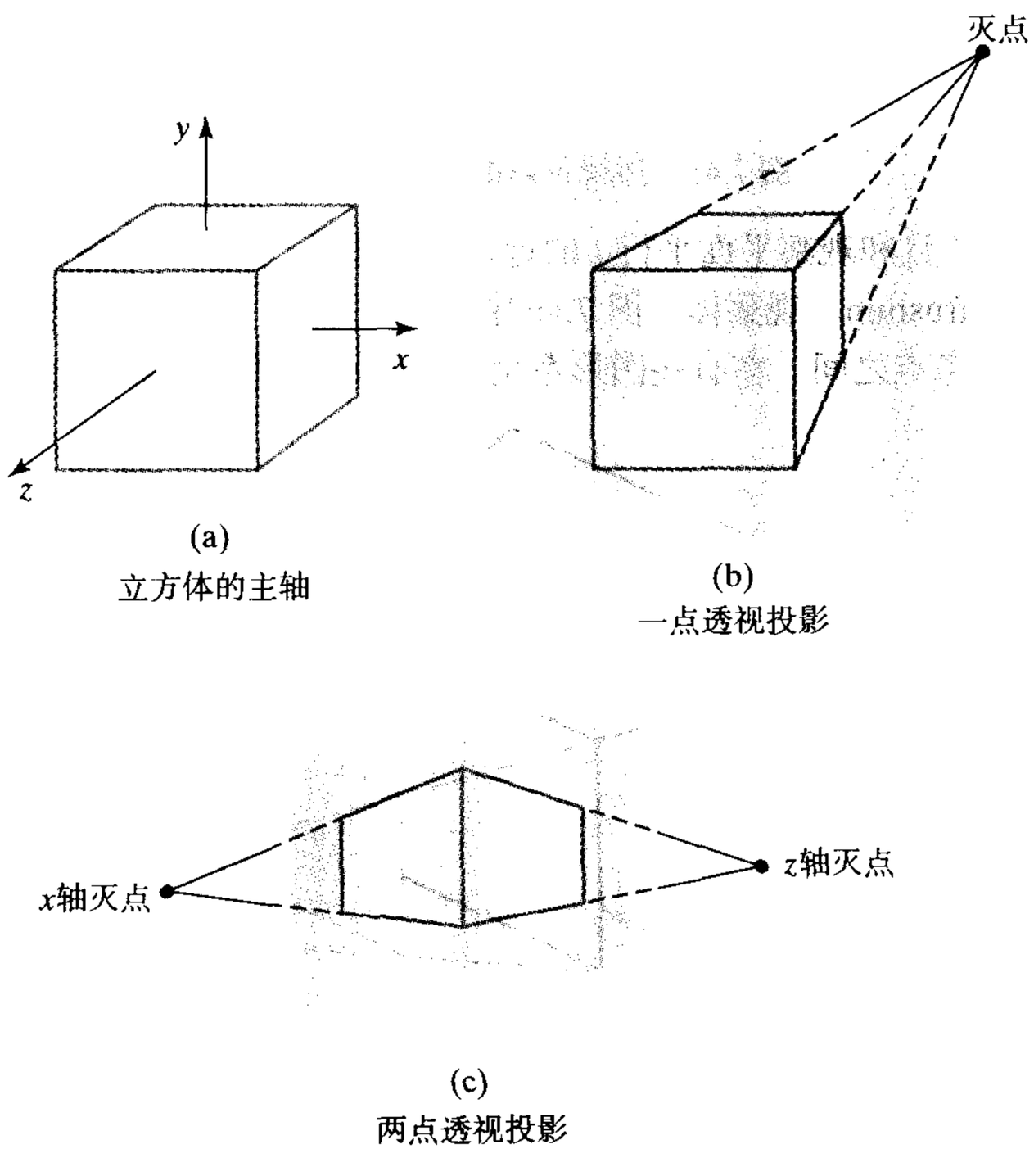


图 7.44　一个立方体透视投影的主灭点。当(a)立方体投影到视平面上时仅与 z 轴相交则生成一个灭点(b)。当该立方体投影到观察平面上时与 z 和 x 轴都相交，则生成两个灭点(c)

7.8.4　透视投影观察体

通过在观察平面上指定一个矩形裁剪窗口可得到一个观察体。但现在的观察体边界面不再平行，因为投影线不是平行的。观察体的底面、顶面和侧面是通过窗口边线相交于投影参考点的平面。这形成一个顶点在投影中心（参见图7.45）的无限矩形棱锥。在该棱锥体之外的所有对象都被裁剪子程序消除。透视投影观察体常称为**视觉棱锥体**（pyramid of vision），因为它与眼睛或照相机的视

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

觉圆锥体（cone of vision）相近。显示的场景视图仅仅包括那些位于棱锥体之内的对象，就像我们不能看到视觉圆锥体外围的对象一样。

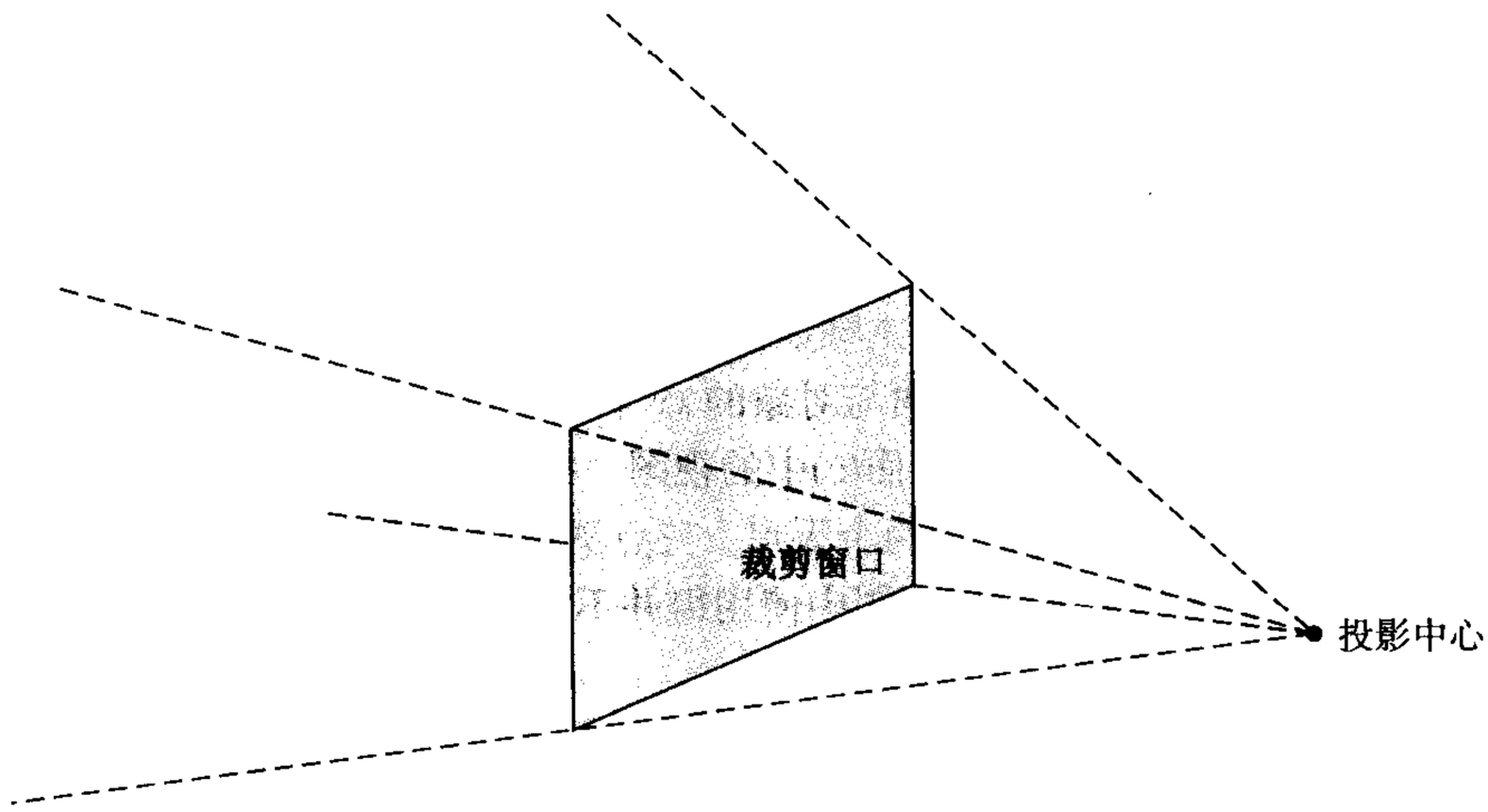


图 7.45　透视投影的棱锥形观察体

添加垂直于 z_{view} 轴（且和观察平面平行）的近、远平面后，切掉了无限、透视投影观察体的一部分，形成一个**棱台**（frustum）观察体。图 7.46 中一个有限的、透视投影观察体形状，其观察平面位于近平面和投影参考点之间。在有些图形系统中，近和远平面是要求的，有的则任选。

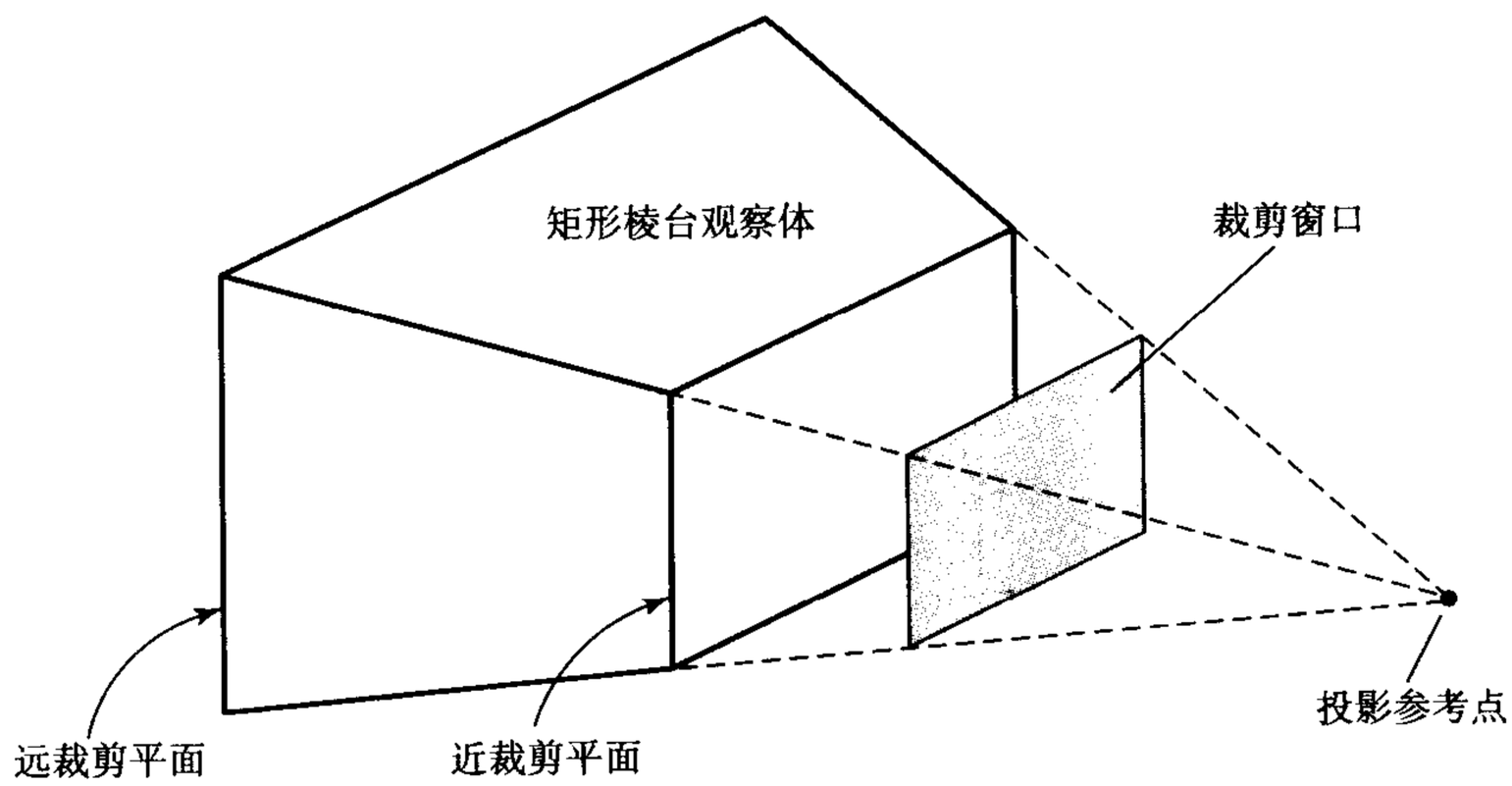


图 7.46　观察平面在近平面之前的透视投影观察体

通常，近和远平面位于投影参考点的同侧，远平面比近平面沿观察方向离投影点更远。在平行投影中，我们可以简单地使用近和远平面以包括观察的场景。但在透视投影中，近裁剪平面可用来除去那些在裁剪窗口内的接近观察平面、其投影形状可能无法辨认的大对象。同样，远裁剪平面可用来切掉那些远离投影参考点、可能投影成观察平面上小点的对象。有些系统相对于近和远平面来限制观察平面的定位，而其他的系统则允许它被定位于除投影参考点以外的任何位置。如果观察平面在投影参考点之后，对象被颠倒，如图 7.42 所示。

7.8.5　透视投影变换矩阵

与平行投影不同，我们不能直接使用等式（7.17）中的 x 和 y 坐标系数来形成透视投影矩阵的元素，因为系数的分母是 z 的函数。但我们可以使用三维齐次坐标表示来给出透视投影的公式如下：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$x_p = \frac{x_h}{h}, \qquad y_p = \frac{y_h}{h} \tag{7.22}$$

这里齐次参数的值为

$$h = z_{prp} - z \tag{7.23}$$

等式（7.22）中的分子与等式（7.17）中的相同：

$$\begin{aligned} x_h &= x(z_{prp} - z_{vp}) + x_{prp}(z_{vp} - z) \\ y_h &= y(z_{prp} - z_{vp}) + y_{prp}(z_{vp} - z) \end{aligned} \tag{7.24}$$

因此，可建立一个变换矩阵将一个空间位置转为齐次坐标位置，使得矩阵仅包含透视参数而不包含坐标值。接着，观察坐标系的透视投影变换分两步实现。先用下列透视变换矩阵计算齐次坐标：

$$\mathbf{P}_h = \mathbf{M}_{\text{pers}} \cdot \mathbf{P} \tag{7.25}$$

这里，$\mathbf{P}_h$ 是齐次点(x_h, y_h, z_h, h)的列矩阵表示而 $\mathbf{P}$ 是坐标位置$(x, y, z, 1)$的列矩阵表示。（实际上，透视矩阵要与观察矩阵合并，然后将组合矩阵应用于场景的世界坐标描述以生成齐次坐标。）第二，在经过规范化变换和裁剪等其他处理后，用参数 h 去除齐次坐标可得到真实的变换坐标位置。

为了获得等式（7.24）中的齐次坐标 x_h 和 y_h 而建立矩阵元素是比较直接的。但我们也必须为保存深度信息而构造矩阵。否则，除以齐次参数 h 后 z 坐标会出现扭曲。我们可通过为 z 变换设定矩阵元素从而对透视投影 z_p 的坐标进行规范化。有多种方法可用来选择矩阵元素，从而为空间点(x, y, z)生成齐次坐标（7.24）和规范化的 z_p 值。下面的矩阵给出了一种可能形成透视投影矩阵的方法。

$$\mathbf{M}_{\text{pers}} = \begin{bmatrix} z_{prp} - z_{vp} & 0 & -x_{prp} & x_{prp}z_{prp} \\ 0 & z_{prp} - z_{vp} & -y_{prp} & y_{prp}z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix} \tag{7.26}$$

参数 s_z 和 t_z 是在对 z 坐标投影值规范化中的比例和平移因子。s_z 和 t_z 的特定值依赖于选择的规范化范围。

矩阵（7.26）将场景描述转换成齐次平行投影坐标。然而，棱台观察体可以有任意的方向，因此，这些变换后的坐标可以和斜平行投影对应。这发生在棱台观察体不对称的时候。如果透视投影棱台观察体是对称的，导致的平行投影坐标与正交投影对应。下面考虑这两种可能性。

7.8.6 对称的透视投影锥体

从投影参考点到裁剪窗口中心并穿过观察体的线条是透视投影棱台的中心线。如果该中心线与投影平面垂直，则有一个**对称棱台**（相对于该中心线），如图 7.47 所示。

由于棱台中心线与观察平面相交于坐标位置$(x_{prp}, y_{prp}, z_{vp})$，我们可以用窗口尺寸表达裁剪窗口的对角位置，如下所示：

$$xw_{\min} = x_{prp} - \frac{\text{宽度}}{2}, \qquad xw_{\max} = x_{prp} + \frac{\text{宽度}}{2}$$

$$yw_{\min} = y_{prp} - \frac{\text{高度}}{2}, \qquad yw_{\max} = y_{prp} + \frac{\text{高度}}{2}$$

因此，可以用裁剪窗口的宽和高代替窗口的坐标来指定一个场景的对称透视投影视图。因为它相对于投影参考点的 x 和 y 坐标对称，因而惟一地建立了裁剪窗口的位置。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

另外一种指定对称透视投影的方法是使用逼近照相机镜头特性的参数。相片在胶片平面上用场景的对称透视投影来生成。从场景对象出发的光线集中到照相机“视锥”中的胶片平面上。该视锥可看做一个**视场角**（field-of-view angle），它是照相机镜头的尺寸度量。例如，一个大的视场角对应于一个广角镜头。在计算机图形学中，视锥用对称棱台来近似，我们可使用视场角指定该棱台的角度尺寸。一般而言，视场角是棱台上裁剪平面和下裁剪平面之间的角度，如图 7.48 所示。

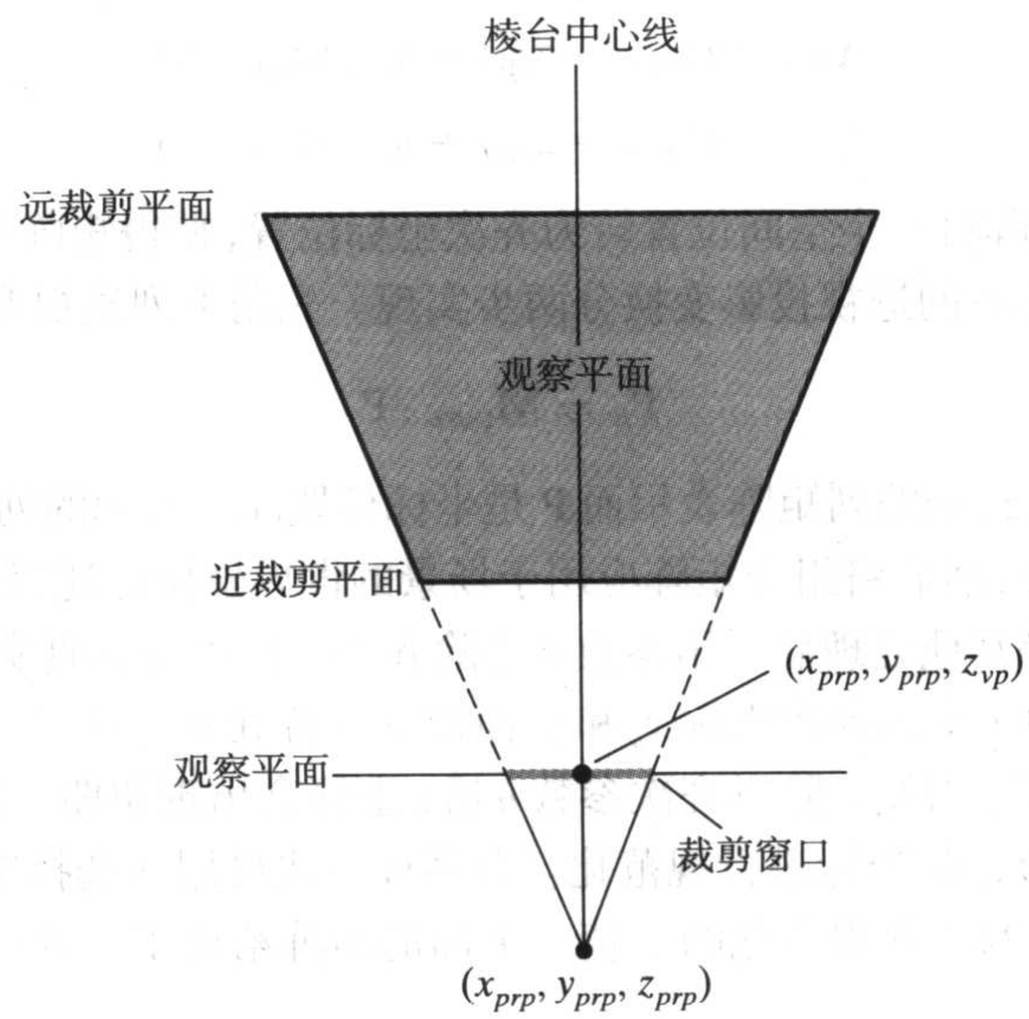


图 7.47　一个对称的透视投影棱台观察体，其观察平面位于投影参考点与近裁剪平面之间。从上面、下面或两侧看时，该棱台相对于其中心线对称

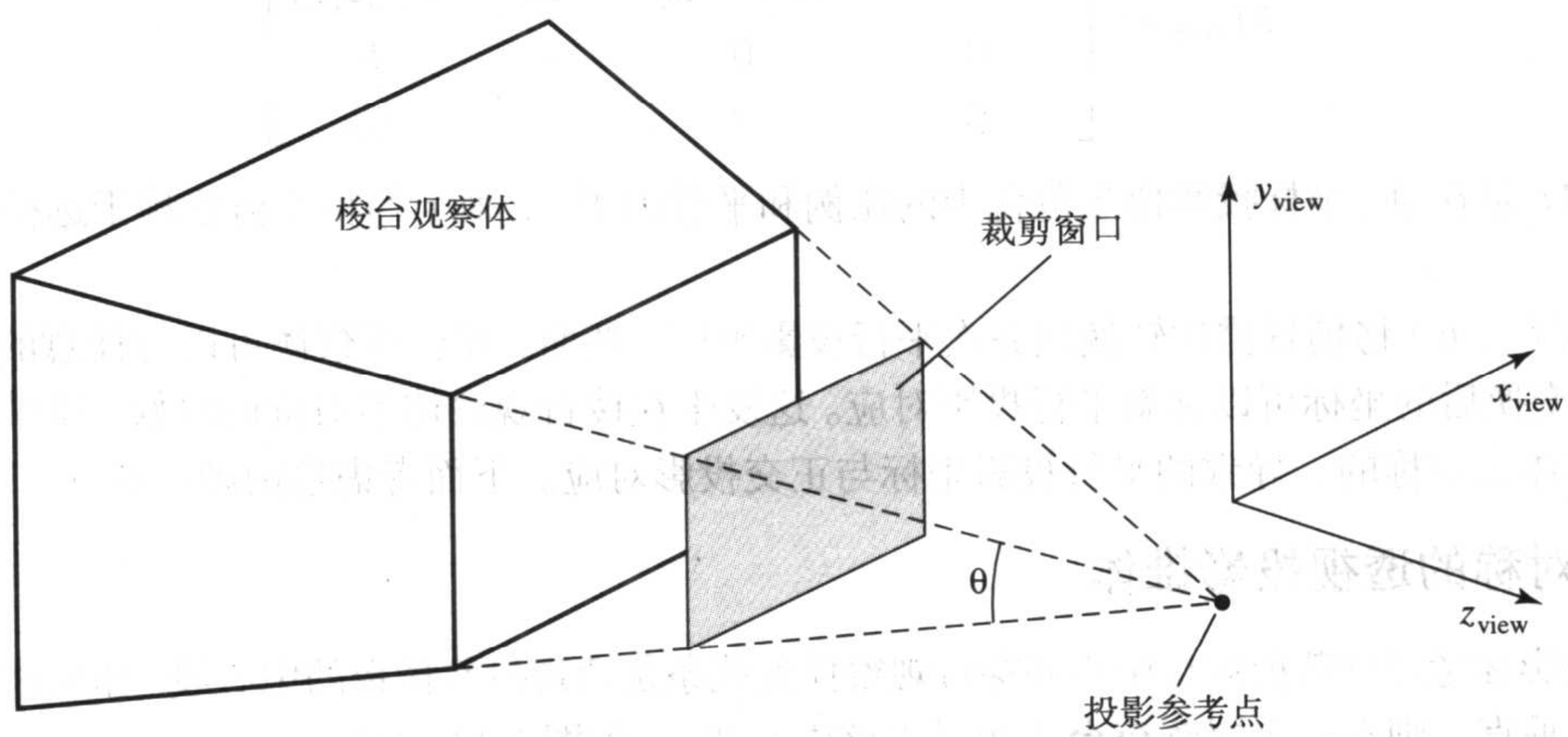


图 7.48　一个对称透视投影观察体的视场角，其裁剪窗口位于近裁剪平面和投影参考点之间

对于给定的投影参考点和观察平面位置，视场角确定裁剪窗口（参见图 7.49）的高，而不是宽。我们需要另一个参数来完整定义裁剪窗口的尺寸，而这第二个尺寸可以是裁剪窗口的窗口宽或纵横比（宽 / 高）。从图 7.49 的右边角度可以看出

$$\tan\left(\frac{\theta}{2}\right) = \frac{\text{高度}/2}{z_{prp} - z_{vp}} \tag{7.27}$$

因此裁剪窗口高度可如下计算：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$高度 = 2(z_{prp} - z_{vp})\tan\left(\frac{\theta}{2}\right) \tag{7.28}$$

因此，矩阵（7.26）中值 $z_{prp} - z_{vp}$ 的对角线元素可以用下列两表达式之一来代替。

$$\begin{aligned} z_{prp} - z_{vp} &= \frac{高度}{2}\cot\left(\frac{\theta}{2}\right) \\ &= \frac{宽度 \cdot \cot(\theta/2)}{2 \cdot 纵横比} \end{aligned} \tag{7.29}$$

在有些图形库中，观察平面和投影参考点使用固定点，因而对称透视投影完全由视场角、裁剪窗口的纵横比及从观察位置到近和远裁剪平面的距离来确定。视口通常使用相同的纵横比。

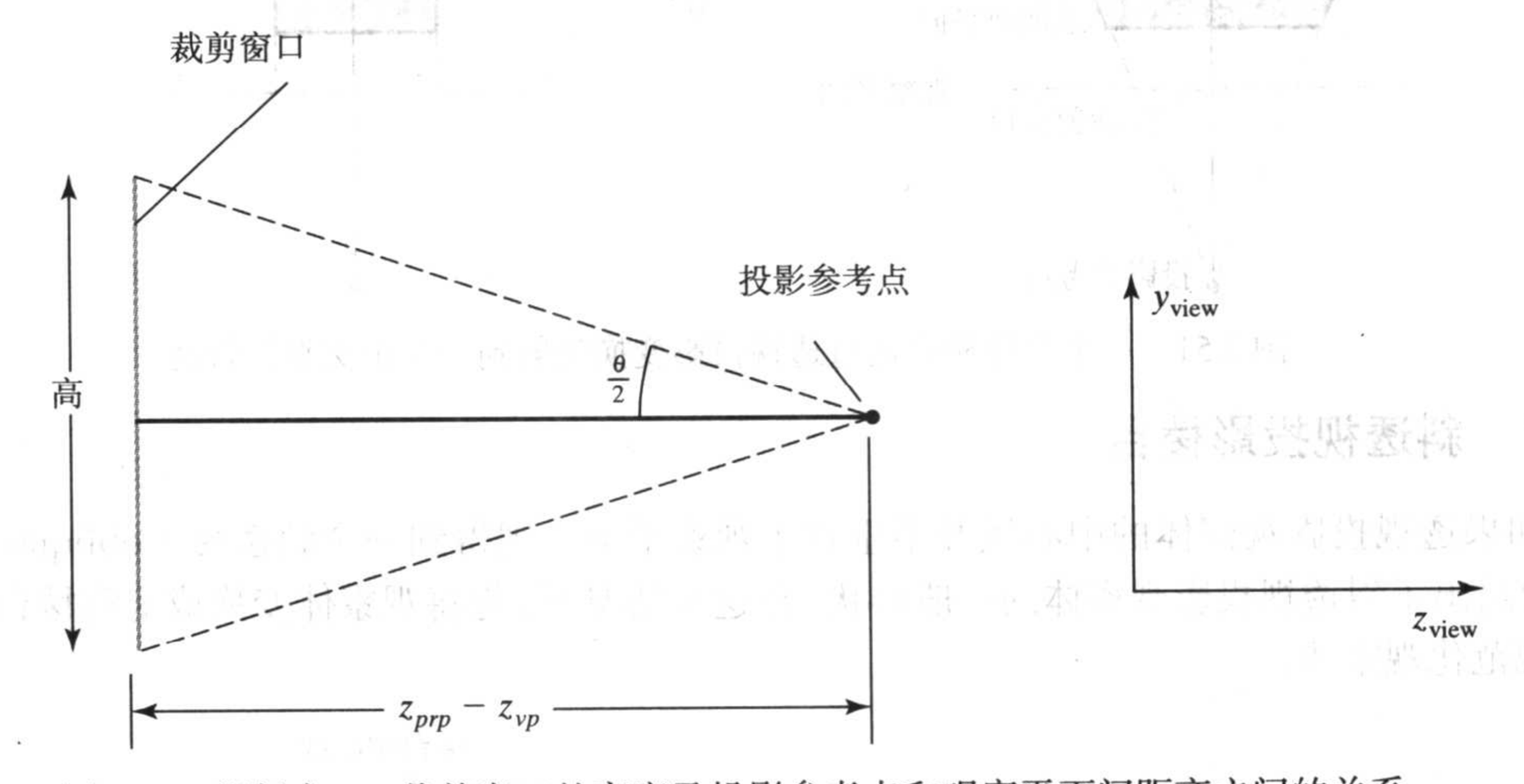


图 7.49　视场角 θ、裁剪窗口的高度及投影参考点和观察平面间距离之间的关系

如果在特定应用中视场角减小，则透视投影的透视缩短效应也减小。这相当于将投影参考点移离观察平面。减小视场角也减小了裁剪平面的高度，这给出了聚焦到场景的一个小区域的方法。类似地，一个大的视场角将导致大的裁剪窗口高度（扩大视野），它增加了透视效果，这也是当我们的投影参考点靠近观察平面时得到的效果。图7.50给出了固定裁剪窗口宽度时不同视场角的效果。

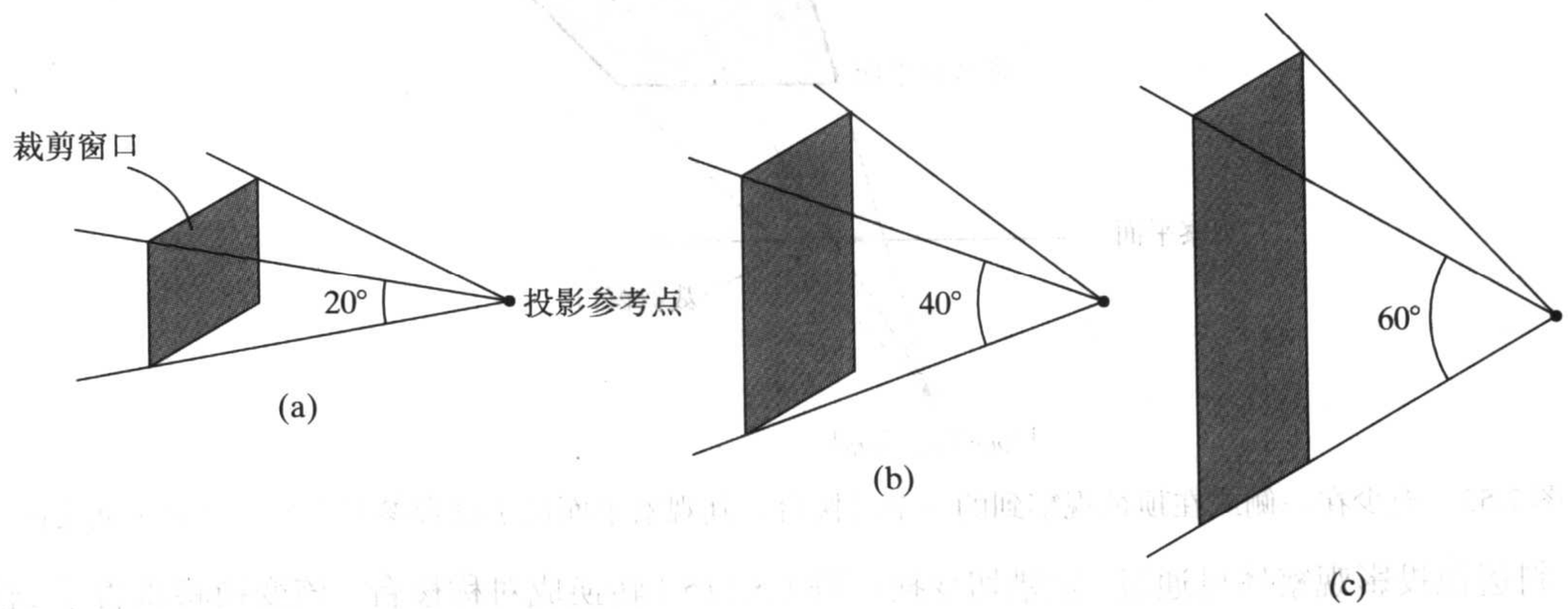


图 7.50　增加视场角就增加了裁剪窗口的高度及透视投影效果

当透视投影观察体是一个对称棱台时，透视变换将棱台内部的位置映射到矩形平行管道中的正交投影坐标。由于棱台中心线已经和观察平面垂直（参见图 7.51），故平行管道的中心线就是棱台

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

的中心线。这是棱台中所有投影线上的位置映射到观察平面上同一点(x_p, y_p)的结果。因此，每一投影线由透视变换转换成正交于观察平面的线条，因而平面于棱台中心线。使用转换到正交投影观察体后的对称棱台，可进入下一步的规范化变换。

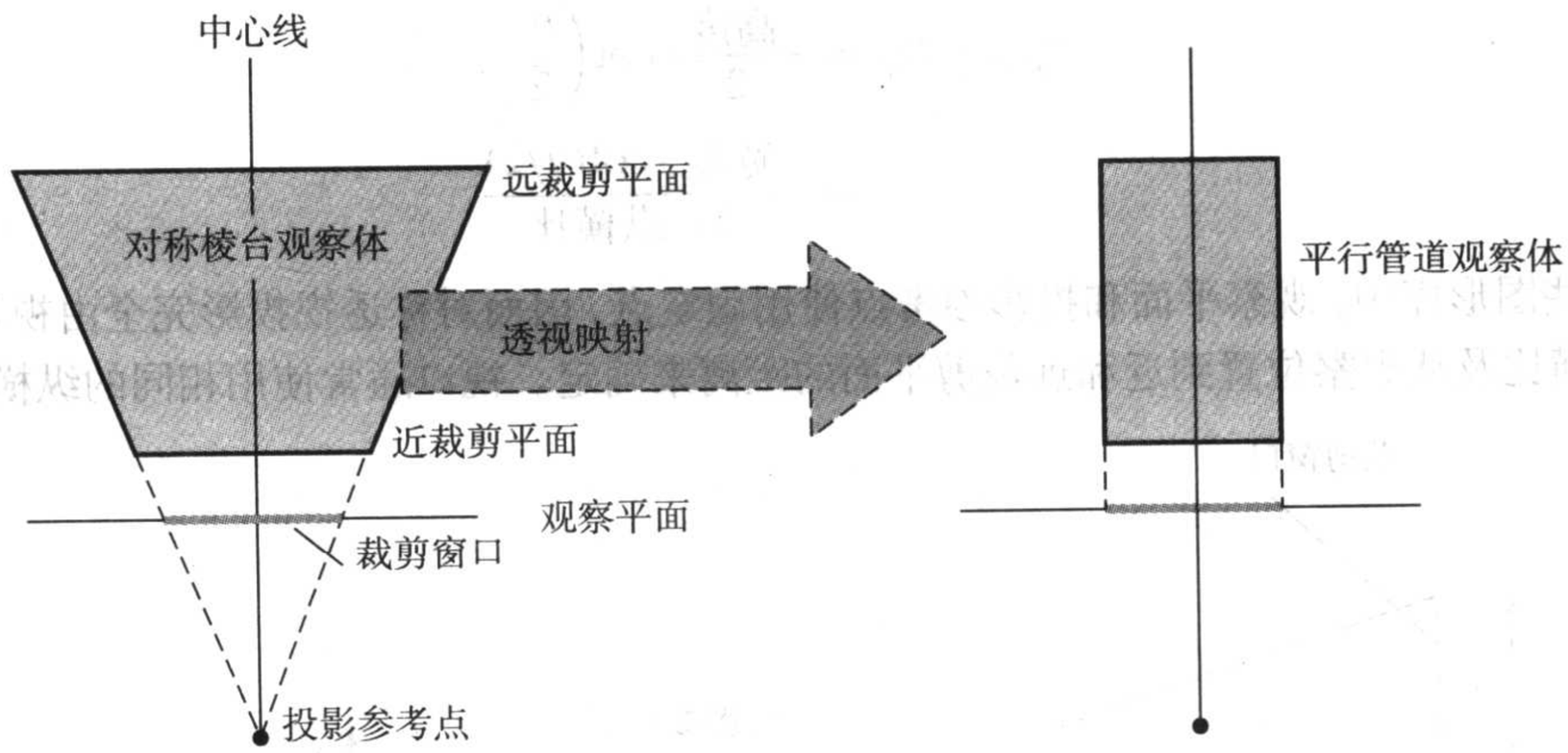


图 7.51　一个对称棱台通过透视投影变换映射到一个正交平行管道

7.8.7　斜透视投影棱台

如果透视投影观察体的中心线并不垂直于观察平面，则得到一个**斜棱台**（oblique frustum）。图7.52给出了斜透视投影观察体的一般形状。在这种情况下，先将观察体变换成对称棱台然后再变换成规范化观察体。

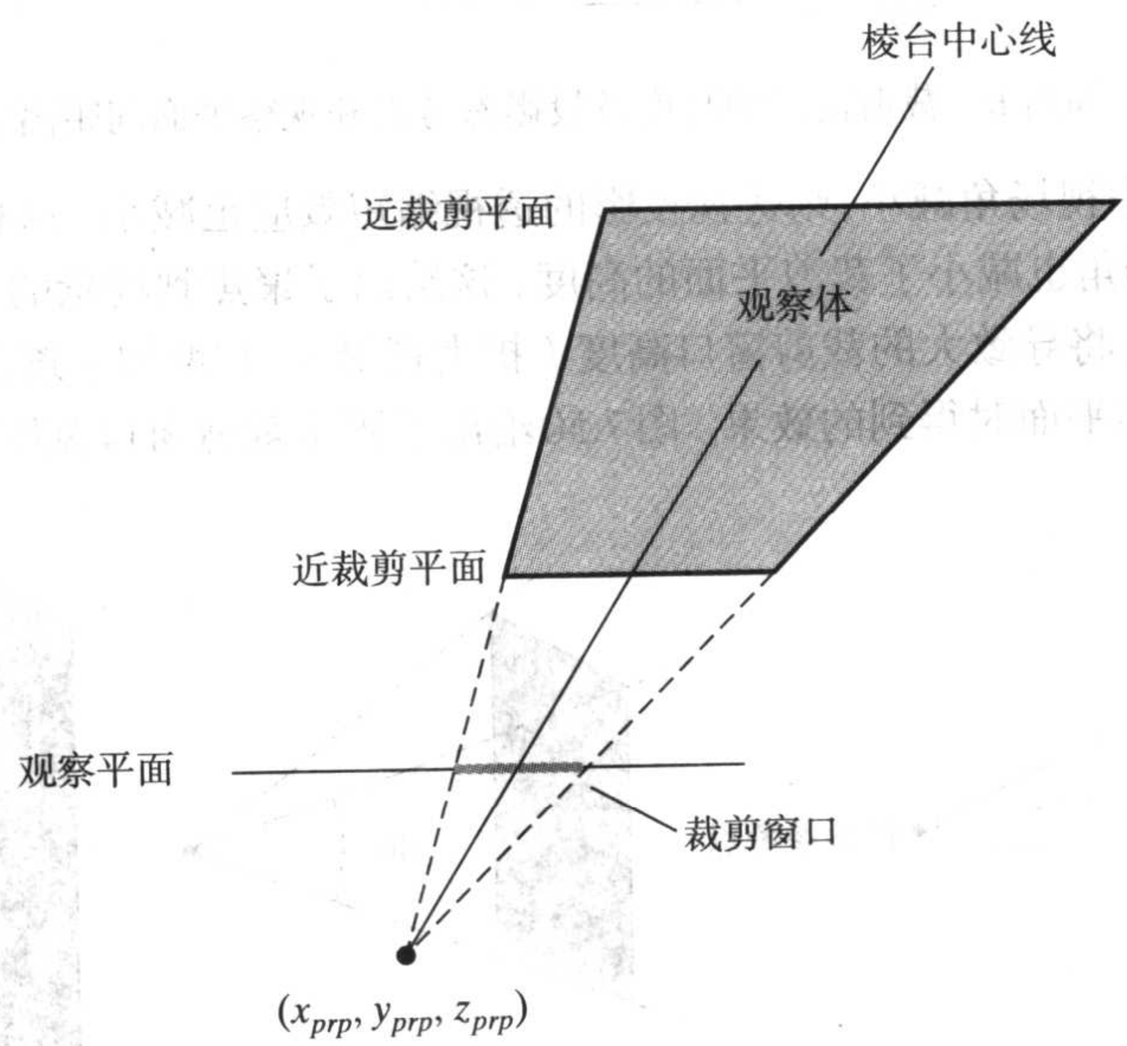


图 7.52　至少在一侧或在顶视观察到的一个斜棱台，其观察平面位于投影参考点和近裁剪平面之间

斜透视投影观察体可通过z轴错切变换矩阵（5.115）转换成对称棱台。该变换将垂直于z轴的所有平面上的位置按正比于平面到指定的z参考位置的距离值而移动。此时，参考位置是z_{prp}，即投影参考点的z坐标。根据把裁剪窗口中心移到观察平面的位置(x_{prp}, y_{prp})上来确定移动量。由于锥体中心线穿过裁剪窗口中心，上述移位使中心线与观察平面垂直，如图 7.47 所示。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如果选择观察坐标系原点作为投影参考点，则大大减少错切变换以及透视和规范化变换的计算量。不失普遍性，我们可以平移场景中所有的坐标位置，使得所选投影参考点与坐标系原点重合。或在最初就将观察坐标系原点设定在所要的场景投影参考点。事实上，有些图形库确实把投影参考点固定在坐标系原点。

取投影参考点为$(x_{prp}, y_{prp}, z_{prp})=(0, 0, 0)$，可得所需的错切变换矩阵的元素，如下所示：

$$\mathbf{M}_{z\,\mathrm{shear}} = \begin{bmatrix} 1 & 0 & \mathrm{sh}_{zx} & 0 \\ 0 & 1 & \mathrm{sh}_{zy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.30}$$

如果将观察平面放在近裁剪平面处，则透视投影矩阵可进一步简化。由于我们要将裁剪窗口中心移到观察平面上坐标位置(0, 0)处，需选择错切参数值满足

$$\begin{bmatrix} 0 \\ 0 \\ z_{\mathrm{near}} \\ 1 \end{bmatrix} = \mathbf{M}_{z\,\mathrm{shear}} \cdot \begin{bmatrix} \dfrac{xw_{\min} + xw_{\max}}{2} \\ \dfrac{yw_{\min} + yw_{\max}}{2} \\ z_{\mathrm{near}} \\ 1 \end{bmatrix} \tag{7.31}$$

因此，该错切变换的参数为

$$\begin{aligned} \mathrm{sh}_{zx} &= -\frac{xw_{\min} + xw_{\max}}{2\,z_{\mathrm{near}}} \\ \mathrm{sh}_{zy} &= -\frac{yw_{\min} + yw_{\max}}{2\,z_{\mathrm{near}}} \end{aligned} \tag{7.32}$$

类似地，当投影参考点位于观察坐标原点且近裁剪平面与观察平面重合时，透视投影矩阵（7.26）简化为

$$\mathbf{M}_{\mathrm{pers}} = \begin{bmatrix} -z_{\mathrm{near}} & 0 & 0 & 0 \\ 0 & -z_{\mathrm{near}} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{7.33}$$

z 坐标缩放和平移参数的表达式由规范化要求来确定。

将简化后的透视投影矩阵（7.33）和错切矩阵（7.30）合并，可得下面的将场景坐标位置转换成齐次正交坐标的斜透视投影矩阵。该变换中的投影参考点是观察坐标原点，而近裁剪平面是观察平面。

$$\begin{aligned} \mathbf{M}_{\mathrm{obliquepers}} &= \mathbf{M}_{\mathrm{pers}} \cdot \mathbf{M}_{z\,\mathrm{shear}} \\ &= \begin{bmatrix} -z_{\mathrm{near}} & 0 & \dfrac{xw_{\min} + xw_{\max}}{2} & 0 \\ 0 & -z_{\mathrm{near}} & \dfrac{yw_{\min} + yw_{\max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix} \end{aligned} \tag{7.34}$$

尽管失去了对投影参考点和观察平面定位的选择机会，但这一矩阵在不损失更多灵活性的前提下提供了生成场景的透视投影视图的高效方法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如果将裁剪窗口的坐标选成$xw_{max}=-xw_{min}$和$yw_{max}=-yw_{min}$，则棱台观察体是对称的且矩阵（7.34）简化为矩阵（7.33）。这是因为投影参考点现在位于观察坐标系原点。在$z_{prp}=0$和$z_{vp}=z_{near}$时，也可以用等式（7.29）以视场角和裁剪窗口尺寸来表达该矩阵的前两个对角线元素。

7.8.8 规范化透视投影变换坐标

矩阵（7.34）将观察坐标系中的对角位置变换到透视投影齐次坐标。使用齐次参数h除齐次坐标，可得实际的正交投影坐标。因此，该透视投影将棱台观察体中所有点变换成矩形平行管道观察体中的位置。透视变换过程的最后一步是将该平行管道映射到规范化观察体（normalized view volume）中。

我们遵循用于平行投影的规范化过程。从棱台观察体变换而来的矩形平行管道映射到对称的左手参考系的规范化立方体（参见图7.53）。z坐标的规范化参数已经包括在透视投影矩阵（7.34）中，但还需要确定在变换到对称的规范化立方体时这些参数的值。x和y坐标的规范化变换参数的值同样要确定。因为现在矩形平行管道的中心线是z_{view}，在x和y方向不再需要平移：我们只需相对于坐标系原点的x和y缩放参数。完成xy规范化的缩放矩阵是

$$\mathbf{M}_{xy\,\mathrm{scale}}=\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.35}$$

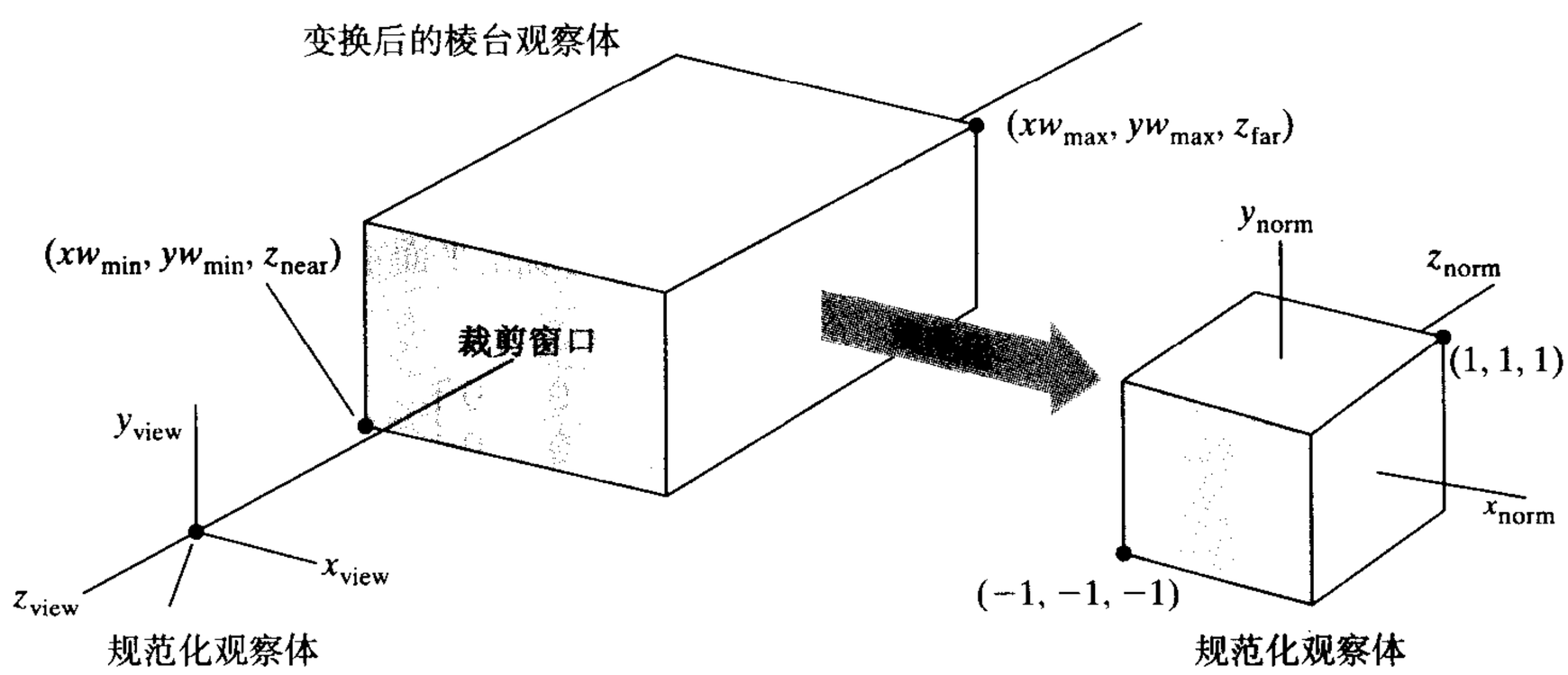


图 7.53 近裁剪平面用做观察平面且投影参考点位于观察坐标系原点时，从透视投影观察体（矩形平行管道）到左手参考系对称规范化立方体的规范化变换

将xy缩放矩阵和矩阵（7.34）合并，生成下列透视投影变换的规范化矩阵：

$$\begin{aligned}\mathbf{M}_{\mathrm{normpers}}&=\mathbf{M}_{xy\,\mathrm{scale}}\cdot\mathbf{M}_{\mathrm{obliquepers}}\\&=\begin{bmatrix} -z_{near}s_x & 0 & s_x\dfrac{xw_{min}+xw_{max}}{2} & 0 \\ 0 & -z_{near}s_y & s_y\dfrac{yw_{min}+yw_{max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}\end{aligned} \tag{7.36}$$

从这一变换，可得齐次坐标：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M}_{\text{normpers}} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{7.37}$$

而投影坐标是

$$\begin{aligned} x_p &= \frac{x_h}{h} = \frac{-z_{\text{near}} s_x x + s_x (x w_{\min} + x w_{\max})/2}{-z} \\ y_p &= \frac{y_h}{h} = \frac{-z_{\text{near}} s_y y + s_y (y w_{\min} + y w_{\max})/2}{-z} \\ z_p &= \frac{z_h}{h} = \frac{s_z z + t_z}{-z} \end{aligned} \tag{7.38}$$

为了将这一透视变换规范化，希望在输入坐标是$(x, y, z) = (xw_{\min}, yw_{\min}, z_{\text{near}})$时投影坐标是$(x_p, y_p, z_p) = (-1, -1, -1)$，在输入坐标是$(x, y, z) = (xw_{\max}, yw_{\max}, z_{\text{far}})$时投影坐标是$(x_p, y_p, z_p) = (1, 1, 1)$。因此，在对方程（7.38）求解规范化参数时使用这些条件，可得

$$\begin{aligned} s_x &= \frac{2}{x w_{\max} - x w_{\min}}, & s_y &= \frac{2}{y w_{\max} - y w_{\min}} \\ s_z &= \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}}, & t_z &= \frac{2 z_{\text{near}} z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \end{aligned} \tag{7.39}$$

而一般的透视投影规范化变换的元素是

$$\mathbf{M}_{\text{normpers}} = \begin{bmatrix} \frac{-2z_{\text{near}}}{x w_{\max} - x w_{\min}} & 0 & \frac{x w_{\max} + x w_{\min}}{x w_{\max} - x w_{\min}} & 0 \\ 0 & \frac{-2z_{\text{near}}}{y w_{\max} - y w_{\min}} & \frac{y w_{\max} + y w_{\min}}{y w_{\max} - y w_{\min}} & 0 \\ 0 & 0 & \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} & -\frac{2z_{\text{near}} z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{7.40}$$

如果透视投影观察体一开始就指定为对称棱台，则可用裁剪窗口的视场角和尺寸来表达规范化透视变换的元素。这样，利用（7.29）式，在投影参考点位于原点且观察平面在近裁剪平面位置时，有

$$\mathbf{M}_{\text{normsymmpers}} = \begin{bmatrix} \frac{\cot\left(\frac{\theta}{2}\right)}{\text{aspect}} & 0 & 0 & 0 \\ 0 & \cot\left(\frac{\theta}{2}\right) & 0 & 0 \\ 0 & 0 & \frac{z_{\text{near}} + z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} & -\frac{2z_{\text{near}} z_{\text{far}}}{z_{\text{near}} - z_{\text{far}}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{7.41}$$

从世界坐标到规范化透视投影坐标的完整变换是由该透视矩阵从左边与观察变换合并所得的矩阵$\mathbf{R} \cdot \mathbf{T}$。接下来，将裁剪子程序应用于规范化观察体。余下的任务是可见性检测、表面绘制及变换到视口。

7.9 视口变换和三维屏幕坐标系

完成向规范化投影坐标的变换后，可高效地对对称立方体（或单位立方体）进行裁剪。裁剪以后，规范化观察体的内容转变到屏幕坐标系。对于规范化的裁剪窗口中的x和y位置，该过程与

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

6.3 节讨论的二维视口变换相同。但三维观察体的所有位置还有一个深度（z 坐标），我们需要将该深度信息留给可见性检测和表面绘制时使用。因此，现在可以把视口变换看成向**三维屏幕坐标系**（three-dimensional screen coordinate）的映射。

从规范化裁剪窗口到矩形视口位置的 x、y 转换方程在矩阵（6.10）给出。可以通过结合将 z 坐标转换到屏幕坐标系的参数而将该矩阵引入到三维应用中。在对称立方体中的规范化 z 值常再次规范化到 0 至 1.0 的范围。这是将监视器屏幕看成 $z=0$，而深度处理可方便地在 0 到 1 的单位区间内进行。如果包括了这个 z 坐标的再次规范化，从规范化的观察体到三维屏幕坐标系的变换是

$$\mathbf{M}_{\text{normviewvol,3D screen}} = \begin{bmatrix} \frac{xv_{\max} - xv_{\min}}{2} & 0 & 0 & \frac{xv_{\max} + xv_{\min}}{2} \\ 0 & \frac{yv_{\max} - yv_{\min}}{2} & 0 & \frac{yv_{\max} + yv_{\min}}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.42}$$

在规范化坐标系中，对称立方体的面 $z_{\text{norm}} = -1$ 对应于裁剪窗口区域。规范化立方体的这一面映射到矩形视口，即现在的 $z_{\text{screen}} = 0$。因此，视口屏幕区域的左下角位于 $(xv_{\min}, yv_{\min}, 0)$ 而右上角位于 $(xv_{\max}, yv_{\max}, 0)$。

视口的每一个 xy 位置与刷新缓存中的一个位置相对应，该位置包含了屏幕上这一点的颜色信息。而每一屏幕点的深度值存放在称为深度缓存的另一个缓存区。在后面几章中，我们将讨论确定可见表面位置和它们的颜色的算法。

利用和二维应用中一样的方式将矩形视口定位于屏幕上。视口左下角通常定位在与显示窗户左下角对应的坐标位置。如果将该视口的纵横比设成和裁剪窗口一样则保持了对象的比例。

7.10 OpenGL 三维观察函数

OpenGL 实用库（GLU）包括了一个指定三维观察参数的函数和另一个设定对称透视投影变换的函数。另一些用于正交投影、斜平行投影和视口变换的函数则包括在 OpenGL 基本库中。另外，还有一些用于定义和管理显示窗口的 GLUT 函数（参见 6.4 节）。

7.10.1 OpenGL 观察变换函数

在 OpenGL 指定观察参数时，生成一个矩阵并和当前建模观察矩阵合并。同时，该观察矩阵和任何可能已指定的几何变换矩阵相结合。然后将该组合矩阵用于将世界坐标系的对象描述变换到观察坐标系中。建模观察模式用下列语句来设定：

```
glMatrixMode (GL_MODELVIEW);
```

观察参数用下列 GLU 函数指定，该函数因引用 OpenGL 基本库中的平移和旋转子程序而归入 OpenGL 实用库。

```
gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
```

该函数中所有参数的值均赋以双精度浮点值。该函数指定观察参考系原点在世界坐标系的 $\mathbf{P}_0 = (x0, y0, z0)$ 处，而参考点在 $\mathbf{P}_{\text{ref}} = (xref, yref, zref)$ 处，向上向量为 $\mathbf{V} = (Vx, Vy, Vz)$。观察坐标系的正 z_{view} 轴在 $\mathbf{N} = \mathbf{P}_0 - \mathbf{P}_{\text{ref}}$ 方向，而观察参考系的单位轴向量用等式（7.1）计算。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

由于观察方向是沿 $-z_{view}$ 轴，故参考点 $\mathbf{P}_{ref}$ 也称为“视点”。它常被定位在场景中心的某个位置，以便作为指定投影参数的参考。因而可将该参考位置看成安放在观察坐标系原点的照相机要瞄准的一个点。照相机的向上方向用垂直于 **N** 的向量 **V** 指定。

gluLookAt 函数指定的观察参数用来形成7.4 节中给出的观察变换矩阵（7.4）。该矩阵由将观察原点移到世界原点的平移和将观察坐标轴与世界坐标轴对齐的旋转组合而成。

如果不引用 gluLookAt 函数，默认的 OpenGL 参数是

$$\mathbf{P}_0 = (0, 0, 0)$$
$$\mathbf{P}_{ref} = (0, 0, -1)$$
$$\mathbf{V} = (0, 1, 0)$$

使用这些默认值，观察坐标系和世界坐标系相同，观察方向沿负的 z_{world} 轴方向。在许多应用中，我们可以很方便地使用这些观察参数的默认值。

7.10.2　OpenGL 正交投影函数

投影矩阵按 OpenGL 投影模式存储。因此，建立一个投影变换矩阵，必须先用下面的语句引入该模式。

```
glMatrixMode (GL_PROJECTION);
```

然后，可使用任一变换命令，由此导致的矩阵将和当前投影矩阵合并。

正交投影参数用下列函数选择：

```
glOrtho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

该函数中所有参数均赋以双精度浮点数。glOrtho 用来选择裁剪窗口坐标和观察系原点到近和远裁剪平面的距离。OpenGL 中不提供对观察平面的选择功能。近裁剪平面永远和观察平面重合，因为裁剪窗口永远位于观察体的近平面上。

函数 glOrtho 生成一个垂直于观察平面（近裁剪平面）的平行投影。因此，该函数为指定的裁剪平面和裁剪窗口建立一个有限的正交投影观察体。在 OpenGL 中，近和远裁剪平面不是任选的，它们必须为任一投影变换而指定。

参数 dnear 和 dfar 给出从观察坐标系原点沿负 z_{view} 轴方向的距离。假如 dfar = 55.0，则远裁剪平面位于坐标位置 $z_{far} = -55.0$。两个参数中任一个负值表明在观察原点“后面”、沿正 z_{view} 轴的距离。这些参数可赋以任一值（正、负、或零），只要 dnear < dfar。

为这一投影变换形成的观察体是一个矩形平行管道。该观察体的坐标位置由矩阵（7.7）在 $z_{near} = -\text{dnear}$ 及 $z_{far} = -\text{dfar}$ 条件下变换到左手系中的对称规范化立方体。

OpenGL 正交投影函数的默认参数值是 ±1，生成一个在右手观察坐标系中的对称规范化立方体。该默认情况与使用下列语句等价：

```
glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

于是，默认的裁剪窗口是一个对称规范化正方形，而默认的观察体是一个 $z_{near} = 1.0$（在观察位置后）且 $z_{far} = -1.0$ 的对称规范化立方体。图 7.54 给出了默认正交投影观察体的外貌和位置。

对于二维应用而言，gluOrtho2D 函数用来建立裁剪窗口。glOrtho 函数也可用来指定裁剪窗口，并将参数 dnear 和 dfar 指定在坐标系原点的另一侧。事实上，调用 gluOrtho2D 与在 dnear = −1.0 和 dfar = 1.0 的情况下调用 glOrtho 是等价的。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

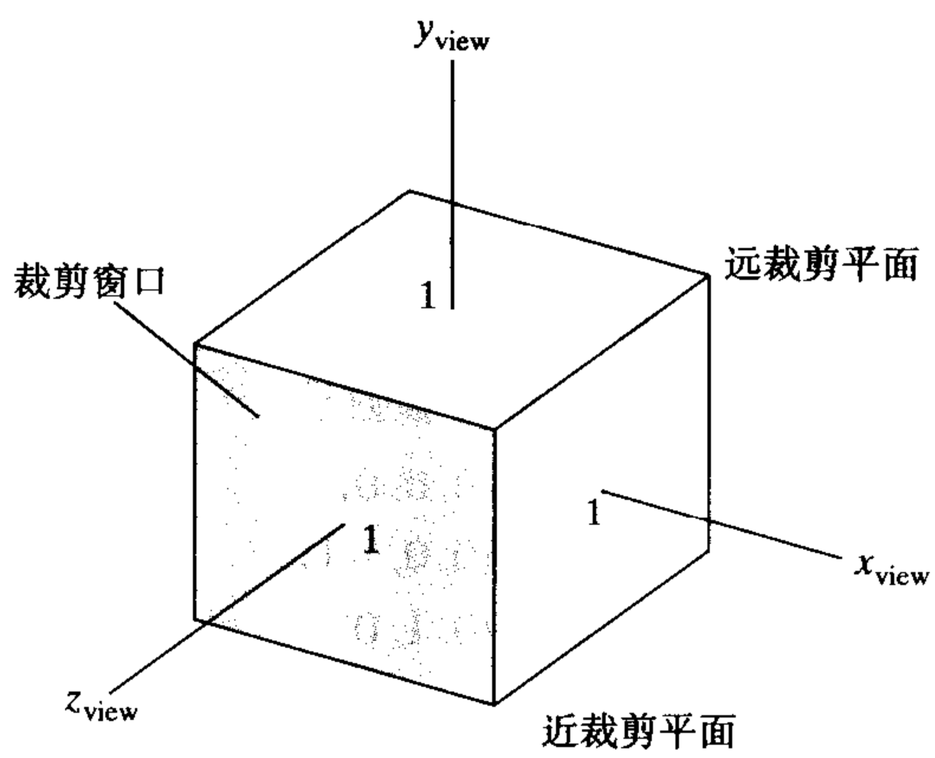


图 7.54　默认的正交投影观察体。该对称立方体的坐标范围是每个方向从 -1 到 +1。近裁剪平面位于 $z_{near}=1$，而远裁剪平面位于 $z_{far}=-1$

OpenGL中没有生成斜平行投影的函数。如果要生成一个斜平行投影，可设定专门的等式（7.14）那样的投影矩阵。然后使用5.17节给出的矩阵函数将其设成当前OpenGL投影矩阵。另一种生成斜投影视图的方法是将场景旋转到合适的位置，从而使 z_{view} 方向的正交投影给出所需的视图。

7.10.3　OpenGL 对称透视投影棱台

有两个函数用来生成透视投影视图。一个函数生成一个关于观察方向（负 z_{view} 轴）的对称锥形观察体。另一个函数用于对称透视投影或斜透视投影。两个函数中的投影参考点都是观察坐标系原点且近裁剪平面是观察平面。

对称透视投影棱台观察体用下列 GLU 函数建立：

```
gluPerspective (theta, aspect, dnear, dfar);
```

四个参数均赋以双精度浮点值。前两个参数定义近裁剪平面上的裁剪窗口尺寸和位置，而后两个参数指定从观察点（坐标系原点）到近和远裁剪平面的距离。参数 theta 表示视场角，即上下裁剪平面间的夹角（参见图 7.48）。该角可赋以 0° 到 180° 之间的任何角度。参数 aspect 赋以裁剪窗口纵横比（宽度 / 高度）的值。

对于OpenGL的一个透视投影，近和远裁剪平面必须永远位于负 z_{view} 轴；都不可在观察位置的“背后”。该约束对正交投影无效，但当观察平面位于观察点后面时预先包含了倒转的透视投影。因此，dnear 和 dfar 都必须赋以正值，且近和远平面的位置记为 $z_{near}=-\text{dnear}$ 及 $z_{far}=-\text{dfar}$。

如果不指定投影函数，则使用默认正交投影来显示场景。此时，观察体是图 7.54 所示的一个对称规范化立方体。

由 gluPerspective 函数建立的棱台观察体关于负 z_{view} 轴对称。场景的描述用矩阵（7.41）转换到规范化齐次投影坐标系。

7.10.4　OpenGL 通用透视投影函数

下列函数可用来指定一个对称棱台观察体或一个斜棱台观察体的透视投影：

```
glFrustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

该函数中的所有参数均赋以双精度浮点数。和其他观察投影函数一样，近平面是观察平面且投影参考点位于观察位置（坐标系原点）。该函数与正交平行投影函数有着相同的参数，但这里的近和远

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

裁剪平面距离必须是正的。前四个参数设定近平面上裁剪窗口的坐标，而后两个参数指定从坐标系原点沿负 z_{view} 轴到近和远裁剪平面的距离。近和远裁剪平面的定位计算为 $z_{near} = -dnear$ 及 $z_{far} = -dfar$。

裁剪窗口可指定在近平面上的任意位置。如果将裁剪窗口选择为 $xw_{min} = -xw_{max}$ 且 $yw_{min} = -yw_{max}$，就得到一个对称棱台（以负 z_{view} 轴作为其中心线）。

同样，如果不显式地引入一个投影命令，OpenGL 对场景提供默认的正交投影。此时的观察体是一个对称立方体（参见图 7.54）。

7.10.5　OpenGL 视口和显式窗口

在规范化坐标系中使用裁剪子程序后，规范化裁剪窗口中的内容及其深度信息变换到三维屏幕坐标。视口中每一 xy 位置的颜色值存储到刷新缓存（颜色缓存）中，而每一 xy 位置的深度信息则存储到深度缓存中。

如 6.4 节所述，矩形视口用下列 OpenGL 函数定义：

```
glViewport (xvmin, yvmin, vpWidth, vpHeight);
```

该函数中前两个参数指定对应于显示窗口左下角的视口左下角的整型屏幕坐标。而后两个参数给出整型的视口宽度和高度。如果要保持场景对象的比例，须将视口纵横比设定为与裁剪窗口一致。

显示窗口用 GLUT 子程序创建并管理，6.4 节详细地讨论了 GLUT 库中的各种显示窗口函数。OpenGL 中的默认视口是当前显示窗口的尺寸和位置。

7.10.6　OpenGL 三维观察程序示例

下列程序显示了图 7.55 给出的正方形的透视投影视图。该正方形在 xy 平面上定义，而观察坐标系原点按一定角度观察前平面来选定。选择正方形的中心为注视点，使用 glFrustum 函数得到一个透视视图。如果将观察原点移到多边形的另一面，则按对象的线框图形式显示后向面。

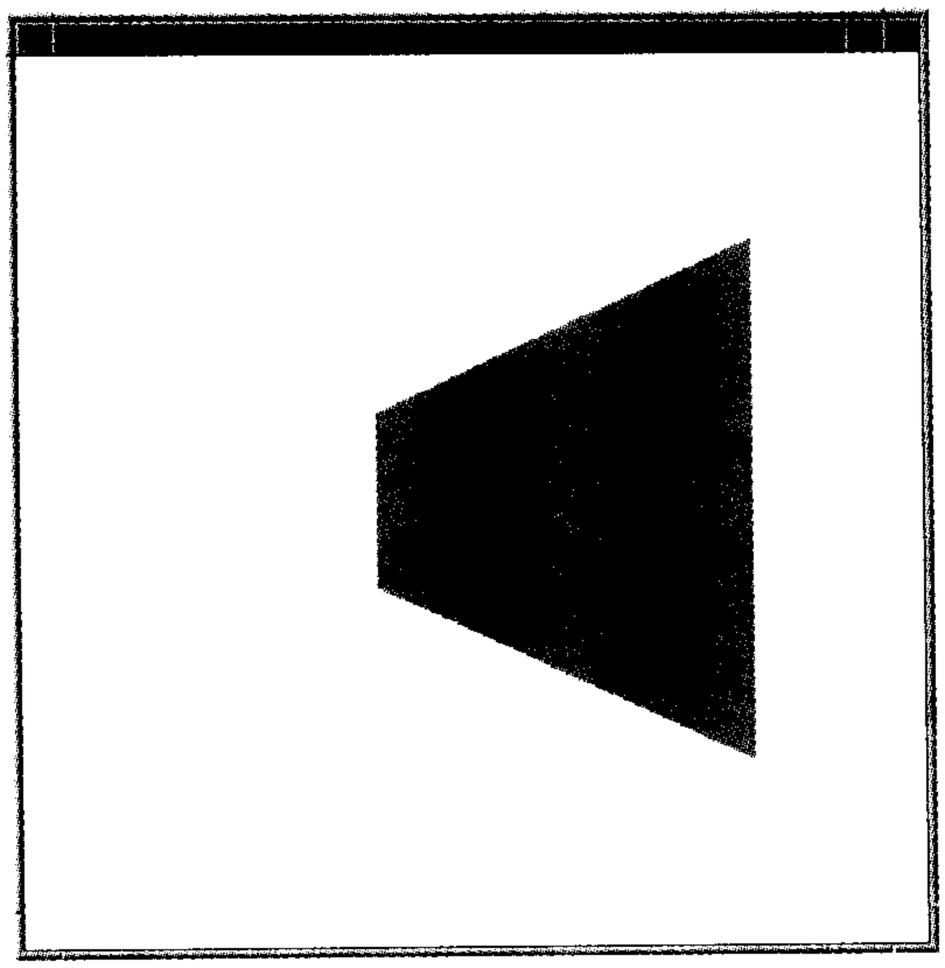


图 7.55　三维观察示例程序的输出显示

```
#include <GL/glut.h>

GLint winWidth = 600, winHeight = 600;   //  Initial display-window size.
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
GLfloat x0 = 100.0, y0 = 50.0, z0 = 50.0;  //  Viewing-coordinate origin.
GLfloat xref = 50.0, yref = 50.0, zref = 0.0;  //  Look-at point.
GLfloat Vx = 0.0, Vy = 1.0, Vz = 0.0;          //  View-up vector.

/*  Set coordinate limits for the clipping window:  */
GLfloat xwMin = -40.0, ywMin = -60.0, xwMax = 40.0, ywMax = 60.0;

/*  Set positions for near and far clipping planes:  */
GLfloat dnear = 25.0, dfar = 125.0;

void init (void)
{
   glClearColor (1.0, 1.0, 1.0, 0.0);

   glMatrixMode (GL_MODELVIEW);
   gluLookAt (x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);

   glMatrixMode (GL_PROJECTION);
   glFrustum (xwMin, xwMax, ywMin, ywMax, dnear, dfar);
}

void displayFcn (void)
{
   glClear (GL_COLOR_BUFFER_BIT);

   /*  Set parameters for a square fill area.  */
   glColor3f (0.0, 1.0, 0.0);          //  Set fill color to green.
   glPolygonMode (GL_FRONT, GL_FILL);
   glPolygonMode (GL_BACK, GL_LINE);  //  Wire-frame back face.
   glBegin (GL_QUADS);
      glVertex3f (0.0, 0.0, 0.0);
      glVertex3f (100.0, 0.0, 0.0);
      glVertex3f (100.0, 100.0, 0.0);
      glVertex3f (0.0, 100.0, 0.0);
   glEnd ( );

   glFlush ( );
}

void reshapeFcn (GLint newWidth, GLint newHeight)
{
   glViewport (0, 0, newWidth, newHeight);

   winWidth = newWidth;
   winHeight = newHeight;
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (50, 50);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Perspective View of A Square");

   init ( );
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    glutDisplayFunc (displayFcn);
    glutReshapeFunc (reshapeFcn);
    glutMainLoop ( );
}
```

7.11 三维裁剪算法

在第6章中，我们讨论了二维裁剪算法中使用裁剪窗口规范边界的优点。同样，可将三维裁剪算法应用于观察体的规范化边界。这保证了高效率地实现观察流水线和裁剪过程。所有与设备无关的变换（几何和观察）合并起来后在裁剪子程序之前应用。规范化观察体的每一裁剪边界是一个平行于某个笛卡儿平面的平面，而不管投影类型和观察体的原始形状。依赖于观察体是否规范化成一个单位立方体或一个边长为2的对称立方体，裁剪平面的坐标位置为0和1或-1和1。对称立方体的三维裁剪平面为

$$\begin{aligned} xw_{\min} &= -1, \qquad xw_{\max} = 1 \\ yw_{\min} &= -1, \qquad yw_{\max} = 1 \\ zw_{\min} &= -1, \qquad zw_{\max} = 1 \end{aligned} \tag{7.43}$$

x 和 y 裁剪边界是裁剪窗口的规范化范围，而 z 裁剪边界是近和远裁剪平面的规范化位置。

三维观察裁剪算法识别并保存在规范化观察体内的对象部分并显示在输出设备上。所有在观察体裁剪平面以外的对象部分均被消除。现在的算法是使用观察体的规范化边界平面来代替规范化窗口的直线边界后二维方法的扩充。

7.11.1 三维齐次坐标系中的裁剪

计算机图形库将空间位置当做四维齐次坐标来处理，因此所有变换均可用4 × 4矩阵来表示。每一坐标位置进入观察流水线时，它被转换成四维表示：

$$(x, y, z) \rightarrow (x, y, z, 1)$$

在一个位置经过几何、观察和投影变换后，成为一个齐次形式：

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{7.44}$$

这里矩阵 **M** 表示从世界坐标到规范化、齐次投影坐标的所有各种变换的组合，而齐次参数 h 可能不再是1。事实上，h 可以有任意实数值，依赖于我们如何在场景中表达对象及使用的投影类型。

如果该齐次参数 h 确实为1，则该齐次坐标与笛卡儿坐标相同。对于平行投影变换来说常常如此。但是，透视投影生成的齐次坐标是任一空间点 z 坐标的函数。透视投影的齐次参数甚至可能为负值。这在坐标位置位于投影参考点之后时发生。同样，对象曲面的有理样条表示常在齐次坐标中形成，其中齐次坐标可正可负。因此，如果裁剪在投影坐标系中除以齐次参数 h 后再完成，则可能失去某些坐标信息而使裁剪不正确。

处理所有可能的投影变换和对象表示的一个有效方法，是在空间位置的齐次坐标系中进行裁剪。而由于所有观察体可转换到一个规范化立方体，因此可以用硬件来完成单一的裁剪过程，在齐次坐标系中用规范化裁剪平面来裁剪对象。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

7.11.2 三维区域码

区域码的概念（参见6.7节）可以通过添加一对表示近和远裁剪平面的位来扩展到三维中。这样，我们使用六位区域码，如图7.56所示。该区域码例子中位的位置从右到左编号，在该次序中代表左、左、下、上、近和远裁剪平面。

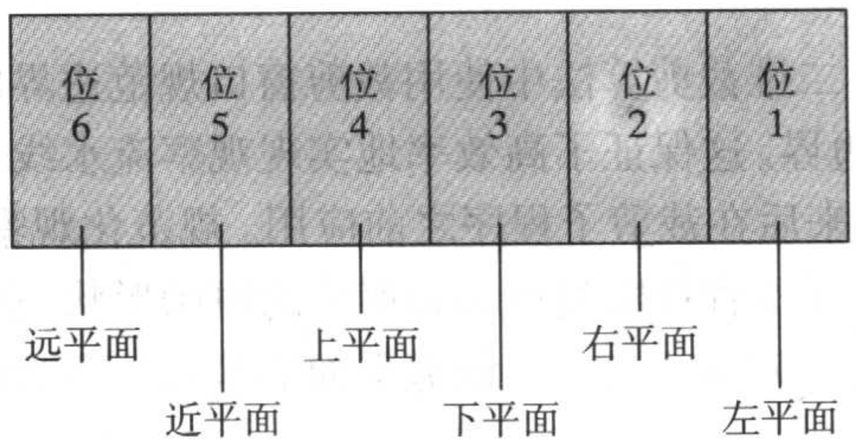


图 7.56 与区域码位的位置对应的观察体裁剪平面的一个可能次序

设定区域码位值的条件与6.7节中相同，加上了用于近和远裁剪平面的两个附加条件。然而，对于一个三维场景，我们需对变换到规范化空间的投影坐标使用裁剪子程序。在投影变换后，场景每一点有四元素表示 $\mathbf{P} = (x_h, y_h, z_h, h)$。假设用规范化对称立方体（参见等式（7.43））的边界进行裁剪，则当一点的投影坐标满足如下六个不等式时该点位于规范化观察体的内部。

$$-1 \leqslant \frac{x_h}{h} \leqslant 1, \qquad -1 \leqslant \frac{y_h}{h} \leqslant 1, \qquad -1 \leq \frac{z_h}{h} \leqslant 1 \tag{7.45}$$

该齐次参数 h 不会为0，除非出现错误。但我们可以在执行区域码过程之前先检查齐次参数为0或一个太小的数的可能性。齐次参数也可以为正或为负。因此，假设 $h \neq 0$，可将前面的不等式写成下列形式：

$$\begin{aligned} -h \leqslant x_h \leqslant h, \qquad -h \leqslant y_h \leqslant h, \qquad -h \leqslant z_h \leqslant h \quad \text{如果} h > 0 \\ h \leqslant x_h \leqslant -h, \qquad h \leqslant y_h \leqslant -h, \qquad h \leqslant z_h \leqslant -h \quad \text{如果} h < 0 \end{aligned} \tag{7.46}$$

多数情况下 $h > 0$，因而可按下列测试对一坐标位置的区域码的位值进行赋值：

$$\begin{aligned} &\text{位}1 = 1 \quad \text{如果} h + x_h < 0 \quad (\text{左}) \\ &\text{位}2 = 1 \quad \text{如果} h - x_h < 0 \quad (\text{右}) \\ &\text{位}3 = 1 \quad \text{如果} h + y_h < 0 \quad (\text{上}) \\ &\text{位}4 = 1 \quad \text{如果} h - y_h < 0 \quad (\text{下}) \\ &\text{位}5 = 1 \quad \text{如果} h + z_h < 0 \quad (\text{近}) \\ &\text{位}6 = 1 \quad \text{如果} h - z_h < 0 \quad (\text{远}) \end{aligned} \tag{7.47}$$

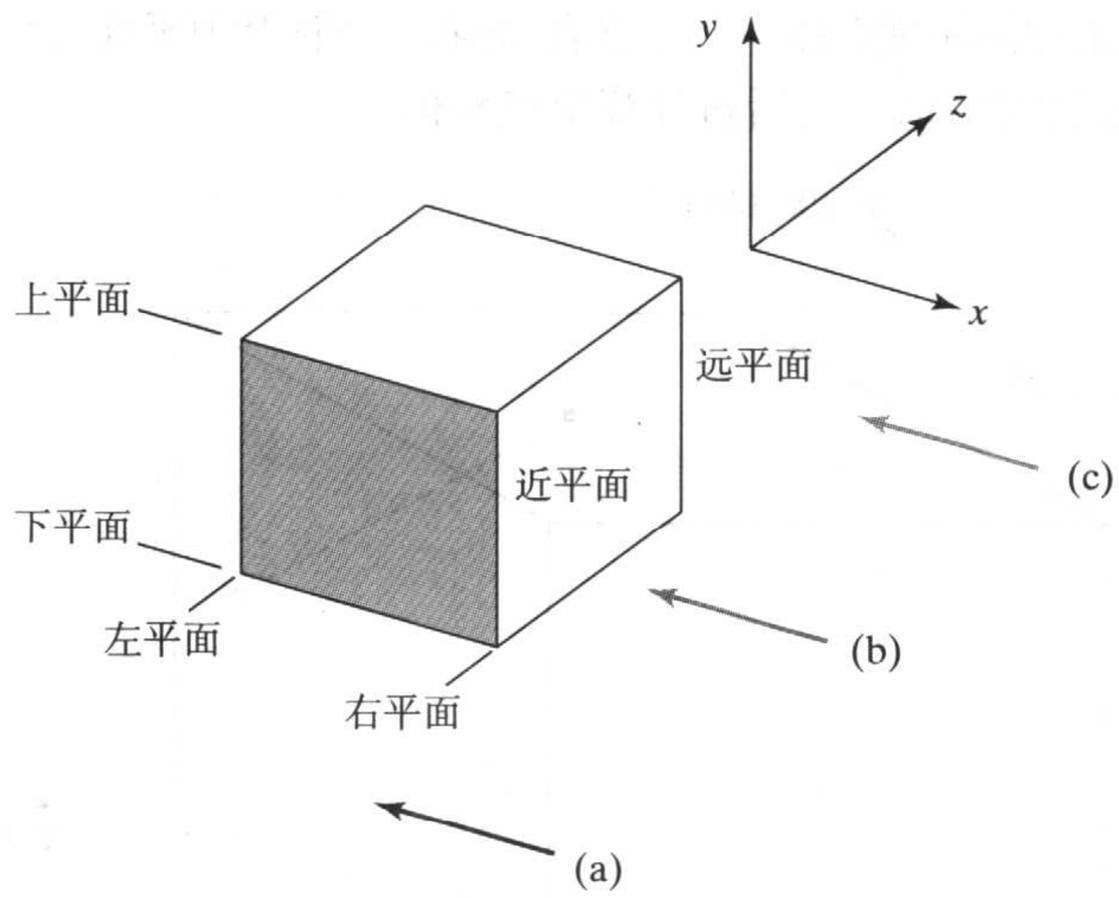
这些位值可使用二维裁剪中相同的方法来设定。即简单地使用计算 $h \pm x_h$、$h \pm y_h$ 或 $h \pm z_h$ 的符号来设定对应的区域码位值。图7.57列出了观察体的27个区域码。对某些 $h < 0$ 的点，可使用表达式（7.46）中的第二套不等式进行裁剪或对坐标取负且用 $h > 0$ 时的测试进行裁剪。

7.11.3 三维点和线的裁剪

对于不在投影参考点后面定义的场景中的一般的点和直线段，所有齐次参数为正且可用表达式（7.47）中的条件建立区域码。然后，在设定场景中每一位置的区域码后，可方便地确认一个点是在观察体外还是体内。例如，区域码为101000表示该点在观察体的上面及后面，区域码为000000表示该点在体内（参见图7.57）。因此，对于点的裁剪，简单地将区域码不为000000的点消除。换句话说，如果表达式（7.47）测试中任一结果为负，该点位于观察体外。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com



011001	011000	011010
010001	010000	010010
010101	010100	010110

近平面之前的区域码
(a)

001001	001000	001010
000001	000000	000010
000101	000100	000110

近和远平面之间的区域码
(b)

101001	101000	101010
100001	100000	100010
100101	100100	100110

远平面之后的区域码
(c)

图 7.57 标志与观察体空间位置关系的三维六位区域码的值

三维线裁剪的方法基本上与二维线裁剪相同。我们可以先对一线段两端点的区域码测试以确定完全接受或拒绝线段。如果一线段两端点的区域码均为000000，则该线段完全在观察体的内部。同样地，如果两端点区域码的逻辑或操作生成值0则可完整地接受该线段。如果线段两端点区域码与操作结果为非零，则可完整地拒绝该线段。该非零值表明两个端点的区域码在某一位均为1，因此该线段完整地在某一裁剪平面之外。例如，图7.58中$\mathbf{P}_3$到$\mathbf{P}_4$的线段的端点区域码为010101和100110。因而该线段完全在下裁剪平面之下。如果一线段对这两个测试均失败，则接下来分析线段方程以确定是否该线段有一部分要保留。

利用参数形式可方便地表达三维线段方程，且Cyrus-Beck或梁友栋-Barsky裁剪方法（参见6.7节）可以扩充到三维场景。对一个端点为$\mathbf{P}_1 = (x_{h1}, y_{h1}, z_{h1}, h_1)$和$\mathbf{P}_2 = (x_{h2}, y_{h2}, z_{h2}, h_2)$的线段，可以写出表示线段上任意一点的参数方程，如下所示：

$$\mathbf{P} = \mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1)u \qquad 0 \leqslant u \leqslant 1 \tag{7.48}$$

当线段参数值$u=0$时，在位置$\mathbf{P}_1$；而$u=1$时在位置$\mathbf{P}_2$。在齐次坐标下用显式参数线段方程表示，有

$$\begin{aligned} x_h &= x_{h1} + (x_{h2} - x_{h1})u \\ y_h &= y_{h1} + (y_{h2} - y_{h1})u \\ z_h &= z_{h1} + (z_{h2} - z_{h1})u \\ h &= h_1 + (h_2 - h_1)u \end{aligned} \qquad 0 \leqslant u \leqslant 1 \tag{7.49}$$

线段使用端点区域码，可以先判断出哪个裁剪平面与它相交。如果一个端点的某位为0而另一端点的相同位为1，则该线段穿过相应的裁剪边界。换句话说，表达式（7.47）中的一个测试生成

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

一个负值，而对另一端点的相同测试生成一个非负值。要找出与该裁剪平面的交点，先使用等式（7.49）中合适的方程确定参数 u 的对应值。然后再计算交点坐标。

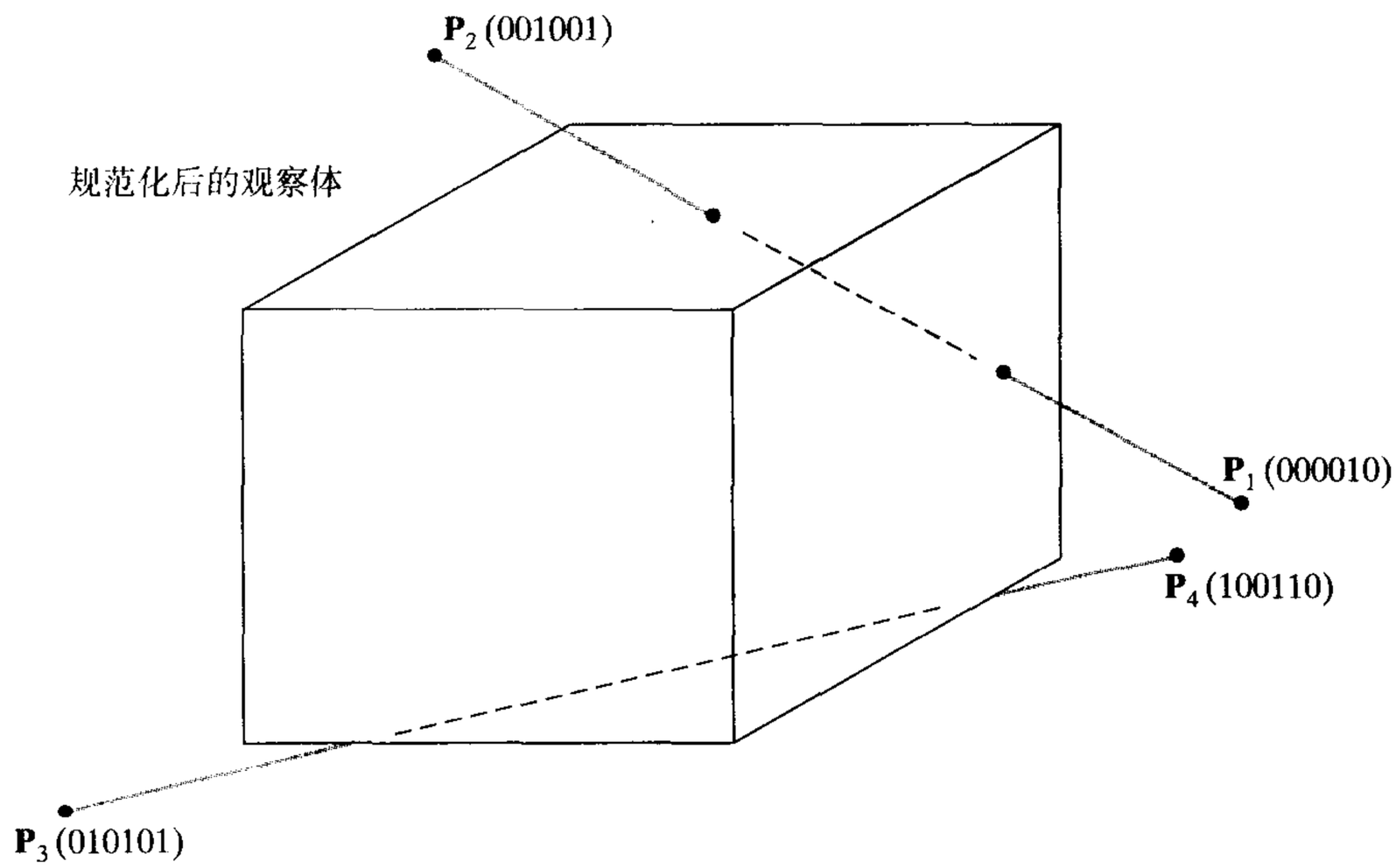


图 7.58 两条线段的三维区域码。线段 $\overline{\mathbf{P}_1\mathbf{P}_2}$ 与观察体的右和上裁剪平面相交，而线段 $\overline{\mathbf{P}_3\mathbf{P}_4}$ 完全位于下裁剪平面之下

作为交点计算过程的例子，考虑图 7.58 中的线段 $\overline{\mathbf{P}_1\mathbf{P}_2}$。该线段与左裁剪平面相交，可用等式 $x_{max}=1$ 来描述。因此，通过设定 x 投影坐标等于 1 来确定参数 u 在交点处的值：

$$x_p = \frac{x_h}{h} = \frac{x_{h1} + (x_{h2} - x_{h1})u}{h_1 + (h_2 - h_1)u} = 1 \tag{7.50}$$

求解参数 u，得

$$u = \frac{x_{h1} - h_1}{(x_{h1} - h_1) - (x_{h2} - h_2)} \tag{7.51}$$

接下来，使用计算所得的 u 值，确定该裁剪平面的 y_p 和 z_p 值。此时，y_p 和 z_p 交点值在观察体的±1 边界之间且该线段穿过观察体内部。因此我们接着定位与上裁剪平面的交点位置。由于与上和右裁剪平面的交点标志该线段位于观察体内部的部分，而该线段的所有其他部分均位于观察体的外部，所以对该线段的处理全部完成。

当一线段与一个裁剪边界相交但并不进入观察体内部时，继续如二维裁剪一样处理下去。消除位于该裁剪边界之外的线段部分，更新在该边界内的线段部分的区域码信息和参数 u 的值。然后用另外的裁剪平面测试余下的线段部分，确定是否有拒绝的可能或需进一步的求交计算。

三维场景中的线段通常不对称。它们常常是场景中实体对象的描述中的组成部分，而我们需要像表面裁剪子程序的一部分那样去处理这些线段。

7.11.4 三维多边形裁剪

图形软件包一般处理包含图形对象的场景。这些对象的边界用线性方程描述，因而每个对象由一组多边形面组成。因此，我们使用多边形面裁剪子程序来裁剪三维场景中的对象。例如，图 7.59 醒目显示了一个四面体被裁剪掉的表面部分，而将位于观察体内部的多边形表面用虚线显示。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

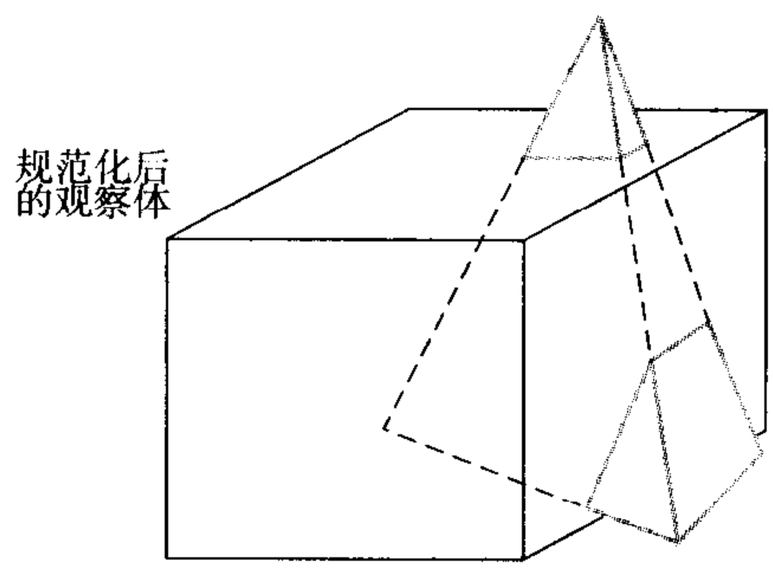


图 7.59　三维对象裁剪。从对象描述中消除在观察体裁剪平面外的表面部分，且可能需要建立新的表面片

我们可先使用坐标范围的包围盒、包围球或某种其他方法来测试多面体是否可完全接受或拒绝。如果该对象的坐标范围在所有裁剪平面的内部，则保留整个对象。如果坐标范围整体位于任一裁剪边界的外部，则消除整个对象。

在不能保留或消除整个对象时，则可接下来处理定义对象表面的多边形的顶点表。使用与二维多边形裁剪类似的方法，可裁剪边以获得对象表面的新顶点表。裁剪操作可能导致创建增加表面的新顶点表。并更新多边形表以加入新多边形及修改表面的连接性和共享边信息。

为了简化一般多面体的裁剪，多边形表面常分解成三角形部分并用三角形带来描述。我们可使用3.15节中讨论的Sutherland-Hodgman方法来裁剪三角形带。每一三角形带用6个裁剪平面依次处理，以获得该带的最后顶点表。

例如，可对凸多边形使用 3.15 节中的分割方法获得一组三角形，然后裁剪这些三角形。或者可使用 6.8 节中叙述的 Weiler-Atherton 算法裁剪三维凸多边形。

7.11.5　三维曲面裁剪

和多面体裁剪一样，先检查曲面对象（如球面或样条曲面）的坐标范围是否完全在观察体内。然后再检查对象是否完全在六个裁剪平面中任意一个的外部。

如果简单的拒绝-接受测试失败，则接下来定位与裁剪平面的交点。为此，求解联立的面方程和裁剪平面方程。由于这一原因，多数图形包并不包括曲线对象的裁剪子程序。而常用一组多边形面片逼近曲面，使之可用多边形裁剪子程序来裁剪。在对多边形面片使用表面绘制过程后仍可提供曲面的高真实感显示。

7.11.6　任意裁剪平面

在有些图形软件中还有可能使用任意空间方向定义的另外的曲面来裁剪三维场景。这在许多应用中很有用。例如，我们可能希望隔开或裁剪一个非规则形状的对象，为了特殊的效果而以斜角度消除场景的一部分，或沿一选定轴切去对象的一部分来展示其内部的剖面视图。

任选的裁剪平面可与场景描述一起指定，因而裁剪操作可在投影变换之前完成。然而，这也意味着该裁剪子程序可用软件实现。

一个裁剪平面可用平面参数 A、B、C 和 D 来指定。该平面将三维空间分为两部分，因此一个场景位于该平面一侧的所有部分被裁剪掉。假设在该平面之后的对象部分被裁剪掉，则任何满足下列不等式的空间位置(x, y, z)从场景中消除。

$$Ax + By + Cz + D < 0 \tag{7.52}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

作为一个例子，如果该平面参数数组$(A, B, C, D) = (1.0, 0.0, 0.0, 8.0)$，则任一满足$x+8.0 < 0.0$（或$x < -8.0$）的坐标位置从场景中消除。

如果要裁剪一线段，首先测试其两端点，确定该线段是否完全位于该裁剪平面的后面或完全在该平面的前面。可以使用法向量$\mathbf{N} = (A, B, C)$给出不等式（7.52）的向量表达式。然后，对以$\mathbf{P}_1$和$\mathbf{P}_2$为端点的一条线段，如果两端点均满足下列不等式则裁掉整条线段。

$$\mathbf{N} \cdot \mathbf{P}_k + D < 0, \qquad k = 1, 2 \tag{7.53}$$

而当两端点都满足下列不等式时保留整条线段：

$$\mathbf{N} \cdot \mathbf{P}_k + D \geqslant 0, \qquad k = 1, 2 \tag{7.54}$$

否则，两端点位于该裁剪平面的两侧，如图7.60所示，且需计算该线段与平面的交点。

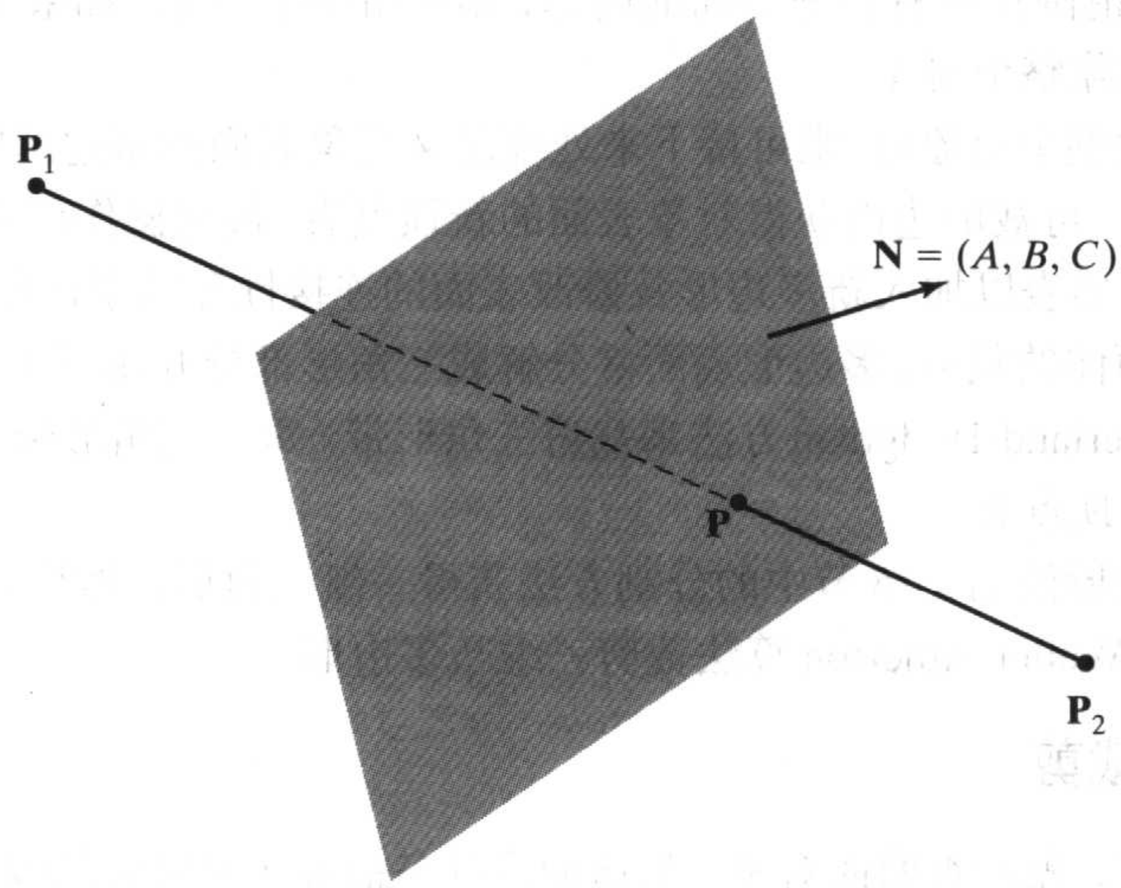


图7.60 使用法向量为**N**的平面裁剪一线段

如果要计算线段与裁剪平面的交点，可以使用线段的下列参数表示：

$$\mathbf{P} = \mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1)u, \qquad 0 \leqslant u \leqslant 1 \tag{7.55}$$

如果点**P**满足下面的平面方程，则它位于该裁剪平面上：

$$\mathbf{N} \cdot \mathbf{P} + D = 0 \tag{7.56}$$

与等式（7.55）中**P**的表示相减，得

$$\mathbf{N} \cdot [\mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1)u] + D = 0 \tag{7.57}$$

求解等式中的参数u，得

$$u = \frac{-D - \mathbf{N} \cdot \mathbf{P}_1}{\mathbf{N} \cdot (\mathbf{P}_2 - \mathbf{P}_1)} \tag{7.58}$$

将u值代入等式（7.55）中的向量参数线段表达式，可得x、y和z交点坐标。对于图7.60中的例子，线段$\mathbf{P}_1$到**P**被裁掉而保留线段**P**到$\mathbf{P}_2$。

对于一个多面体，如图7.61中的四边体，应用类似的过程。首先测试对象是否完全在裁剪平面的后面或完全在前面。如不是这两种情况，则处理多面体每个面的顶点。依次对每个多面形边使用和观察体裁剪一样的线裁剪方法来获得表面顶点表。但此时只需处理一个裁剪平面。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

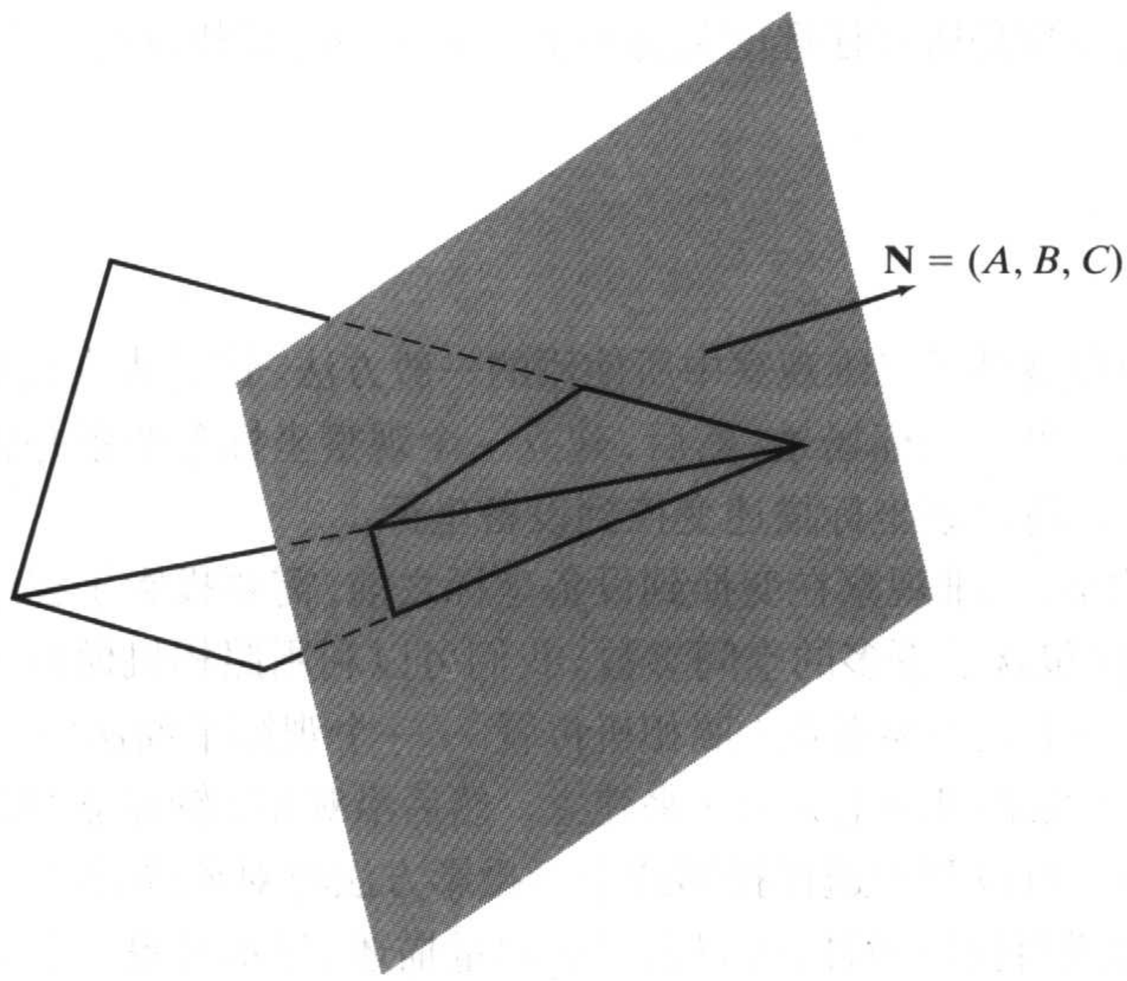


图7.61 用法向量**N**表示的平面裁剪一个四边形的表面。保留在该平面前面的表面，而清除在后面的表面

使用单个裁剪平面裁剪一个曲线对象比用观察体的六个面裁剪一个对象要容易得多。但仍需通过求解一组非线性方程来定位交点，除非使用直线段逼近曲线。

7.12 OpenGL任选裁剪平面

除了包围观察体的六个裁剪平面以外，OpenGL提供了对场景中其他的裁剪平面的描述功能。与每个观察体裁剪平面垂直于某个坐标轴不同，这些附加裁剪平面可以有任意方向。

使用下列语句可指定一个任选的裁剪平面并激活使用该平面的裁剪：

```
glClipPlane (id, planeParameters);
glEnable (id);
```

参数id用做裁剪平面的标识。用值GL_CLIP_PLANE0、GL_CLIP_PLANE1等来赋值，直到设备定义的最大值。然后用表示四个平面公式参数A、B、C和D的双精度浮点型四元素的数组planeParameters来定义该平面。已指定标识并已激活的裁剪平面用下列函数使其无效：

```
glDisable (id);
```

四个平面参数A、B、C和D变换到观察坐标系，并用来测试场景中观察坐标位置。随后在观察或几何变换参数方面的改变不会影响存储的平面参数。因此，如果在指定任一几何或观察变换之前建立任选的裁剪平面，存储的平面参数和输入参数相同。同样，由于这些平面的裁剪子程序对应于观察坐标操作，且不在规范化坐标空间，所以一个程序的性能在任选裁剪平面激活后会降低。

任一位于激活的OpenGL裁剪平面后面的点被消除。因此，如果一个观察坐标位置(x, y, z)满足条件（7.52）则被裁掉。

任一OpenGL实现中可提供六个或更多的任选裁剪平面。使用下列查询函数可发现某个具体的OpenGL实现中允许多少个任选裁剪平面。

```
glGetIntegerv (GL_MAX_CLIP_PLANES, numPlanes);
```

参数numPlanes是将赋以表示可用的任选裁剪平面数目的一个整数数组的名字。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

glClipPlane的默认参数是所有任选的裁剪平面参数A、B、C和D均为0。初始化时，所有任选裁剪平面均无效。

7.13 小结

三维场景的观察程序遵循了二维观察中所使用的一般方法。即首先在建模坐标或直接在世界坐标里的对象定义中创建一世界坐标场景。然后，建立一个观察坐标参照系并将对象描述从世界坐标变换到观察坐标。最后，将观察坐标描述变换到设备坐标。

与二维观察不同的是，三维观察在变换到设备坐标之前，需要投影子程序把对象描述变换到观察平面上。三维观察操作包含了更多的空间参数。我们可以利用照相机模拟过程来描述三维观察参数。观察坐标参照系由一个观察参考点（照相机位置）、一个观察平面法向量**N**（照相机镜头方向）和一个观察向上向量**V**（照相机向上方向）来建立。然后沿观察z轴建立观察平面位置，并且将对象描述投影到此平面中。可以利用透视投影或平行投影方法将对象描述变换到观察平面上。

平行投影有正投影或斜投影两种，可以由投影向量描述。显示对象一个侧面以上的正平行投影称为轴测投影。对象的等距视图通过按透视法以相同数量缩短每个主轴的轴测投影而得到。常用的斜投影是斜等测投影和斜二测投影。对象的透视投影由相交于投影参考点的投影线来获得。平行投影保持对象的比例，但透视投影缩小远距离的对象。透视投影引起一组平行线看起来会聚到一个灭点，因而在观察平面上表现为不平行。依赖于与观察平面相交的主轴数目，工程和建筑图显示采用一点、两点或三点透视投影方式生成。当从投影参考点到裁剪窗口中心的线与观察平面不垂直时得到斜透视投影。

三维场景中的对象可用观察体进行裁剪来消除场景的不需要部分。观察体的顶、底和侧面由平行于投影线的平面形成且通过观察平面中窗口的边。近和远平面（也称为前和后平面）用于生成一封闭的观察体。对于平行投影，观察体是一个平行六面体。对于透视投影，观察体是一个棱台。每一种情况下，都可以将观察体转换成边界坐标为0和1或为-1和1的规范化立方体。高效的裁剪算法用规范化观察体的边界平面对场景中的对象进行裁剪。在图形软件包中，通常在完成所有的观察和其他变换后进行齐次坐标裁剪。然后再把齐次坐标转变为三维笛卡儿坐标。也可使用方向任意的附加裁剪平面来消除场景的选定部分或生成特殊效果。

OpenGL实用库中提供了用于指定观察参数（参见表7.1）的三维观察函数。库中还有用于设定对称透视投影变换的函数。OpenGL基本库中还有三个函数用于指定正交投影、通用透视投影和任选裁剪平面。表7.1总结了本章讨论的OpenGL观察函数。另外，表中列出了一些与观察相关的函数。

表7.1 OpenGL三维观察函数小结

函数	描述
gluLookAt	指定三维观察参数
glOrtho	为正交投影指定裁剪窗口和近、远裁掉平面的参数
gluPerspective	为对称透视投影指定视场角
glFrustum	为透视投影（对称或斜）指定裁剪窗口和近、远裁剪平面的参数
glClipPlane	为任选裁剪平面指定参数

参考文献

关于三维观察和裁剪操作算法的讨论请参见Weiler and Atherton(1977)、Weiler(1980)、Cyrus and Beck(1978)以及Liang and Barsky(1984)。齐次坐标裁剪算法在Blinn and Newell(1978)、Riesenfeld

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

(1981)和 Blinn(1993, 1996, and 1998)中叙述。三维观察的各种编程技术在 Glassner(1990)、Arvo (1991)、Kirk(1992)、Heckbert(1994)和 Paeth(1995)中讨论。

三维OpenGL观察函数的完整列表在Shreiner(2000)中给出。使用三维观察的OpenGL编程例子请参见 Woo, Neider, Davis, and Shreiner(1999)。其他的编程例子可参见 Nate Robins 的网站：http://www.cs.utah.edu/~narobins/opengl.html。

练习题

7.1 给定 $\mathbf{P}_0$、**N** 和 **V**，编写一个生成将世界坐标变换到三维观察坐标的矩阵的程序。观察向上向量可以是不平行于 **N** 的任意方向。

7.2 使用平行投影方法及任意指定的投影向量，编写从多面体顶点到投影坐标的变换程序。

7.3 编写一个先进行指定旋转再获得对象的不同平行投影视图的程序。

7.4 编写一个实现对象的一点透视投影的程序。

7.5 编写一个实现对象的二点透视投影的程序。

7.6 编写一个实现对象的三点透视投影的程序。

7.7 编写一个将透视投影棱台变换到规则平行六面体的程序。

7.8 将二维Cohen-Sutherland线裁剪算法修改成用规范化对称观察体正方形裁剪三维线段的算法。

7.9 将二维梁友栋-Barsky 线裁剪算法修改成用指定的规则六面体裁剪三维线段的算法。

7.10 将二维梁友栋-Barsky线裁剪算法修改成用指定的规则六面体裁剪一给定多面体的算法。

7.11 编写一个在齐次坐标系中实现线裁剪的子程序。

7.12 开发一个按指定棱台对对象进行裁剪的算法。将其中所用的操作与按规则平行六面体裁剪所用的操作进行比较。

7.13 将Sutherland-Hodgman多边形裁剪算法推广到用规范化对称观察体对凸多边进行的裁剪算法。

7.14 编写一个实现上题算法的子程序。

7.15 编写一个在齐次坐标下实现多面体裁剪的算法。

7.16 修改 7.10 节的示例程序，允许用户为正方形的前面或后面指定视图。

7.17 修改 7.10 节的示例程序，允许将透视观察参数作为用户输入来指定。

7.18 修改 7.10 节的示例程序，使能生成任一输入多面体的视图。

7.19 修改上题中的程序，使它能用正交投影生成多面体的视图。

7.20 修改上题中的程序，使它能用斜平行投影生成多面体的视图。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第8章　三维对象的表示

图形场景能包含很多种不同类型的对象和材质表面：树、花、云、石、水、砖、木板、橡胶、纸、大理石、钢、玻璃、塑料和布等，以上仅为其中的一小部分。因此不存在某一种方法可以用来描述具有上述各种不同物质的所有特征的对象，这一点是不奇怪的。

多边形和二次曲面能够为诸如多面体和椭圆体等简单的欧氏对象提供精确的描述；样条曲面和实体几何构造技术可以用于设计机翼、齿轮及其他具有曲面的工程结构；过程式的方法，如分形结构和微粒系统，可以用来建立地形、云、草丛和其他自然景物的模型；运用相互作用力的基于物理的建模方法，可以用来描述一块布或一个胶状球的非刚性行为；八叉树编码可以用于表示对象的内部特征，如医用CT映像；等值面显示、体绘制和其他可视化技术可以应用到三维离散数据集，从而获得数据的可视表示。

实体表示方法通常为两大类，尽管不是所有的表示都能完全地属于这两类范畴中的某一个。**边界表示**（boundary representation，B-reps）使用一组曲面来描述三维对象，这些曲面将对象分为内部和外部。多边形平面和样条曲面是边界表示的典型例子。**空间分区表示**（space-partitioning representation）用来描述内部性质，将包含一个对象的空间区域划分成一组较小的、非重叠的连续实体（通常是立方体）。三维对象的一般空间划分描述是八叉树表示。这一章，我们将讨论各种表示方法的特点以及如何应用到实际中。

8.1　多面体

三维图形对象中运用边界表示的最普遍方式是使用一组包围对象内部的表面多边形。很多图形系统以一组表面多边形来存储对象的描述。由于所有表面以线性方程形式加以描述，因此会简化并加速对象的表面绘制和显示。由于这个原因，通常将多边形描述称为“标准图形对象”（standard graphics object）。某些情况下，多边形表示是惟一可用的，但很多图形软件包也允许使用其他方法来描述对象，如样条曲面，在将其转换到多边形表示后再进入观察流水线加以处理。

为了用一组多边形面片描述一个对象，我们为覆盖对象表面的每一多边形给出一组顶点。这些面片的顶点坐标和边的信息以及每一多边形的表面法向量等其他信息存入一张表中（参见3.15节）。有些图形软件提供生成由三角形或四边形组成的多边形网的子程序。这使我们可以利用一个命令来描述对象的一大块表面甚至整个表面。而有些软件则提供用多边形面片显示立方体、球体或圆柱体等普通形状的子程序。复杂的图形系统使用每秒显示一百万或更多的多边形（通常是三角形）、包括应用表面纹理和特殊的光照效果能力的硬件来快速实现多边形绘制。

8.2　OpenGL多面体函数

在OpenGL程序中有两种指定多边形表面的方法。使用3.16节的多边形图元，可以生成各种多面体形状和面片网。另外，可以使用GLUT函数来显示五种规则多面体。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.2.1　OpenGL 多边形填充函数

描述对象表面一部分的一组多边形面片，或多面体的完整描述，可用 OpenGL 图元常量 GL_POLYGON、GL_TRIANGLES、GL_TRIANGLE_STRIP、GL_TRIANGLE_FAN、GL_QUADS 和 GL_QUAD_STRIP 来给出。例如，我们可以用四边形带来装饰圆柱体侧面（轴向）。类似地，一个平行管道的所有面都可用一组三角形描述，而一个三棱锥可以用一组互相连接的三角形面指定。

8.2.2　GLUT 规则多面体函数

某些标准的形体——五种规则多面体——由GLUT库中的子程序预先定义。这些多面体也称为柏拉图（Platonic）式实体，由它的每一面是一个单位规则多边形来标志。因此，规则多边形的每条边都相等，所有边间的夹角都相等，所有面间的夹角都相等。多面体按其每一实体的面数来命名，这五个规则多面体是规则四面体（或三棱锥）、规则八面体、规则十二面体和规则二十面体。

在GLUT中有10个函数用来生成这些实体：5个生成线框图，另外5个以浓淡填充区方式显示多面体面片。显示的填充区表面特征由为场景设定的材质和光照条件来确定。每一个规则多面体在模型坐标系中描述，故每一个均以世界坐标系原点为中心。

使用下列两个函数之一可获得 4 个面的规则三棱锥。

```
glutWireTetrahedron ( );
```

或

```
glutSolidTetrahedron ( );
```

该多面体以其世界坐标系原点为中心、$\sqrt{3}$为半径（从四面体中心到任一顶点的距离）来生成。

6 个面的规则多面体（立方体）的显示函数为

```
glutWireCube (edgeLength);
```

或

```
glutSolidCube (edgeLength);
```

参数 edgeLength 可设定为任一正的、双精度浮点值，而该立方体以坐标系原点为中心。

要显示 8 个面的规则八面体，必须引入下列两个函数之一：

```
glutWireOctahedron ( );
```

或

```
glutSolidOctahedron ( );
```

该多面体有等边三角形面片且半径（从位于坐标原点的八面体中心到任一顶点的距离）为 1.0。

中心在世界坐标系原点、12 个面的十二面体的生成函数为

```
glutWireDodecahedron ( );
```

或

```
glutSolidDodecahedron ( );
```

该多面体的每一个面是一个五角形。

下面两个函数用来生成 20 个面的规则二十面体。

```
glutWireIcosahedron ( );
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

或

```
glutSolidIcosahedron ( );
```

二十面体的默认半径（从位于坐标系原点的多面体中心到任一顶点的距离）是 1.0，而每一个面是一个等边三角形。

8.2.3 GLUT 多面体程序示例

使用GLUT柏拉图式实体的函数，下列程序生成这些多面体经过变换的透视线框显示。所有 5 个实体位于同一个显示窗口内（参见图 8.1）。

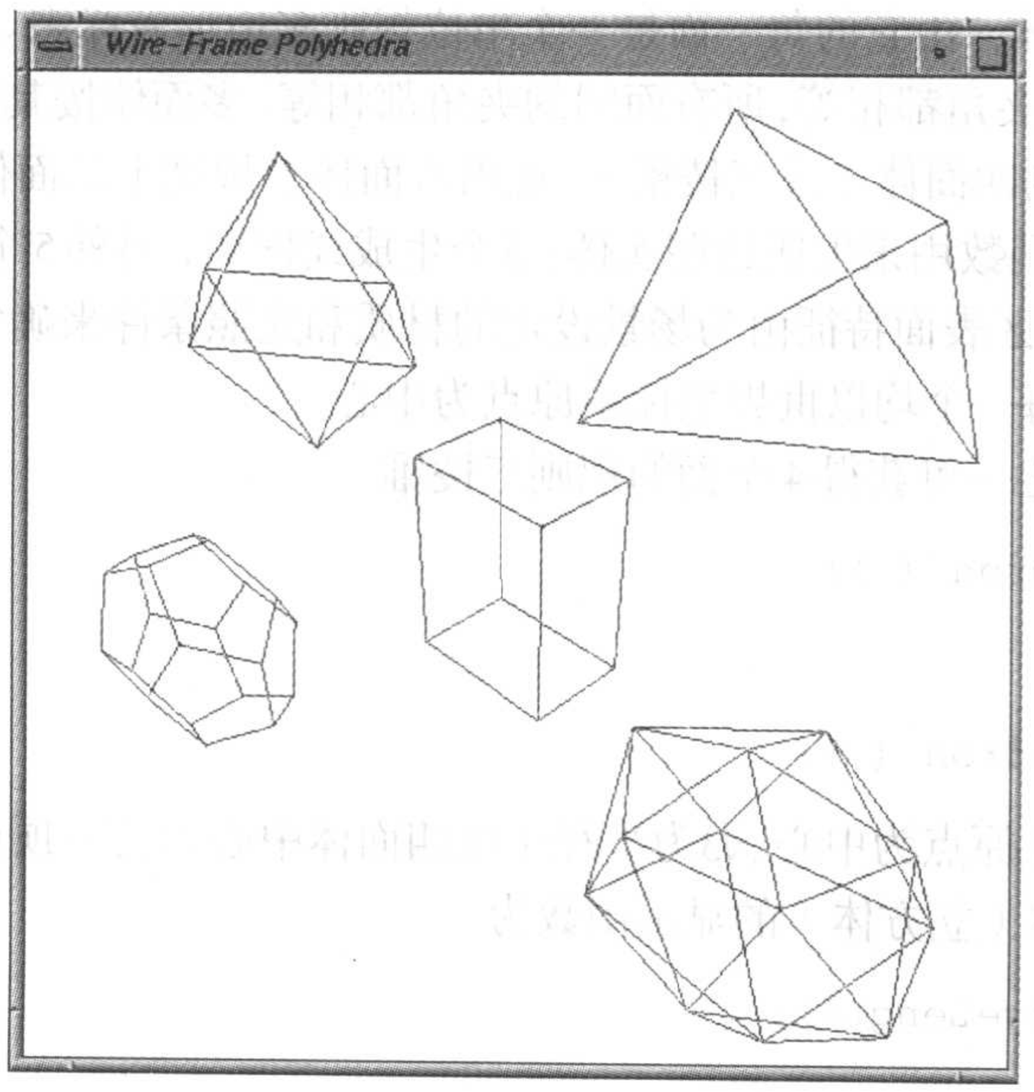


图 8.1 5 个 GLUT 多面体的透视图，由过程 displayWirePolyhedra 进行缩放并定位于一个显示窗口内

```
#include <GL/glut.h>

GLsizei winWidth = 500, winHeight = 500;  // Initial display-window size.

void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 0.0);  // White display window.
}

void displayWirePolyhedra (void)
{
    glClear (GL_COLOR_BUFFER_BIT);  // Clear display window.

    glColor3f (0.0, 0.0, 1.0);      // Set line color to blue.

    /*  Set viewing transformation.  */
    gluLookAt (5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    /*  Scale cube and display as wire-frame parallelepiped. */
    glScalef (1.5, 2.0, 1.0);
    glutWireCube (1.0);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    /*  Scale, translate, and display wire-frame dodecahedron. */
    glScalef (0.8, 0.5, 0.8);
    glTranslatef (-6.0, -5.0, 0.0);
    glutWireDodecahedron ( );

    /*  Translate and display wire-frame tetrahedron. */
    glTranslatef (8.6, 8.6, 2.0);
    glutWireTetrahedron ( );

    /*  Translate and display wire-frame octahedron. */
    glTranslatef (-3.0, -1.0, 0.0);
    glutWireOctahedron ( );

    /*  Scale, translate, and display wire-frame icosahedron. */
    glScalef (0.8, 0.8, 1.0);
    glTranslatef (4.3, -2.0, 0.5);
    glutWireIcosahedron ( );

    glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
    glViewport (0, 0, newWidth, newHeight);

    glMatrixMode (GL_PROJECTION);
    glFrustum (-1.0, 1.0, -1.0, 1.0, 2.0, 20.0);

    glMatrixMode (GL_MODELVIEW);

    glClear (GL_COLOR_BUFFER_BIT);
}

void main (int argc, char** argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (100, 100);
    glutInitWindowSize (winWidth, winHeight);
    glutCreateWindow ("Wire-Frame Polyhedra");

    init ( );
    glutDisplayFunc (displayWirePolyhedra);
    glutReshapeFunc (winReshapeFcn);

    glutMainLoop ( );
}
```

8.3　曲面

曲面边界对象的等式可用参数或非参数形式表示，附录 A 给出了参数和非参数表示的总结和有关对比。计算机图形应用中有用的各种对象有二次曲面、超二次曲面、多项式和指数函数及样条曲面。这些输入对象的描述一般都用多边形网格来近似表示其表面。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.4 二次曲面

二次曲面是一类常用的对象，这类表面使用二次方程进行描述。其中包括球面、椭球面、环面、抛物面和双曲面。二次曲面，尤其是球面和椭球面，是图形场景的基本元素，而图形软件包中也都有生成这些曲面的子程序。二次曲面也可以用有理样条表示来生成。

8.4.1 球面

在笛卡儿坐标系中，中心在原点，半径为 r 的球面定义为满足下列方程的点集(x, y, z)：

$$x^2 + y^2 + z^2 = r^2 \tag{8.1}$$

我们也可以使用参数形式来描述球面，即使用纬度和经度角（参见图 8.2）：

$$\begin{aligned} x &= r\cos\phi\cos\theta, \qquad -\pi/2 \leqslant \phi \leqslant \pi/2 \\ y &= r\cos\phi\sin\theta, \qquad -\pi \leqslant \theta \leqslant \pi \\ z &= r\sin\phi \end{aligned} \tag{8.2}$$

在等式（8.2）的参数表达式中，角度参数 θ 和 ϕ 的范围是对称的。另外，可以利用标准球面坐标来写出参数方程，这里的角度 ϕ 指定为余纬度（参见图 8.3）。这样，ϕ 的取值范围是 $0 \leqslant \phi \leqslant \pi$，$\theta$ 的取值范围是 $0 \leqslant \theta \leqslant 2\pi$。也可以使用取值范围在 0 和 1 之间的参数 u、v 来代替 ϕ、θ，即 $\phi = \pi u$、$\theta = 2\pi v$。

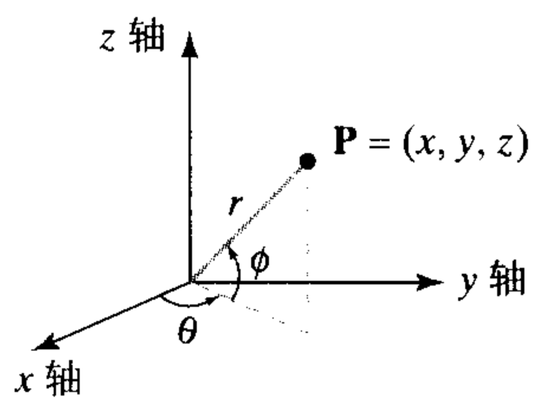


图 8.2 参数坐标位置(r, θ, ϕ)在半径为 r 的球面上

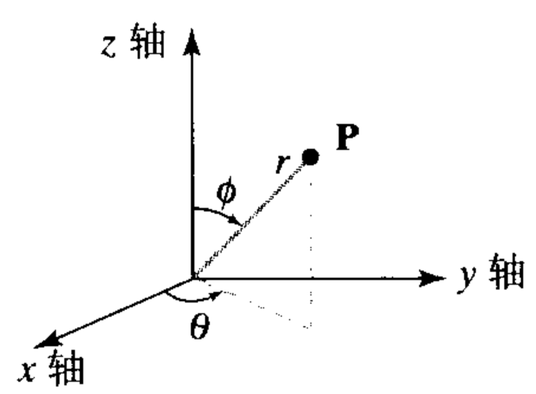


图 8.3 球面坐标参数(r, θ, ϕ)，利用了角 ϕ 的余纬度

8.4.2 椭球面

椭球面可以看成是球面的扩展，其中三条相互垂直的半径具有不同的值（参见图8.4）。中心在原点的椭球面笛卡儿表达式为

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1 \tag{8.3}$$

在图 8.2 中，使用纬度角 ϕ 和经度角 θ 所表示的参数方程为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$
\begin{aligned}
x &= r_x \cos\phi \cos\theta, \qquad -\pi/2 \leqslant \phi \leqslant \pi/2 \\
y &= r_y \cos\phi \sin\theta, \qquad -\pi \leqslant \theta \leqslant \pi \\
z &= r_z \sin\phi
\end{aligned}
\tag{8.4}
$$

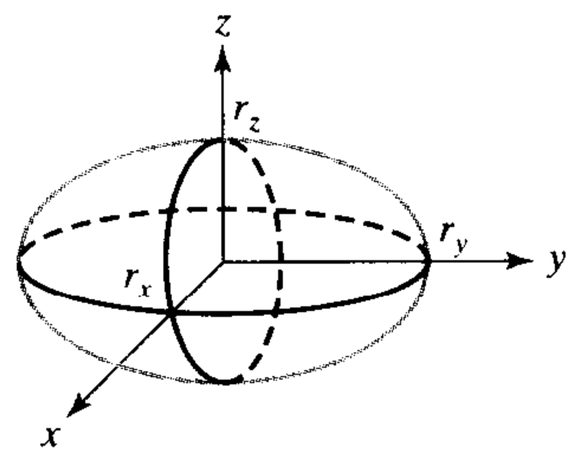


图 8.4　中心在原点、半径为 r_x、r_y、r_z 的椭球面

8.4.3　环面

汽车轮胎状的对象称为环面或锚状环。它常描述成将圆或椭圆绕该二次曲线之外的一个共面轴旋转而得的表面。环面的定义参数是该二次曲线中心到旋转轴之间的距离以及该二次曲线的尺寸。图 8.5 给出了 yz 平面上的半径为 r 的圆绕 z 轴旋转而得的环面。圆心在 y 轴上、环面轴向半径 r_{axial} 与从圆心沿 y 轴到 z 轴（旋转轴）的距离相等。而环面的剖面半径是生成圆的半径。

图 8.5 中侧视图给出的剖面方程是

$$(y - r_{\text{axial}})^2 + z^2 = r^2$$

将该圆绕 z 轴旋转生成该环面，其表面位置用下列笛卡儿方程表示：

$$\left(\sqrt{x^2 + y^2} - r_{\text{axial}}\right)^2 + z^2 = r^2 \tag{8.5}$$

有圆形剖面的环面的相应参数方程是

$$
\begin{aligned}
x &= (r_{\text{axial}} + r\cos\phi)\cos\theta, \qquad -\pi \leqslant \phi \leqslant \pi \\
y &= (r_{\text{axial}} + r\cos\phi)\sin\theta, \qquad -\pi \leqslant \theta \leqslant \pi \\
z &= r\sin\phi
\end{aligned}
\tag{8.6}
$$

我们也可以通过将一个椭圆代替圆绕 z 轴旋转来生成环面。对于一个在 yz 平面上、长轴和短轴为 r_y 和 r_z 的椭圆，我们可将该椭圆公式写为

$$\left(\frac{y - r_{\text{axial}}}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

其中 r_{axial} 是旋转 z 轴沿 y 轴到椭圆中心的距离。生成的环面可描述为下列笛卡儿方程：

$$\left(\frac{\sqrt{x^2 + y^2} - r_{\text{axial}}}{r_y}\right) + \left(\frac{z}{r_z}\right)^2 = 1 \tag{8.7}$$

剖面为椭圆的环面的相应参数表示为

$$
\begin{aligned}
x &= (r_{\text{axial}} + r_y\cos\phi)\cos\theta, \qquad -\pi \leqslant \phi \leqslant \pi \\
y &= (r_{\text{axial}} + r_y\cos\phi)\sin\theta, \qquad -\pi \leqslant \theta \leqslant \pi \\
z &= r_z\sin\phi
\end{aligned}
\tag{8.8}
$$

前面的环面方程可能还有其他变形。例如，可通过将一个圆或椭圆绕旋转轴沿一个椭圆路径旋转。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

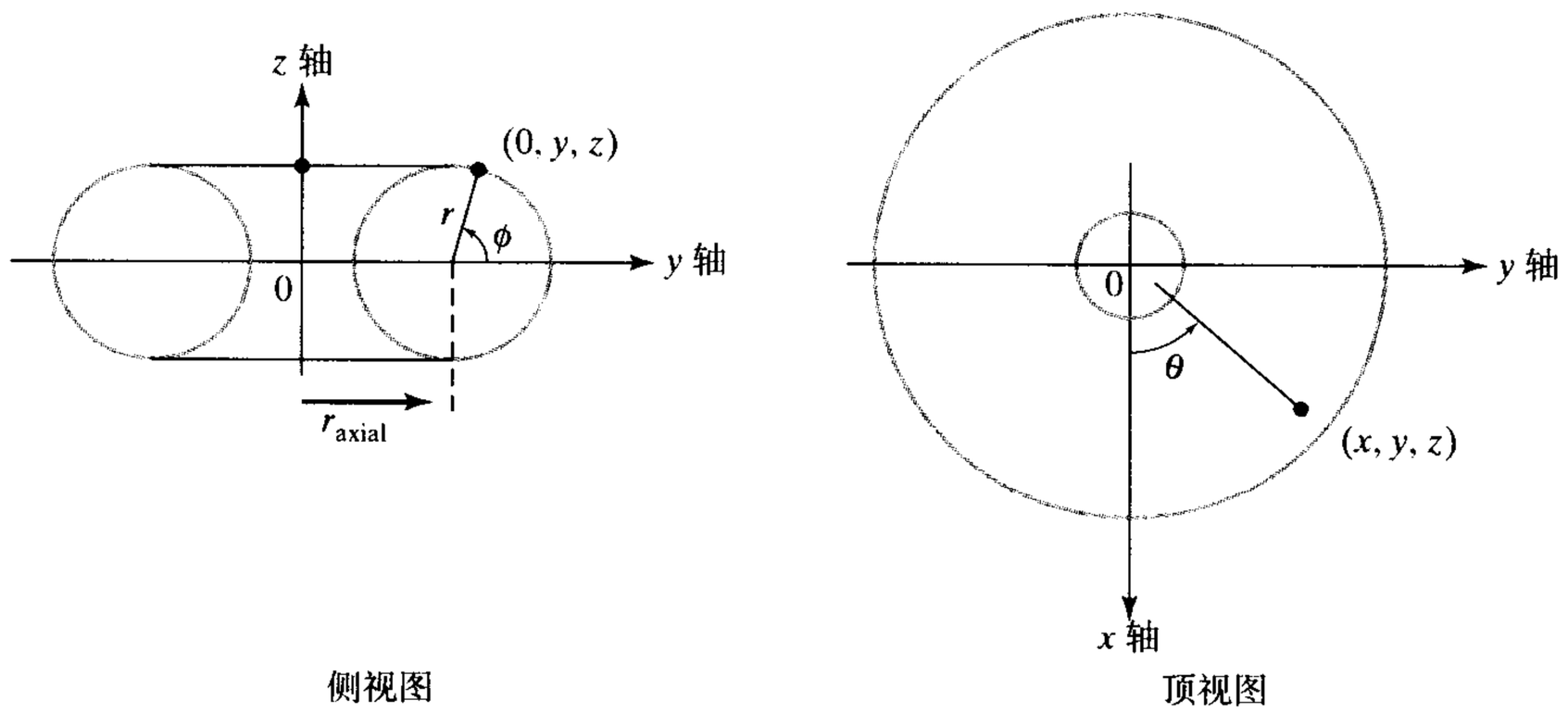


图 8.5　以坐标原点为中心、剖面为圆且环面轴为 z 轴的一个环面

8.5　超二次曲面

这类对象由二次曲面的表达式生成。**超二次曲面**（superquadrics）通过将额外参数插入二次方程而形成，这样便于调整对象的形状。对于曲线增加一个参数，而对于曲面则增加两个参数。

8.5.1　超椭圆

在相应的椭圆方程中，通过允许 x 和 y 项的指数为变量，可以得到超椭圆的笛卡儿表达式。笛卡儿超椭圆方程的表示形式之一是

$$\left(\frac{x}{r_x}\right)^{2/s}+\left(\frac{y}{r_y}\right)^{2/s}=1 \tag{8.9}$$

其中参数 s 是任何实数。当 $s=1$ 时，可以得到一般的椭圆表达式。

相对于方程（8.9），超椭圆参数方程可以表示成

$$\begin{aligned} x &= r_x \cos^s \theta, \qquad -\pi \leqslant \theta \leqslant \pi \\ y &= r_y \sin^s \theta \end{aligned} \tag{8.10}$$

图 8.6 给出了运用不同的参数 s 值而产生的超椭圆形状。

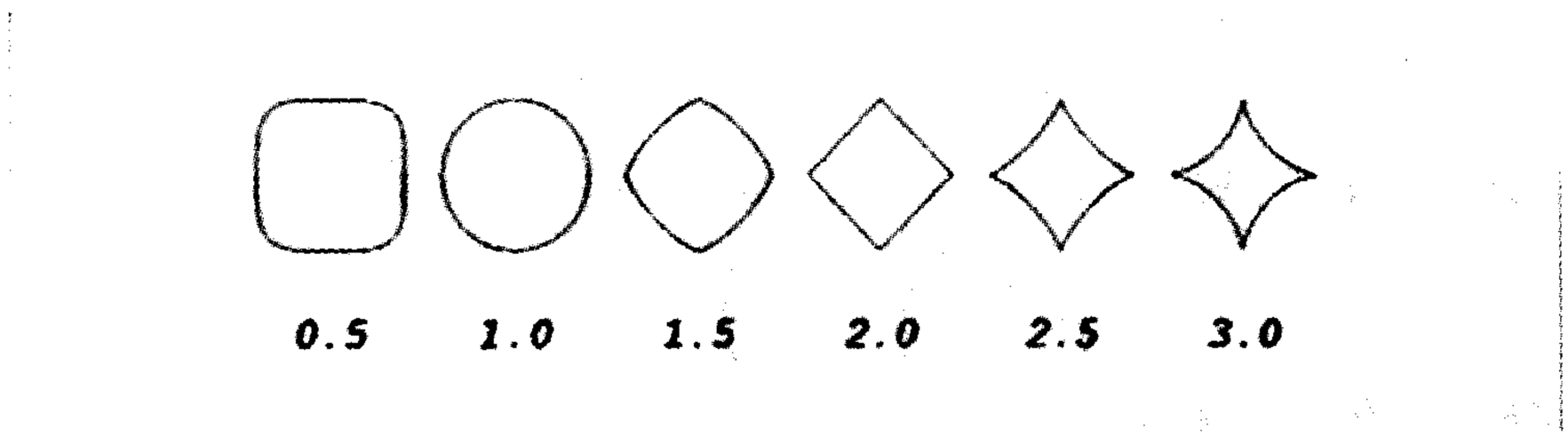


图 8.6　$r_x=r_y$、参数 s 值从 0.5 到 3.0 的超椭圆

8.5.2　超椭球面

超椭球面的笛卡儿表达式，是通过在椭球面方程中增加两个指数参数而得到的：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\left[\left(\frac{x}{r_x}\right)^{2/s_2}+\left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1}+\left(\frac{z}{r_z}\right)^{2/s_1}=1 \tag{8.11}$$

当 $s_1=s_2=1$ 时，得到一般的椭球面。

对于超椭球面的方程（8.11），可以将相应的参数表达式写成

$$\begin{aligned} x &= r_x\cos^{s_1}\phi\cos^{s_2}\theta, & -\pi/2\leqslant\phi\leqslant\pi/2\\ y &= r_y\cos^{s_1}\phi\sin^{s_2}\theta, & -\pi\leqslant\theta\leqslant\pi\\ z &= r_z\sin^{s_1}\phi \end{aligned} \tag{8.12}$$

图 8.7 给出了由不同参数 s_1 和 s_2 值生成的超椭球面形状。这些形状及其他超二次曲面形状的组合可以生成很复杂的形状，如家具、螺栓和其他金属构件。

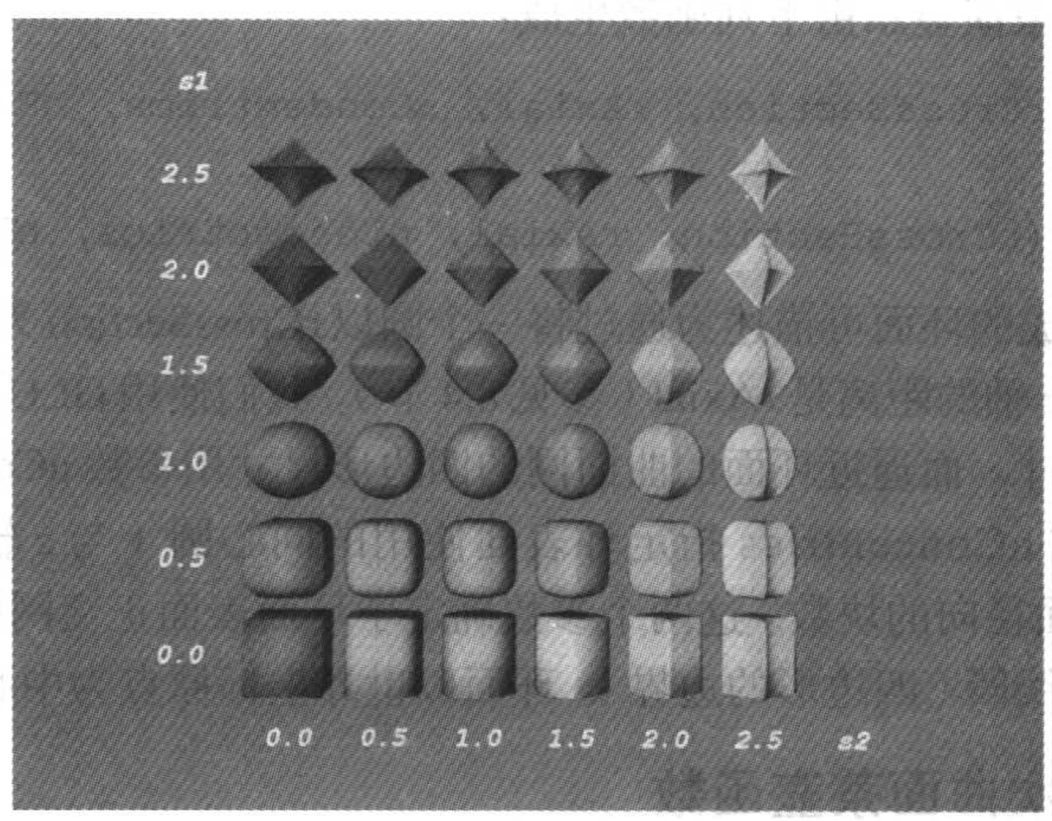


图 8.7　$r_x=r_y=r_z$、参数值 s_1 和 s_2 的范围从 0.0 到 2.5 的超椭球面图示

8.6　OpenGL 二次曲面和三次曲面函数

使用 OpenGL 实用工具包（GLUT）和实用库（GLU）中的函数可以显示一个球面和其他一些三维的二次曲面对象。另外，GLUT 有一个函数用来显示由双三次曲面片定义的茶壶形体。每一个 GLUT 函数都有两个版本且都很容易结合到应用程序中。一个版本用来显示线框曲面，另一个版本则把曲面显示成一组填充多边形面片。使用 GLUT 函数可显示球面、锥面、环面或茶壶。二次曲面的 GLU 函数较多用于设置，但提供了少量选择。使用 GLU 函数可显示球面、圆柱面、圆台面、圆锥、圆盘（或空心盘）和圆环（盘）的剖面。

8.6.1　GLUT 二次曲面函数

下列函数之一均可生成一个 GLUT 球面：

```
glutWireSphere (r, nLongitudes, nLatitudes);
```

或

```
glutSolidSphere (r, nLongitudes, nLatitudes);
```

该球面的半径由赋给参数 r 的双精度浮点数确定。参数 nLongitudes 和 nLatitudes 用来选择经、纬度数目，它们用来给出四边形网格以逼近球面。四边形面片的边是经、纬线的直线段逼近。该球面在模型坐标中定义，中心在世界坐标原点，极坐标轴位于 z 轴。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

使用下列函数之一可获得一个GLUT圆锥面：

```
glutWireCone (rBase, height, nLongitudes, nLatitudes);
```

或

```
glutSolidCone (rBase, height, nLongitudes, nLatitudes);
```

锥底半径设定为双精度浮点数而锥高分别用参数rbase和height。与GLUT球面一样，参数nLongitudes和nLatitudes赋以整数值，用来指定四边形网格逼近的正交面线的数目。圆锥经线是从顶到底、经过圆锥轴的一条直线段。每条纬线用一组沿圆锥表面、平行于锥底、位于与圆锥轴线垂直的平面上的一个圆周的直线段显示。该圆锥在模型坐标中定义，底的中心在世界坐标原点且圆锥轴线位于世界系 z 轴。

环面的线框或表面浓淡显示由下列函数生成：

```
glutWireTorus (rCrossSection, rAxial, nCondentrics, nRadialSlices);
```

或

```
glutSolidTorus (rCrossSection, rAxial, nCondentrics, nRadialSlices);
```

使用这些GLUT函数生成的环面可描述为通过将一个圆以rCrossSection为半径、绕共面 z 轴旋转而得的表面，其中圆心到 z 轴的距离是rAxial（参见8.4节）。我们使用GLUT函数中的双精度浮点数半径值来指定环面的尺寸。而逼近环面的曲面网的四边形尺寸用参数nConcentrics和nRadialSlices中的整数来设定。参数nConcentrics指定环表面使用的同心圆（与 z 轴中心）的数量，而参数nRadialSlices指定环表面径向的块数。这两个参数确定了环面表面正交网格线数量，网格交点之间的每一网格线显示为直线段（四边形的边界）。环面以世界坐标原点为中心显示，其轴落在 z 轴上。

8.6.2 OpenGL三次曲面茶壶函数

在研制计算机算法的早期，人们建立了若干个三维对象的多边形网格数据表，用于绘制技术的测试。这些对象包括一辆大众（Volkswagen）汽车和一把茶壶，由犹他（Utah）大学开发。在1975年由Martin Newell构造的Utah茶壶包含306个顶点，定义了32个双三次Bézier曲面片（参见8.11节）。因为确定一个复杂对象的表面坐标是十分耗时的，所以这些数据集，尤其是茶壶表面网格得到广泛的应用。

我们可以使用下列两个GLUT函数之一用一千多个双三次曲面片来显示茶壶。

```
glutWireTeapot (size);
```

或

```
glutSolidTeapot (size);
```

该茶壶表面用OpenGL的Bézier曲线函数（参见8.11节）生成。参数size设定为双精度浮点数、用做茶壶球状体的最大半径。该茶壶的中心在世界坐标原点、其垂直轴与 y 轴重合。

8.6.3 GLU二次曲面函数

要使用GLU生成一个二次曲面，必须先为该二次曲面赋一个名字，激活GLU二次曲面绘制器，并指定曲面参数的值。另外，我们可以设定其他参数值来控制GLU二次曲面的外貌。

下列语句给出了显示一个以世界坐标原点为中心的球面线框图的基本调用序列。

```
GLUquadricObj *sphere1;
sphere1 = gluNewQuadric ( );
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
gluQuadricDrawStyle (sphere1, GLU_LINE);
gluSphere (sphere1, r, nLongitudes, nLatitudes);
```

第一个语句定义二次曲面对象的名字，在本例中选择的名字是sphere1。这个名字在其他GLU函数中用来指向这一特定的二次曲面。接着，gluNewQuadric 函数激活二次曲面绘制器，然后gluQuadricDrawStyle命令为sphere1选定GLU_LINE模式。这样，以每对顶点之间显示一条直线段将球面按线框图形式显示。参数r赋以双精度浮点数用做球面半径，而球面按相等的经、纬度划分成一组多边形。参数nLongitudes和nLatitudes赋以整数来指定经、纬线的数量。

GLU二次曲面还有另外三个模式。在gluQuadricDrawStyle中使用符号常量GLU_POINT，则以绘点方式显示二次曲面。对于该球面来说，显示了经线和纬线的交点。另一个选择是符号常量GLU_SILHOUETTE。它生成的线框图中不包括共平面多边形面片的公共边。而使用符号常量GLU_FILL，则按浓淡填充区方式显示多边形面片。

使用相同的基本命令序列可生成其他GLU二次曲面图元。要生成一个圆锥面、圆柱面或圆台面的视图，我们可用下列函数代替gluSphere函数：

```
gluCylinder (quadricName, rBase, rTop, height, nLongitudes, nLatitudes);
```

该对象的底部在xy平面上（$z=0$），且其轴在z轴上。我们对参数rBase赋以双精度值表示该二次曲面底部的半径，而使用参数rTop表示该二次曲面顶部半径。如果rTop = 0.0，我们得到一个圆锥面；如果rTop = rBase，则得到圆柱面。否则，显示一个圆台面。参数height赋以双精度值表示高度，而参数nLongitudes和nLatitudes赋以整数值以确定将曲面均匀划分的垂直和水平线的数量。

如果要在xy平面（$z=0$）显示一个以世界坐标原点为中心的、平的圆环或实心盘，可使用

```
gluDisk (ringName, rInner, rOuter, nRadii, nRings);
```

我们用参数rInner和rOuter设定双精度的内部半径和外部半径值。如果rInner = 0，该盘是实心的。否则，在盘中心显示一个同心的洞。该盘表面用整数参数nRadii和nRings划分成一组面片，这两个参数分别指定放射状的网格片数和同心的环数。环的方向相对于z轴定义，环的前面朝$+z$轴方向而环的后面朝$-z$轴方向。

使用下列函数可指定圆环面的一个部分：

```
gluPartialDisk (ringName, rInner, rOuter, nRadii, nRings,
                startAngle, sweepAngle);
```

双精度的参数startAngle以度为单位指定xy平面上从正y轴顺时针旋转的角度。类似地，参数sweepAngle给出从startAngle位置以度为单位的角度。因此，一个圆盘部分的显示角度从startAngle位置到startAngle + sweepAngle位置。例如，如果startAngle = 0.0且sweepAngle = 90.0，则显示位于xy平面第一象限的盘部分。

使用下列函数可以回收分配给任一GLU二次曲面的存储器并消除该曲面：

```
gluDeleteQuadric (quadricName);
```

我们也可以使用下列函数定义任何二次曲面的前/后方向：

```
gluQuadricOrientation (quadricName, normalVectorDirection);
```

参数normalVectorDirection赋以GLU_OUTSIDE或GLU_INSIDE，从而指出曲面法向量方向，其中“outside”指出**前向面方向**（front-face direction）而“inside”指出**后向面方向**（back-face direction）。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

默认值是GLU_OUTSIDE。对于该圆盘，默认的前向面方向是正向 z 轴（盘的上方）。另一选择是表面法向量的生成。

```
gluQuadricNormals (quadricName, generationMode);
```

参数generationMode赋以符号常量，指出表面法向量如何生成。默认值是GLU_NONE，表示不生成表面法向量，而一般情况下不对二次曲面应用光照条件。对平表面的浓淡（每一个面一种均匀颜色），使用符号常量GLU_FLAT。这为每一多边形面片生成一个表面法向量。当提供其他光照条件时，使用常量GLU_SMOOTH，为每一顶点位置生成一个法向量。

GLU二次曲面的其他选择包括设定表面纹理参数。我们可以指定一个在生成二次曲面过程中出错时引入的函数：

```
gluQuadricCallback (quadricName, GLU_ERROR, function);
```

8.6.4 使用GLUT和GLU二次曲面函数的程序示例

下面的例子程序显示三个面对象（球面、圆锥面和圆柱面）的线框图。我们用 z 轴设定观察向上向量，因此所有显示对象的轴都向上。这三个对象位于同一显示窗口的不同位置，如图8.8所示。

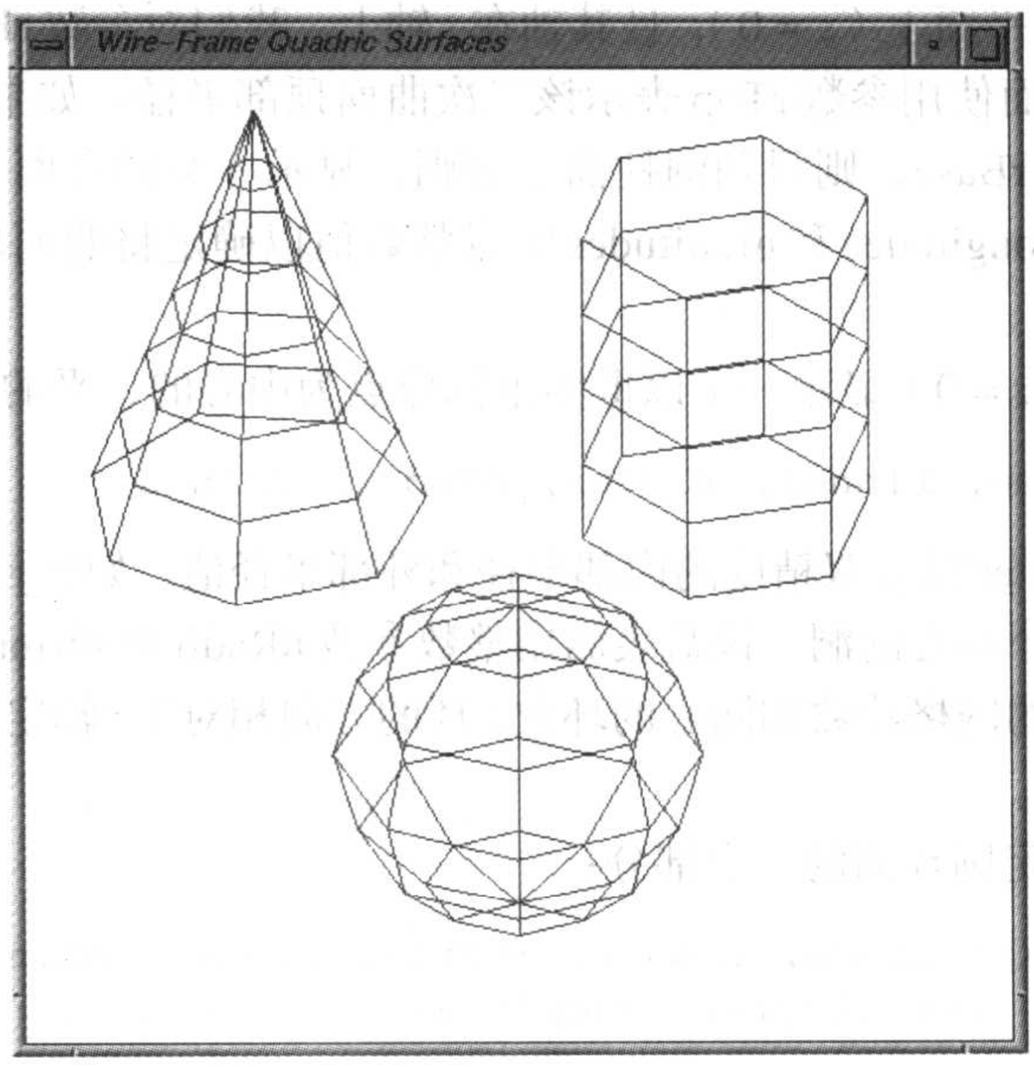


图8.8 由过程wireQuadSurfs定位在同一显示窗口的一个GLUT球面、一个GLUT圆锥面和一个GLU圆柱面的显示

```
#include <GL/glut.h>

GLsizei winWidth = 500, winHeight = 500; // Initial display-window size.

void init (void)
{
   glClearColor (1.0, 1.0, 1.0, 0.0);   // Set display-window color.
}

void wireQuadSurfs (void)
{
   glClear (GL_COLOR_BUFFER_BIT);       // Clear display window.
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glColor3f (0.0, 0.0, 1.0); // Set line-color to blue.

/* Set viewing parameters with world z axis as view-up direction. */
gluLookAt (2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);

/* Position and display GLUT wire-frame sphere. */
glPushMatrix ( );
glTranslatef (1.0, 1.0, 0.0);
glutWireSphere (0.75, 8, 6);
glPopMatrix ( );

/* Position and display GLUT wire-frame cone. */
glPushMatrix ( );
glTranslatef (1.0, -0.5, 0.5);
glutWireCone (0.7, 2.0, 7, 6);
glPopMatrix ( );

/* Position and display GLU wire-frame cylinder. */
GLUquadricObj *cylinder; // Set name for GLU quadric object.
glPushMatrix ( );
glTranslatef (0.0, 1.2, 0.8);
cylinder = gluNewQuadric ( );
gluQuadricDrawStyle (cylinder, GLU_LINE);
gluCylinder (cylinder, 0.6, 0.6, 1.5, 6, 4);
glPopMatrix ( );

glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   glViewport (0, 0, newWidth, newHeight);

   glMatrixMode (GL_PROJECTION);
   glOrtho (-2.0, 2.0, -2.0, 2.0, 0.0, 5.0);

   glMatrixMode (GL_MODELVIEW);

   glClear (GL_COLOR_BUFFER_BIT);
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (100, 100);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Wire-Frame Quadric Surfaces");

   init ( );
   glutDisplayFunc (wireQuadSurfs);
   glutReshapeFunc (winReshapeFcn);

   glutMainLoop ( );
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.7 柔性对象

在计算机应用中开发了对非刚体对象建模的多种技术，8.26节将讨论显示布和橡胶等材料特征的方法。但是其他对象，如分子结构、液体和水滴、熔化的对象及动物和人的肌肉形状表现为一定程度的变化性。这些对象在一定的运动或在接近其他对象时会改变它们的表面形状，且它们的曲面表面很难用常规形状表示。这一类对象一般称为**柔性对象**（blobby object）。

例如，分子形状在独立存在时可以使用球形进行描述，但当分子接近其他分子时会改变其本身形状。这是因为在另一分子出现时电子云的形状会变形，因此两个分子之间发生了“结合”现象。图8.9表示了当两个分子分离时分子形状的延伸、分裂和收缩效果。这些特性不能简单地使用球形或椭圆形进行描述。同样，图8.10展示了具有类似特征的人体上臂的肌肉形状。

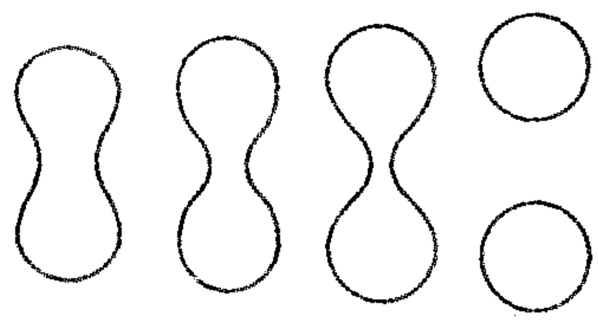


图8.9 分子粘结。当两个分子从结合到分离时，表面形状延伸、分裂，最后收缩成圆形

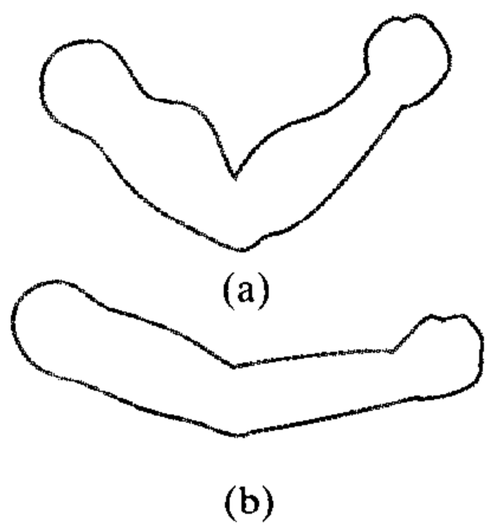


图8.10 人体上臂的“柔性”肌肉形状

已经开发了几种使用分布函数来表示柔性对象的建模方法。它们一般都把对象的体积描述成在任何运动或交互中保持不变。其中一个方法是使用高斯密度函数或高斯凸起方法（参见图8.11）的混合对对象进行建模。这里的曲面函数定义为

$$f(x, y, z) = \sum_k b_k e^{-a_k r_k^2} - T = 0 \tag{8.13}$$

其中，$r_k^2 = x_k^2 + y_k^2 + z_k^2$，参数$T$是某个特定的临界值，参数$a_k$和$b_k$用来调整单个对象的柔性程度。参数$b_k$取负值能产生“凹”形而不是“凸”形。图8.12表示了由四个高斯密度函数建模的混合对象的表面结构。在临界值级别上，数值化求根技术可以定位坐标重叠值。此时，单个对象的剖面按照圆或椭圆进行建模。如果两个剖面彼此靠近，则它们趋于一种柔性形状，如图8.9所示，其中的结构依赖于两个对象分离的程度。

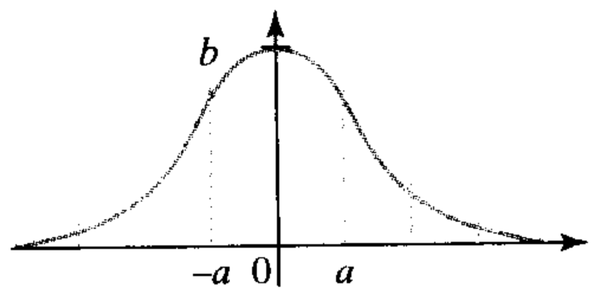


图8.11 三维高斯凸形，其中心在位置0、高度为b、标准方差为a

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

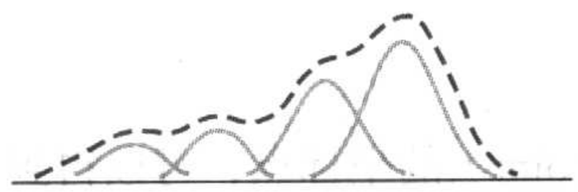


图 8.12　由四个高斯凸形复合而成的柔性对象

其他用于生成柔性对象的方法是，使用在几个区间内取 0 而不是指数形式的密度函数。**元球**（meta-ball）模型将混合对象看成是几个二次密度函数的复合形式：

$$f(r)=\begin{cases} b\left(1-\dfrac{3r^2}{d^2}\right), & 0<r\leqslant\dfrac{d}{3} \\ \dfrac{3}{2}b\left(1-\dfrac{r}{d}\right)^2, & \dfrac{d}{3}<r\leqslant d \\ 0, & r>d \end{cases} \tag{8.14}$$

而**软对象**（soft object）模型使用函数：

$$f(r)=\begin{cases} 1-\dfrac{22r^2}{9d^2}+\dfrac{17r^4}{9d^4}-\dfrac{4r^6}{9d^6}, & 0<r\leqslant d \\ 0, & r>d \end{cases} \tag{8.15}$$

现在，有一些设计和绘图程序软件包提供柔性函数建模，从而处理那些不能只用多边形或样条函数进行模拟的应用。图 8.13 给出了使用元球的柔性对象模拟程序的用户界面。

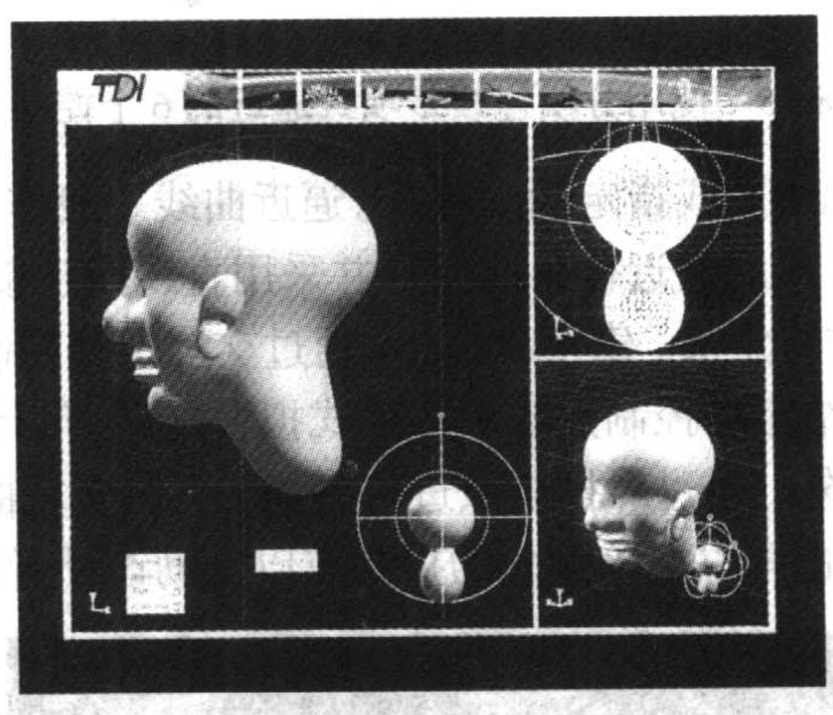


图 8.13　使用元球对对象建模的Blob Modeler和Blob Animator动画软件中的屏幕布局（由 Thomson Digital Image 提供）

8.8　样条表示

在绘图术语中，样条是通过一组指定点集而生成平滑曲线的柔性带。当绘制曲线时，几个较小的加权沿着样条的长度分配并固定在绘图表上的相应位置。术语——**样条曲线**（spline curve）原指使用这种方式绘制的曲线。数学上使用分段三次多项式函数来描述这种曲线，其中各曲线段的连接处有连续的一次和二次导数。在计算机图形学中，样条曲线指由多项式曲线段连接而成的曲线，在每段的边界处满足特定的连续性条件。**样条曲面**（spline surface）可以使用两组正交样条曲线进行描述。在图形学应用中使用几种不同的样条描述。每种描述简单地表示一个带有某种特定的边界条件的多项式的特殊类型。

样条用于设计曲线和曲面形状，将绘制的图形数字化以及指定场景中对象的动画路径或照相机位置。样条的典型CAD应用包括：汽车车身设计、飞机和航天飞机表面的设计、船体设计以及家庭应用。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.8.1 插值和逼近样条

给定一组称为**控制点**（control point）的坐标点，可以得到一条样条曲线，这些点给出了曲线的大致形状。根据这些坐标位置，可以使用以下的两种方法之一选取分段连续参数多项式函数。当选取的多项式使得曲线通过每个控制点时，如图 8.14 所示，则所得曲线称为这组控制点的**插值**（interpolate）样条曲线。另一种情况是，当选取的多项式使部分或全部控制点都不在生成的曲线上，所得曲线称为这组控制点的**逼近**（approximate）样条曲线（参见图 8.15）。构造插值和逼近样条曲面的方法类似。

图 8.14 使用分段连续多项式插值的 6 个控制点

图 8.15 使用分段连续多项式逼近的 6 个控制点

插值曲线通常用于数字化绘图或指定动画路径。逼近曲线一般作为设计工具来构造对象形体。图 8.16 给出了在设计应用中生成的逼近样条曲面，并采用直线段连接曲面上的控制点。

一条样条曲线由控制点进行定义、建模和控制。通过交互选择控制点的空间位置，设计者可以建立一条初始曲线。在对一组给定的控制点显示多项式拟合之后，设计者可以重定位部分或全部控制点，以重建曲线的形状。另外，通过对控制点进行变换，可以实现曲线对象的变换（平移、旋转或缩放）。另外，CAD 软件包也可以插入额外的控制点以帮助设计者调整曲线形状。

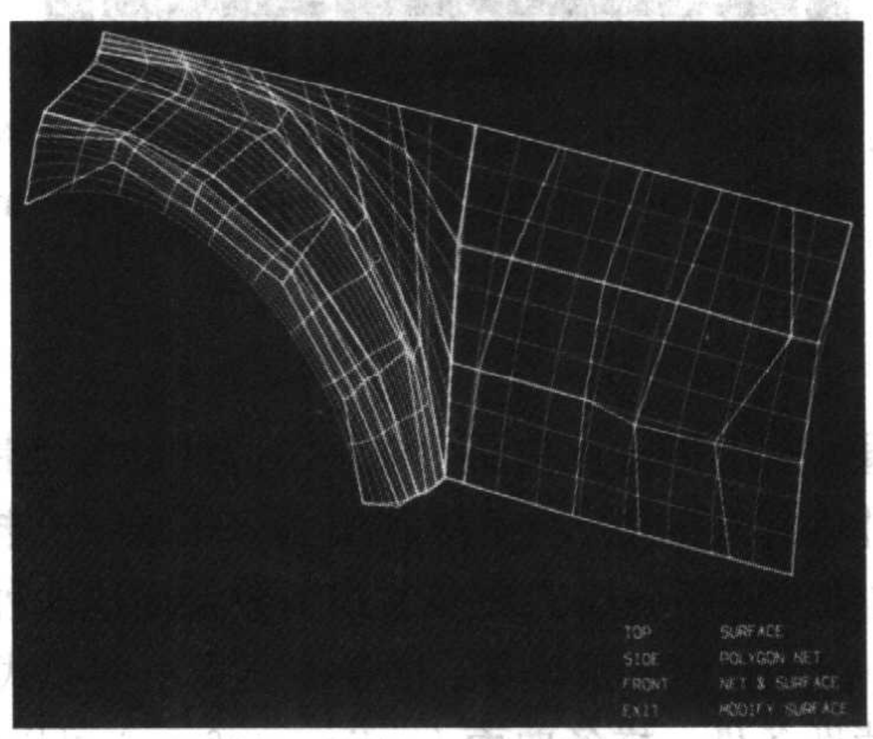


图 8.16 汽车设计 CAD 应用中的一个逼近样条曲面。曲面网格线用多项式曲线段绘制，而控制点用直线段连接（Evans & Sutherland 提供）

包含一组控制点的凸多边形边界称为**凸壳**（convex hull）。可以将一个二维曲线的凸壳形状设想成一个围绕控制点位置的拉紧的橡皮筋，使得每个控制点或者在凸壳的边界上或者在凸壳内

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

（参见图 8.17）。因此，一个二维样条曲线的凸壳是一个凸多边形。凸壳提供了曲线或曲面与围绕控制点区域间的偏差的测量。多数情况下，样条以凸壳为界，这样就保证了对象形态平滑地而不是不稳定地摆动着沿控制点前进。凸壳也给出了所设计曲线或曲面的坐标范围，因而它在裁剪和观察程序中十分有用。

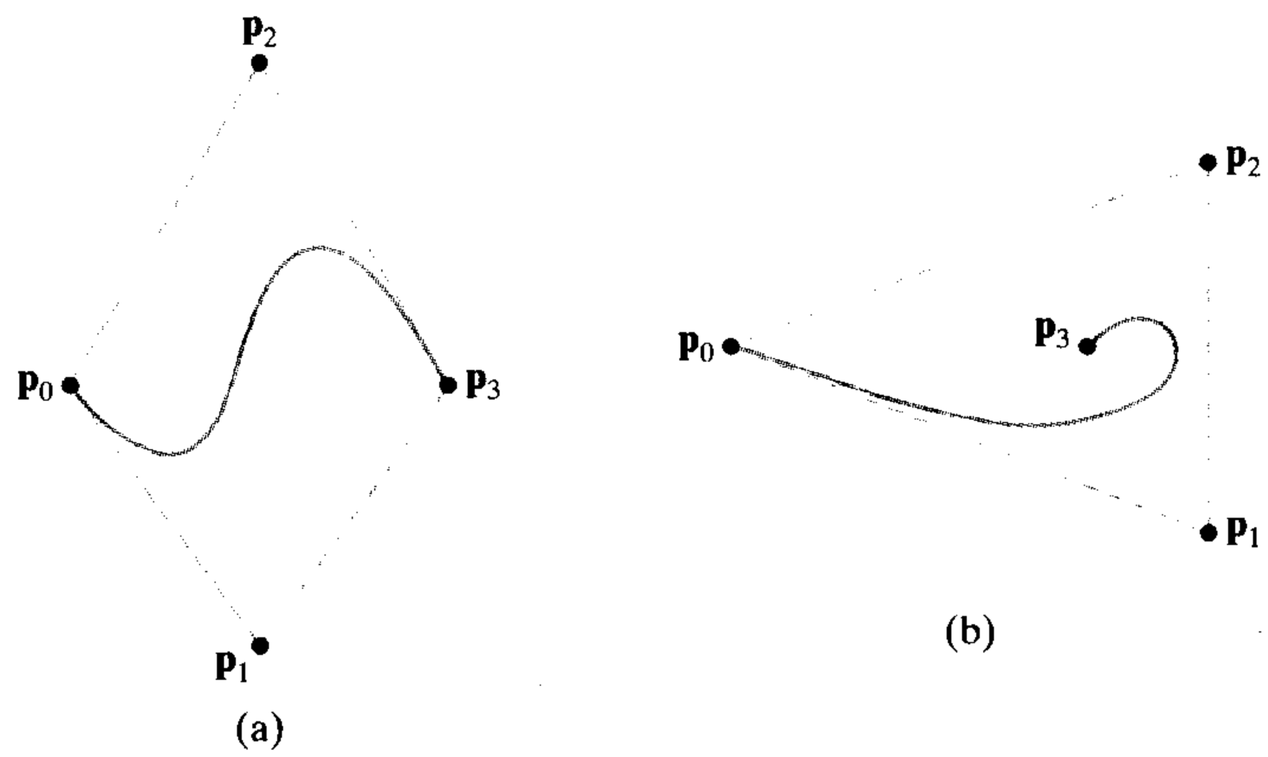


图 8.17　xy 平面上两组控制点的凸壳形状（虚线）

对于逼近样条，连接控制点序列的折线通常会显示出来，以提醒设计者控制点的顺序。这一组连接线段通常称为**曲线的控制图**（control graph）。该控制图还可以称为“控制多边形”或“特征多边形”，在控制图是一条折线而不是一个多边形时也是如此。图 8.18 显示了图 8.17 中控制点序列的控制图形状。对于一个样条曲面，两组由控制点连接成的折线为曲面控制图形成一个二次曲面网的多边形面片，如图 8.16 所示。

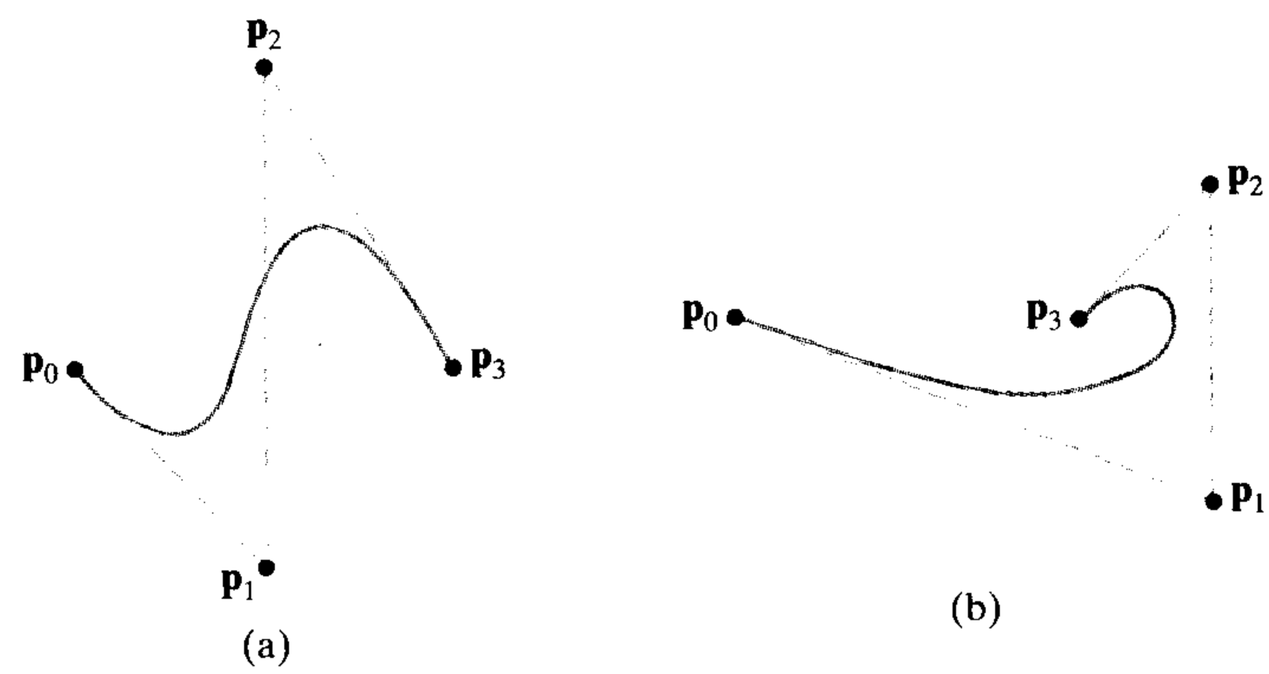


图 8.18　两个不同控制点组形成的控制图形状

8.8.2　参数连续性条件

为了保证分段参数曲线从一段到另一段平滑过渡，可以在连接点处要求各种**连续性条件**（continuity condition）。如样条的每一部分以参数坐标函数形式进行描述：

$$x = x(u), \qquad y = y(u), \qquad z = z(u), \qquad u_1 \leqslant u \leqslant u_2 \tag{8.16}$$

则通过在曲线段的公共部分匹配连接的参数导数，从而建立**参数连续性**（parametric continuity）。

0 阶参数连续性（zero-order parametric continuity），记做 C^0 连续，可以简单地表示曲线相连。即第一个曲线段在 u_2 处的 x、y、z 值与第二个曲线段在 u_1 处的 x、y、z 值相等。**一阶参数连续性**（first-

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

order parametric continuity），记做C^1连续性，说明代表两个相邻曲线段的方程在相交点处有相同的一阶导数（切线）。**二阶参数连续性**（second-order parametric continuity），记做C^2连续性，是指两个曲线段在交点处有相同的一阶和二阶导数。高阶参数连续性可以类似定义。图8.19展示了C^0、C^1和C^2连续性的例子。

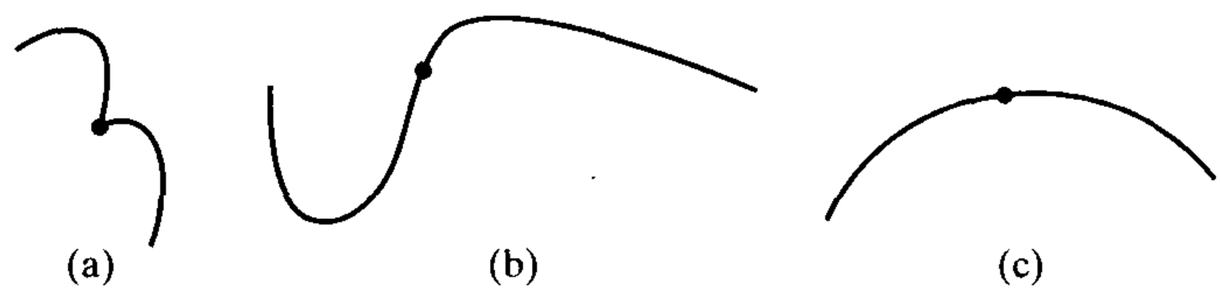


图 8.19　曲线的分段构造，由具有不同参数连续性的两个曲线段组成：(a)0 阶连续性；(b)一阶连续性；(c)二阶连续性

对于二阶连续性，交点处的切向量变化率相等。这样，切线从一个曲线段平滑地变化到另一个曲线段（参见图8.19(c)）。但对于一阶连续性，两曲线段的切向量变化率可能会不同（参见图8.19(b)），因此两个相邻曲线段的总体形状会有突变。一阶连续性已经能够满足数字化绘图及一些设计应用，而二阶连续性对摄影机移动的动画路径和很多精密CAD需求有一定作用。沿图8.19(b)中的曲线路径，以参数u中的相同步长移动的镜头会在两个曲线段的交汇处经历一个突变，移动过程中将产生不连续性。但如果镜头沿图8.19(c)中的路径移动，在相交处会平稳地运动。

8.8.3　几何连续性条件

连接两个相邻曲线段的另一种方法是指定**几何连续性**（geometric continuity）条件。这种情况下，只要求两条曲线段在相交处的参数导数成比例，而不是必须相等。

0 阶几何连续性（zero-order geometric continuity），记为G^0连续性，与0阶参数连续性相同；即两个曲线段必在公共点处有相同的坐标位置。**一阶几何连续性**（first-order geometric continuity），记为G^1连续性，表示一阶导数在两个相邻曲线段的交点处成比例。若曲线上的参数位置记为$\mathbf{P}(u)$、切向量为$\mathbf{P}'(u)$，但在G^1连续性下，相邻曲线段在交点处的切向量大小不一定相等。**二阶几何连续性**（second-order geometric continuity），记为G^2连续性，表示两个曲线段在相交处的一阶和二阶导数均成比例。在G^2连续性下，两个曲线段在交点处的曲率相等。

生成带有几何连续性条件的曲线与生成带有参数连续性条件的曲线有一些类似，但二者的曲线形状有些差别。图8.20是几何连续性与参数连续性的比较。对于几何连续性，曲线将向具有较大切向量的部分弯曲。

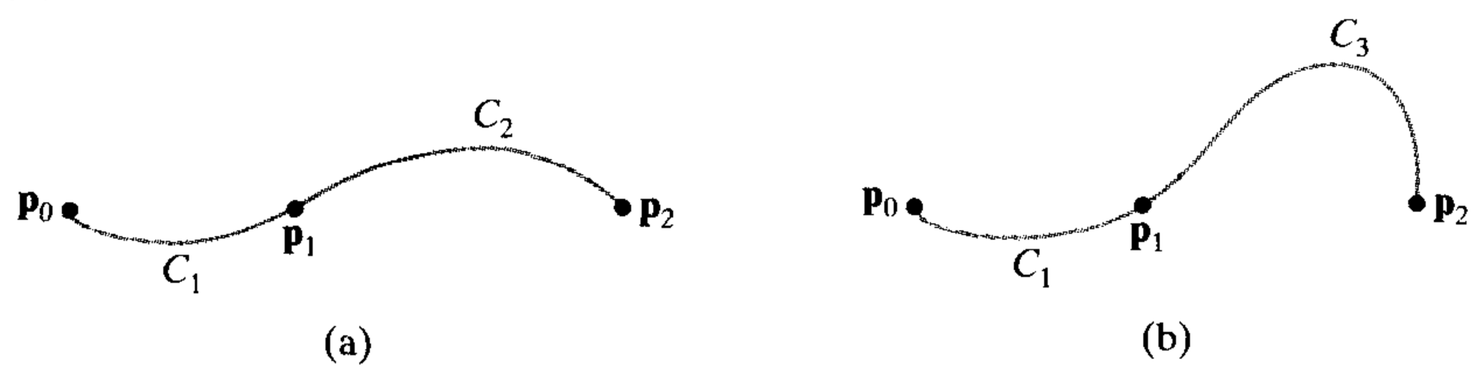


图 8.20　三个控制点拟合成两条曲线段并带有(a)参数连续性和(b)几何连续性。其中曲线C_3上$\mathbf{P}_1$点处的切向量值比曲线C_1上$\mathbf{P}_1$点处的切向量值大

8.8.4　样条描述

给定多项式的阶和控制点位置后，指定一个具体的样条表达式有三个等价方法：(1)列出一组加在样条上的边界条件，(2)列出描述样条特征的行列式，(3)列出一组混合函数或基函数，确定如何组合指定的曲线几何约束，以计算曲线路径上的位置。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

为了说明这三个等价描述，假设沿样条路径有下列关于 x 坐标的三次参数多项式表达式：

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x, \qquad 0 \leqslant u \leqslant 1 \tag{8.17}$$

该曲线的边界条件可以设为端点坐标 $x(0)$ 和 $x(1)$ 以及端点处的一次导数 $x'(0)$ 和 $x'(1)$。这四个边界条件是确定四个系数 a_x、b_x、c_x 和 d_x 值的充分条件。

从边界条件中，通过将等式（8.17）重写为矩形乘积形式，可以得到描述这个样条曲线的矩阵：

$$\begin{aligned} x(u) &= [u^3 \quad u^2 \quad u \quad 1] \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix} \\ &= \mathbf{U} \cdot \mathbf{C} \end{aligned} \tag{8.18}$$

其中，$\mathbf{U}$ 是参数 u 的幂次行矩阵，$\mathbf{C}$ 是系数列矩阵。运用等式（8.18），就可以写出矩阵形式的边界条件，并求得系数矩阵 $\mathbf{C}$：

$$\mathbf{C} = \mathbf{M}_{\text{spline}} \cdot \mathbf{M}_{\text{geom}} \tag{8.19}$$

其中，$\mathbf{M}_{\text{geom}}$ 是包含样条上的几何约束值（边界条件）的四元素列矩阵，$\mathbf{M}_{\text{spline}}$ 是 4×4 矩阵，该矩阵将几何约束值转化成多项式系数，并且提供了样条曲线的特征。矩阵 $\mathbf{M}_{\text{geom}}$ 包含了控制点的坐标值和其他已经指定的几何约束。这样，我们可以使用矩阵表示来替代等式（8.18）中的 $\mathbf{C}$，从而得到

$$x(u) = \mathbf{U} \cdot \mathbf{M}_{\text{spline}} \cdot \mathbf{M}_{\text{geom}} \tag{8.20}$$

矩阵 $\mathbf{M}_{\text{spline}}$ 描述了一个样条表示。有时称该矩阵为基本矩阵（basis matrix），对从一个样条表达式转换到另一个样条表达式特别有用。

最后，通过扩展等式（8.20），可以得到关于坐标 x 的几何约束参数的多项式表示：

$$x(u) = \sum_{k=0}^{3} g_k \cdot \mathrm{BF}_k(u) \tag{8.21}$$

其中，g_k 是约束参数，类似控制点坐标和控制点处的曲线斜率，多项式 $\mathrm{BF}_k(u)(k = 0, 1, 2, 3)$ 是混合函数或基本函数。在以后的章节中，我们将讨论一些计算机图形应用中常用的样条曲线和曲面的特点，包括它们的矩阵和混合函数描述。

8.8.5　样条曲面

定义一个样条曲面通过使用某个空间区域中的一个控制点网指定两组正交的样条曲线来实现。如果我们用 $\mathbf{p}_{k_u,k_v}$ 表示控制点的位置，则样条曲面上的任意一点可用样条曲线混合函数的积来计算：

$$\mathbf{P}(u, v) = \sum_{k_u,k_v} \mathbf{p}_{k_u,k_v} \mathrm{BF}_{k_u}(u) \mathrm{BF}_{k_v}(v) \tag{8.22}$$

曲面参数 u 和 v 在 0 到 1 的范围内变化，但这个范围依赖于所使用的样条曲线类型。一种指定三维控制点位置的方法是在地平面二维网格位置选择高度值。

8.8.6　修剪样条曲面

在 CAD 应用中，曲面设计可能需要一些不能仅通过简单地调整控制点位置来实现的功能。例如，样条曲面的一部分可能需要剪断来拟合两个设计面片，或需要有一个洞让一个管道穿过。对于

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

这些应用，图形软件包常提供生成**修剪曲线**（trimming curves）的函数，从而在样条曲线上挖去这些部分，如图 8.21 所示。修剪曲线在参数的 *uv* 曲面坐标上定义，并且必须是封闭曲线。

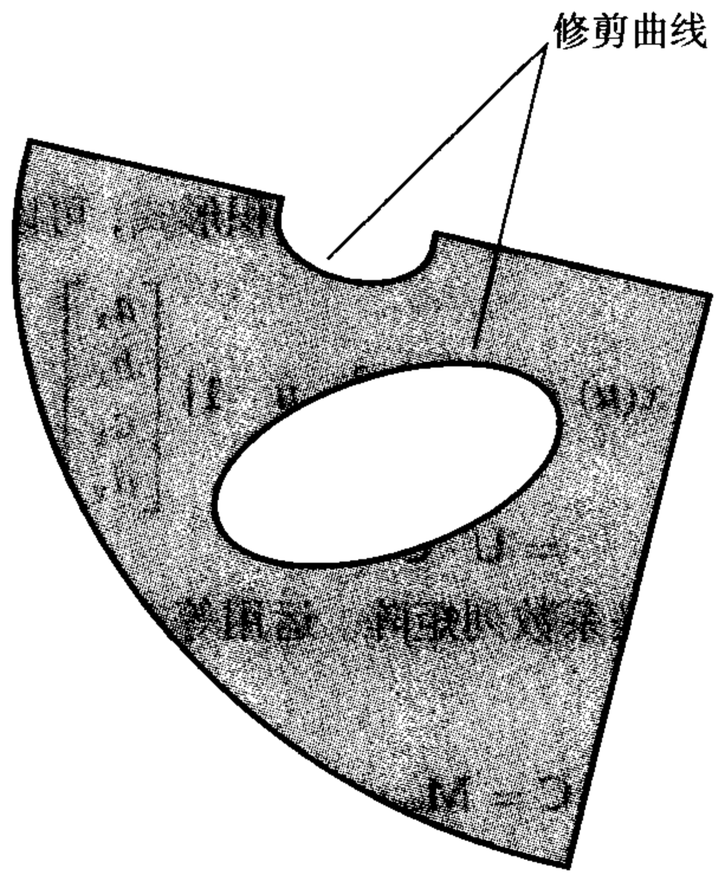


图 8.21　使用修剪曲线修改曲面的一部分

8.9　三次样条插值方法

此类样条大多用于建立对象运动路径或提供实体表示和绘画，但插值样条有时也用来设计物体形状。三次多项式在灵活性和计算速度之间提供了一个合理的折中方案。与更高次多项式相比，三次样条只需较少的计算和存储空间，并且较为稳定。与低次多项式相比，三次样条在模拟任意曲线形状时显得更加灵活。

给出一组控制点，通过拟合这些输入点生成分段三次多项式曲线，可以得到经过每一控制点的三次插值样条。假设有 $n+1$ 个控制点，其坐标分别为

$$\mathbf{p}_k = (x_k, y_k, z_k) \quad k = 0, 1, 2, \ldots, n$$

这些点的三次插值拟合在图 8.22 中给出。我们可以使用下列方程组来描述拟合每对控制点的参数三次多项式：

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y, \qquad (0 \leqslant u \leqslant 1) \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \tag{8.23}$$

对于这三个方程中的每一个，必须确定出多项式表示中的四个系数 a、b、c、d 值，而 $n+1$ 个控制点共产生 n 条曲线段，每一段都需要确定系数值。通过在两条曲线段的“交点”处设置足够的边界条件来解决上述问题，可以得到所有的系数值。在后面的内容中，将探讨用于对三次样条插值建立边界条件的常用方法。

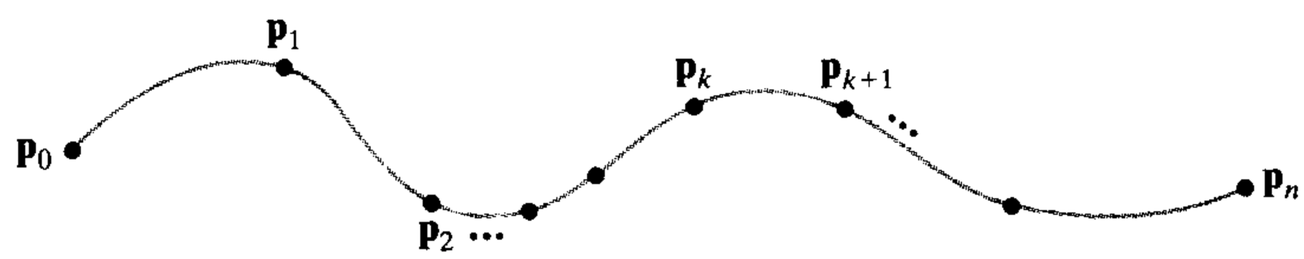


图 8.22　n+1 个控制点的分段连续三次样条插值

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.9.1　自然三次样条

首先用于图形应用的样条曲线之一是**自然三次样条**（natural cubic spline）。这个插值曲线是原始绘图样条的一个数学表达式。在使用公式表示一个自然三次样条时，需要两个相邻曲线段在公共边界处有相同的一阶和二阶导数，即自然三次样条具有 C^2 连续性。

如图 8.22 所示，需要拟合 $n+1$ 个控制点，这样共有 n 条曲线段和 $4n$ 个多项式系数需要确定。对于每一个内部控制点（共 $n-1$ 个）有四个边界条件：在该控制点两侧的两个曲线段在该点处有相同的一阶和二阶导数，并且两条曲线段都要通过该点。这就给出了由 $4n$ 个多项式系数组成的 $4n-4$ 个方程。另外再给出从第一个控制点 $\mathbf{p}_0$（即曲线起点）所得的方程以及从另一个控制点 $\mathbf{p}_n$（即曲线终点）所得的条件。我们还需要两个条件才能解出所有的系数值。获取这两个额外方程的方法之一是在 $\mathbf{p}_0$ 和 $\mathbf{p}_n$ 处设二阶导数为 0。另一个方法是增加两个“隐含”控制点，分别位于控制点序列的两端，即增加控制点 $\mathbf{p}_{-1}$ 和 $\mathbf{p}_{n+1}$。这时，所有的原有控制点都成为了内部点，并且具有了所需的 $4n$ 个边界条件。

尽管自然三次样条对于绘图样条是一个数学模型，但其有一个主要缺点。如果控制点中的任意一个发生了变动，则整条曲线都将受到影响。这样，自然三次样条不允许“局部控制”，因此不给出完整的新控制点集，则不可能构造曲线的一部分。因此，人们开发了其他一些三次样条插值表示。

8.9.2　Hermite 插值

Hermite 样条（以法国数学家 Charles Hermite 命名）是一个分段三次多项式，并在每个控制点具有给定的切线。与自然三次样条不同，Hermite 样条可以局部调整，因为每个曲线段仅依赖于端点约束。

如图 8.23 所示，如果在控制点 $\mathbf{p}_k$ 和 $\mathbf{p}_{k+1}$ 之间的曲线段是参数三次函数 $\mathbf{P}(u)$，那么 Hermite 曲线段的边界条件是

$$\begin{aligned}\mathbf{P}(0) &= \mathbf{p}_k\\ \mathbf{P}(1) &= \mathbf{p}_{k+1}\\ \mathbf{P}'(0) &= \mathbf{D}\mathbf{p}_k\\ \mathbf{P}'(1) &= \mathbf{D}\mathbf{p}_{k+1}\end{aligned} \tag{8.24}$$

其中 $\mathbf{Dp}_k$ 和 $\mathbf{Dp}_{k+1}$ 指定了控制点 $\mathbf{p}_k$ 和 $\mathbf{p}_{k+1}$ 处的参数导数值（曲线的斜率）。

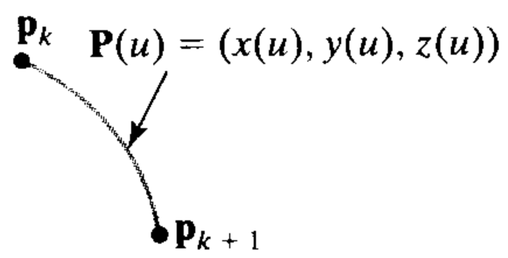


图 8.23　在控制点 $\mathbf{p}_k$ 和 $\mathbf{p}_{k+1}$ 间的 Hermite 曲线段的参数点函数 $\mathbf{P}(u)$

对于这个 Hermite 曲线段，可以写出与方程（8.23）等价的向量方程：

$$\mathbf{P}(u) = \mathbf{a}u^3 + \mathbf{b}u^2 + \mathbf{c}u + \mathbf{d}, \quad 0 \leqslant u \leqslant 1 \tag{8.25}$$

其中，$\mathbf{P}(u)$的分量 x 是 $x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$，分量 y 和 z 也类似。与方程（8.25）等价的矩阵方程是

$$\mathbf{P}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \begin{bmatrix}\mathbf{a}\\ \mathbf{b}\\ \mathbf{c}\\ \mathbf{d}\end{bmatrix} \tag{8.26}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

函数的导数可以表示为

$$\mathbf{P}'(u) = [3u^2 \quad 2u \quad 1 \quad 0] \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{8.27}$$

以 0 和 1 代替以上两个方程中的 u，可以把 Hermite 边界条件（8.24）表示为矩阵形式：

$$\begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{8.28}$$

该方程对多项式系数求解，于是有

$$\begin{aligned} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} \\ &= \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} \\ &= \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} \end{aligned} \tag{8.29}$$

其中，Hermite 矩阵 $\mathbf{M}_H$ 是边界约束矩阵的逆矩阵。使用边界条件，方程（8.26）可以写成

$$\mathbf{P}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{Dp}_k \\ \mathbf{Dp}_{k+1} \end{bmatrix} \tag{8.30}$$

计算方程（8.30）中的矩阵乘积，并且获得满足边界约束的系数，可以得到多项式形式。最后，得到的 Hermite 混合函数表达式为

$$\begin{aligned} \mathbf{P}(u) &= \mathbf{p}_k(2u^3 - 3u^2 + 1) + \mathbf{p}_{k+1}(-2u^3 + 3u^2) + \mathbf{Dp}_k(u^3 - 2u^2 + u) \\ &\quad + \mathbf{Dp}_{k+1}(u^3 - u^2) \\ &= \mathbf{p}_k H_0(u) + \mathbf{p}_{k+1} H_1 + \mathbf{Dp}_k H_2 + \mathbf{Dp}_{k+1} H_3 \end{aligned} \tag{8.31}$$

多项式 $H_k(u)$（$k = 0, 1, 2, 3$）称为混合函数，因为其混合了边界约束值（终点坐标和斜率）以得到曲线上每个坐标点的位置。图 8.24 给出了四个 Hermite 混合函数的形状。

Hermite 多项式对于某些数字化应用十分有用，可以较容易地得出或估算出曲线的斜率。但对于计算机图形学中的大部分问题，除了控制点坐标以外，更好的处理方法是，不需要输入曲线斜率值或其他几何信息就能生成样条曲线。cardinal 样条和 Kochanek-Bartels 样条这两种基于 Hermite 样

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

条的变化形式，我们将在下两节中讨论。这两种样条就不需要输入控制点上的曲线导数值。这些样条的程序是通过控制点的坐标位置来计算参数导数。

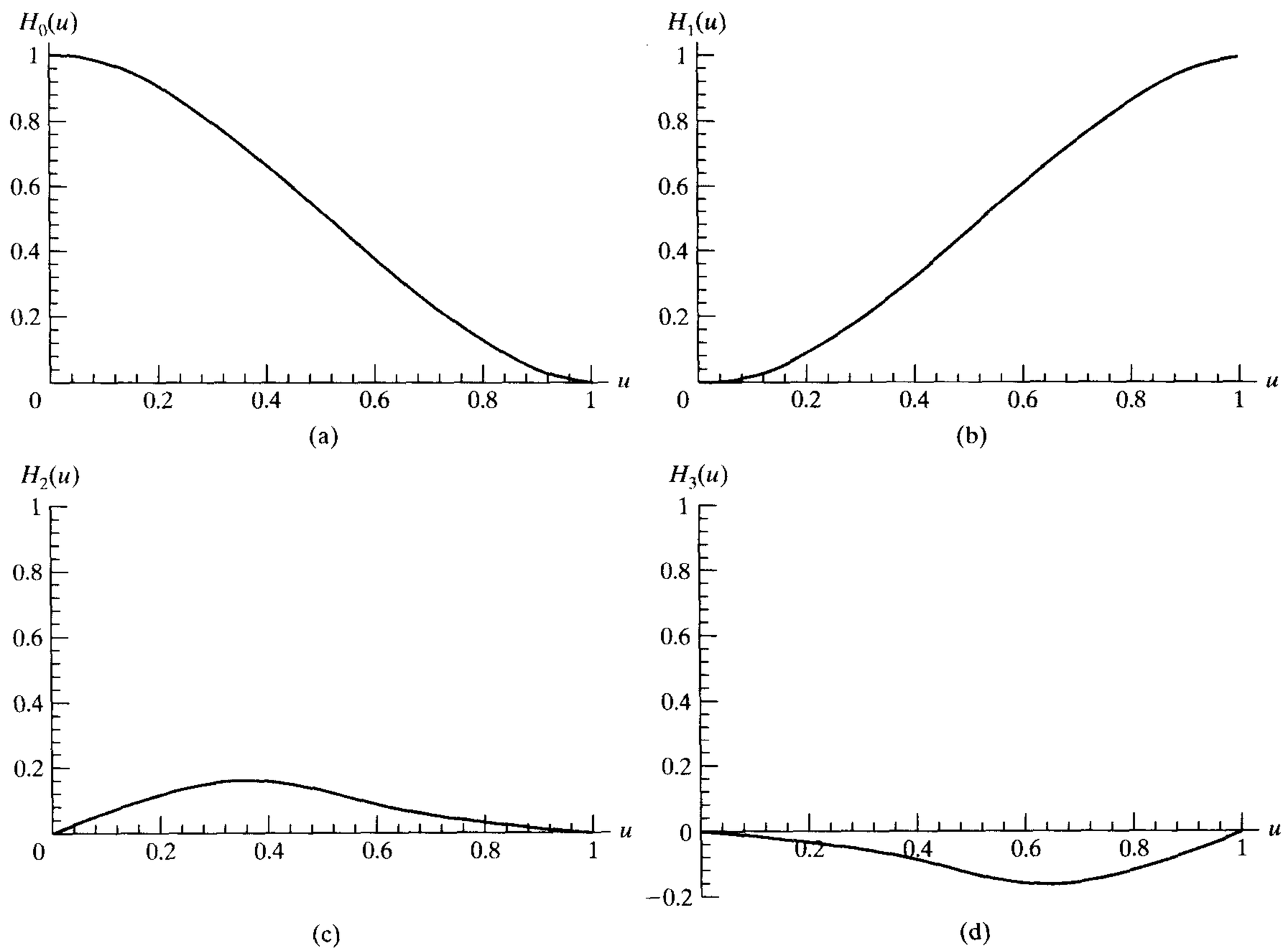


图 8.24　Hermite 混合函数

8.9.3　cardinal 样条

类似于 Hermite 样条，**cardinal 样条**也是插值分段三次曲线，并且每条曲线段的端点位置均指定切线。与 Hermite 样条不同，cardinal 样条不一定要给出端点的切线值。在 cardinal 的样条中，一个控制点的斜率值可以由两个相邻控制点的坐标进行计算。

一个 cardinal 样条完全由四个连续控制点给出。中间两个控制点是曲线段端点，其他两个点用于计算端点斜率。如图 8.25 所示，设 $\mathbf{P}(u)$ 是两控制点 $\mathbf{p}_k$ 和 $\mathbf{p}_{k+1}$ 间的参数三次函数式，则从 $\mathbf{p}_{k-1}$ 到 $\mathbf{p}_{k+1}$ 间的四个控制点用于建立 cardinal 样条段的边界条件：

$$
\begin{aligned}
\mathbf{P}(0) &= \mathbf{p}_k \\
\mathbf{P}(1) &= \mathbf{p}_{k+1} \\
\mathbf{P}'(0) &= \frac{1}{2}(1-t)(\mathbf{p}_{k+1}-\mathbf{p}_{k-1}) \\
\mathbf{P}'(1) &= \frac{1}{2}(1-t)(\mathbf{p}_{k+2}-\mathbf{p}_k)
\end{aligned}
\tag{8.32}
$$

控制点 $\mathbf{p}_k$ 和 $\mathbf{p}_{k+1}$ 处的斜率分别与弦 $\overline{\mathbf{p}_{k-1}\mathbf{p}_{k+1}}$ 和 $\overline{\mathbf{p}_k\mathbf{p}_{k+2}}$ 成正比（参见图 8.26）。参数 t 称为**张量**（tension）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

参数，因为t控制cardinal样条与输入控制点间的松紧程度。图8.27说明了张量t取很小和很大值时cardinal曲线的形状。当$t=0$时，这类曲线称为Catmull-Rom样条或Overhauser样条。

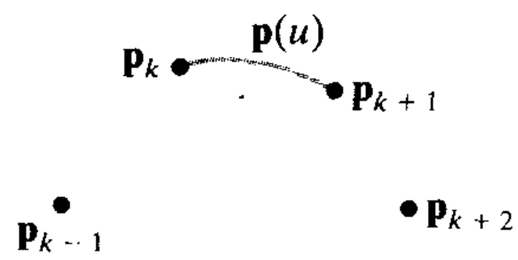


图8.25 在控制点$\mathbf{p}_k$和$\mathbf{p}_{k+1}$间cardinal样条段的参数向量函数$\mathbf{p}(u)$

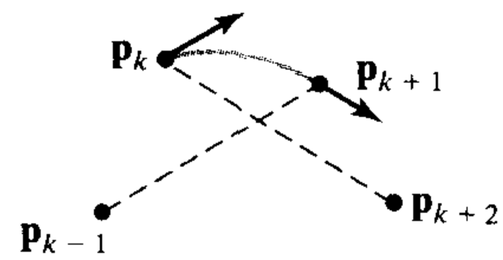


图8.26 cardinal样条段端点处的切向量正比于由相邻控制点所形成的弦（划线）

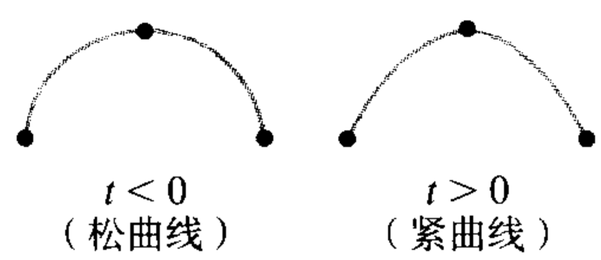


图8.27 张量参数在cardinal样条段形状中起到的作用

利用类似Hermite样条中的方法，我们可以将边界条件（8.32）转换成矩阵形式：

$$\mathbf{P}(u)=[u^3 \quad u^2 \quad u \quad 1]\cdot\mathbf{M}_C\cdot\begin{bmatrix}\mathbf{p}_{k-1}\\ \mathbf{p}_k\\ \mathbf{p}_{k+1}\\ \mathbf{p}_{k+2}\end{bmatrix} \tag{8.33}$$

其中cardinal矩阵是

$$\mathbf{M}_C=\begin{bmatrix}-s & 2-s & s-2 & s\\ 2s & s-3 & 3-2s & -s\\ -s & 0 & s & 0\\ 0 & 1 & 0 & 0\end{bmatrix} \tag{8.34}$$

这里$s=(1-t)/2$。

将矩阵方程（8.33）展开成多项式形式，可以有

$$\begin{aligned}\mathbf{P}(u)&=\mathbf{p}_{k-1}(-s\,u^3+2s\,u^2-s\,u)+\mathbf{p}_k[(2-s)u^3+(s-3)u^2+1]\\&\quad+\mathbf{p}_{k+1}[(s-2)u^3+(3-2s)u^2+s\,u]+\mathbf{p}_{k+2}(s\,u^3-s\,u^2)\\&=\mathbf{p}_{k-1}\,\mathrm{CAR}_0(u)+\mathbf{p}_k\,\mathrm{CAR}_1(u)+\mathbf{p}_{k+1}\,\mathrm{CAR}_2(u)+\mathbf{p}_{k+2}\,\mathrm{CAR}_3(u)\end{aligned} \tag{8.35}$$

这里，多项式$\mathrm{CAR}_k(u)(k=0,1,2,3)$是cardinal混合函数。图8.28给出了当$t=0$时，cardinal样条的基本函数的图。

图8.29、图8.30和图8.31给出了使用cardinal样条混合函数生成的曲线示例。在图8.29中，四段cardinal样条形成一条封闭曲线。第一段曲线用控制点集$\{\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$来生成，第二段曲线用控制点集$\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_0\}$来生成，第三段曲线的控制点集是$\{\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_0, \mathbf{p}_1\}$，最后一段曲线的控制点集是$\{\mathbf{p}_3, \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2\}$。在图8.30中，通过使用一个cardinal样条段并设定第二个与第三个控制点相同来获得一条封闭曲面。在图8.31中，通过设定第三个控制点接近第二个控制点来生成一条自相交cardinal样条曲线段。造成自相交的原因是在端点$\mathbf{p}_1$和$\mathbf{p}_2$的曲线斜率约束。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

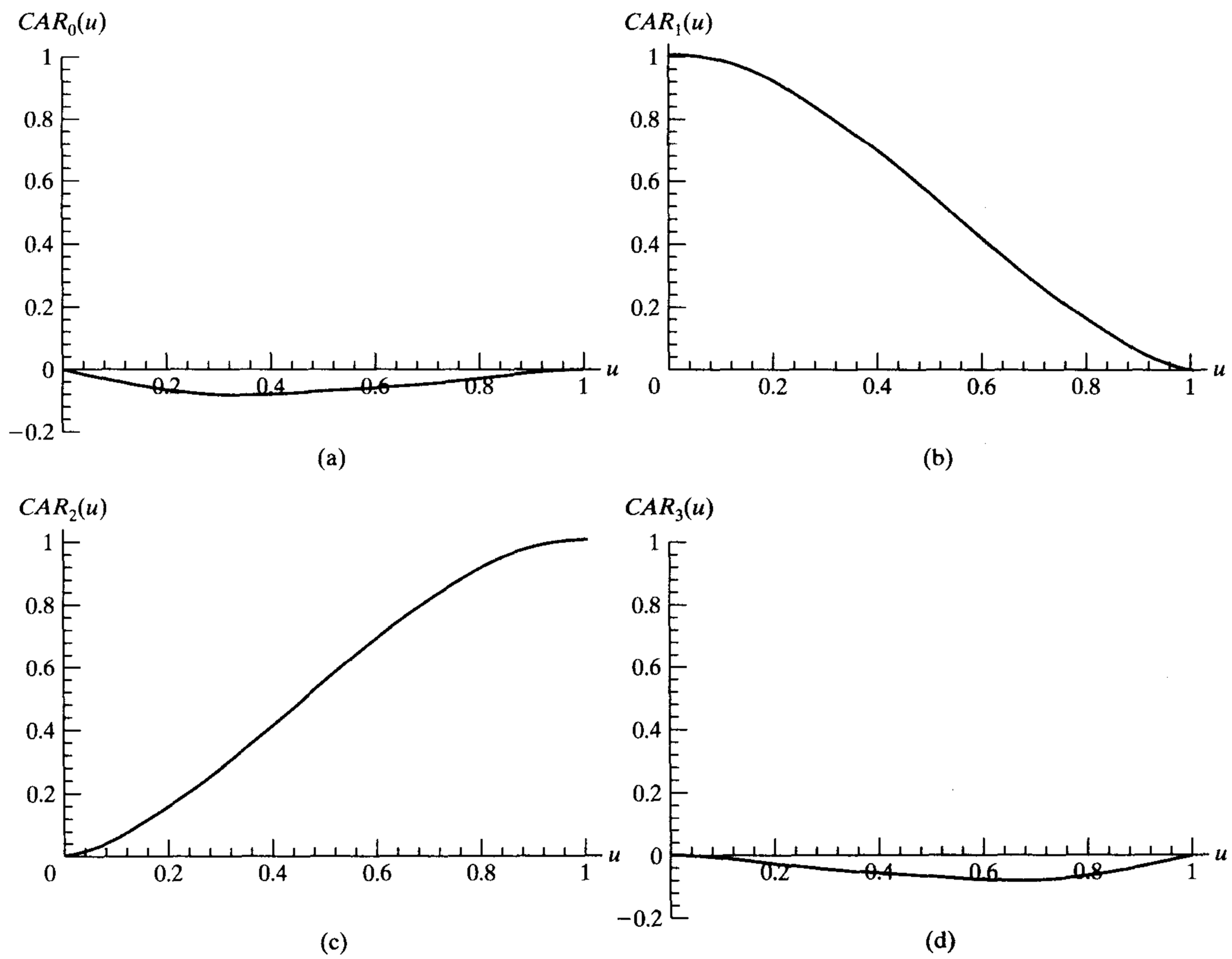


图 8.28　$t = 0$ 和 $s = 0.5$ 时的 cardinal 混合函数

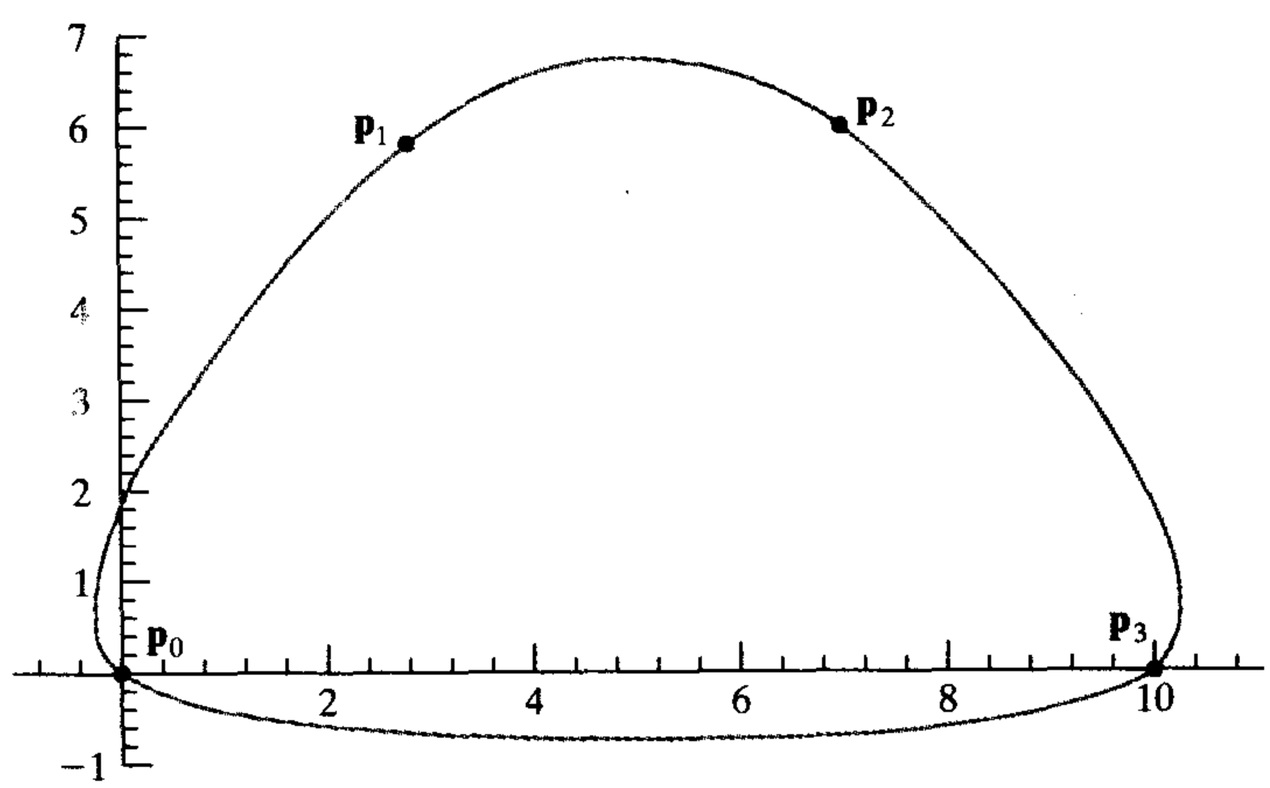


图 8.29　循环地改变控制点并取张量参数 $t = 0$ 所得的四段 cardinal 样条曲线组成的封闭曲线

8.9.4　Kochanek-Bartels 样条

这些插值三次多项式是 cardinal 样条的扩展。将两个附加参数引入到约束方程中，可以得到 Kochanek-Bartels 样条，从而为调整曲线段形状提供更多方便。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

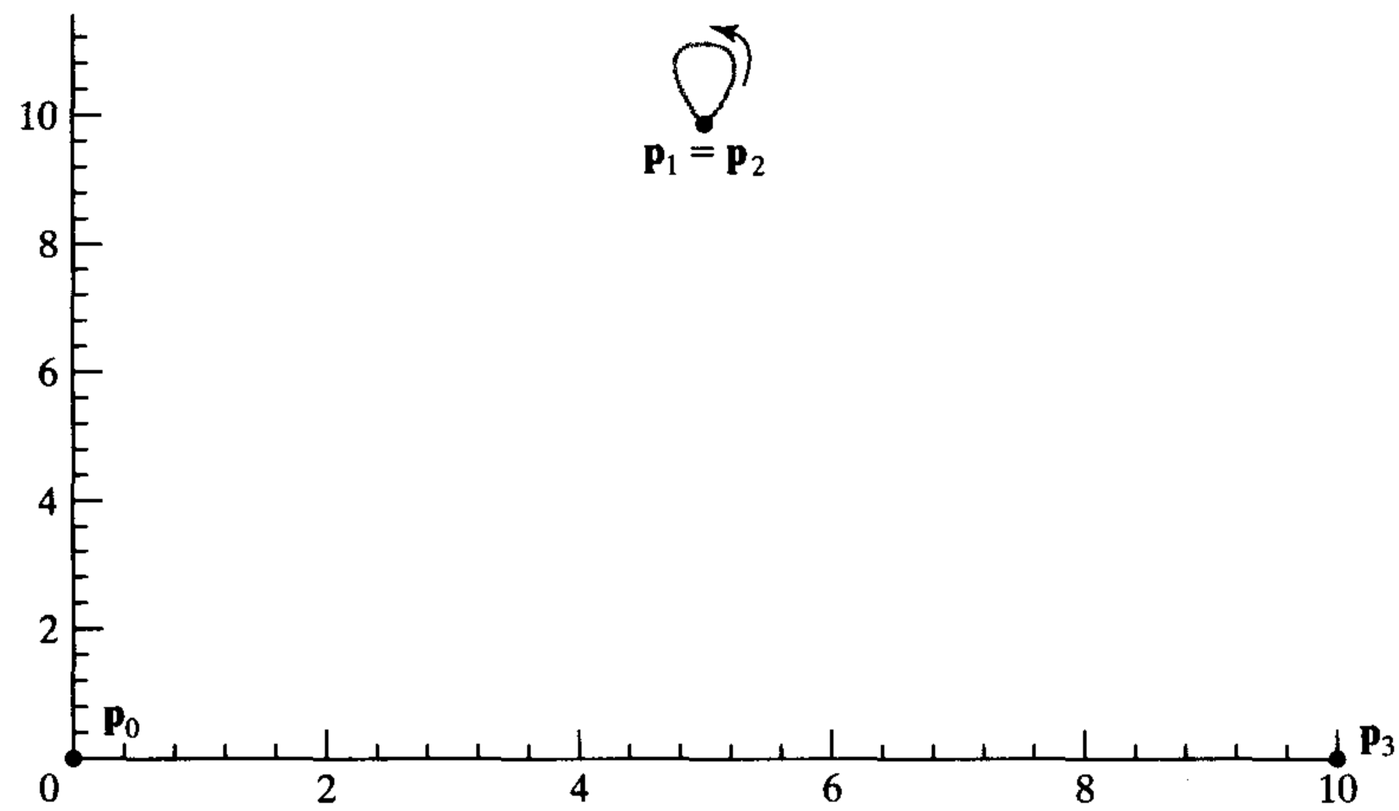


图 8.30　两端点重合生成的 cardinal 样条圈。张量参数设成 0

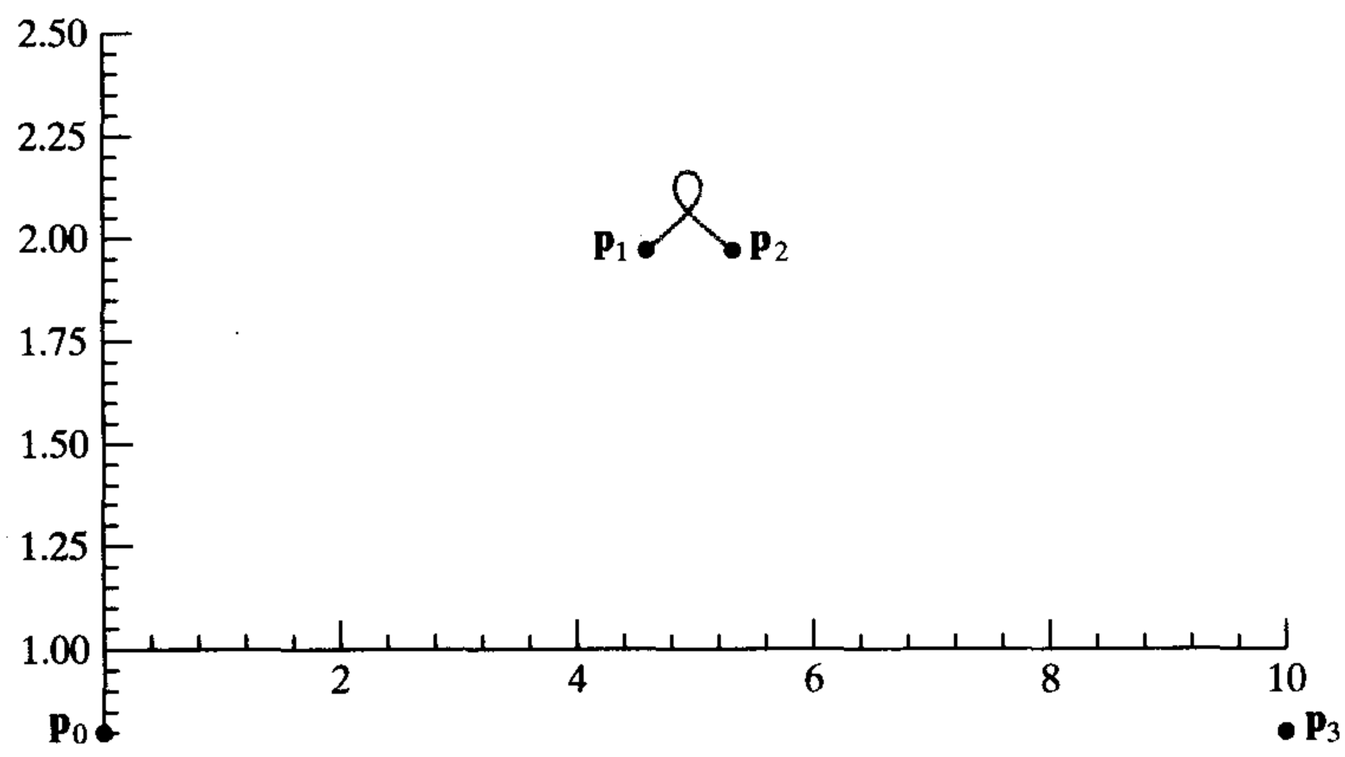


图 8.31　两端点靠近生成一条自相交 cardinal 样条曲线。张量参数设成 0

给出四个连续控制点，记为 $\mathbf{p}_{k-1}$、$\mathbf{p}_k$、$\mathbf{p}_{k+1}$ 和 $\mathbf{p}_{k+2}$，在 $\mathbf{p}_k$ 和 $\mathbf{p}_{k+1}$ 间的 Kochanek-Bartels 曲线段中的边界条件定义为

$$
\begin{aligned}
\mathbf{P}(0) &= \mathbf{p}_k \\
\mathbf{P}(1) &= \mathbf{p}_{k+1} \\
\mathbf{P}'(0)_{\text{in}} &= \frac{1}{2}(1-t)[(1+b)(1-c)(\mathbf{p}_k - \mathbf{p}_{k-1}) \\
&\qquad + (1-b)(1+c)(\mathbf{p}_{k+1} - \mathbf{p}_k)] \\
\mathbf{P}'(1)_{\text{out}} &= \frac{1}{2}(1-t)[(1+b)(1+c)(\mathbf{p}_{k+1} - \mathbf{p}_k) \\
&\qquad + (1-b)(1-c)(\mathbf{p}_{k+2} - \mathbf{p}_{k+1})]
\end{aligned}
\tag{8.36}
$$

其中 t 是**张量**（tension）参数，b 是**偏离**（bias）参数，c 是**连续性**（continuity）参数。在 Kochanek-Bartels 公式中，导数在曲线段边界处不一定连续。

张量参数 t 具有 cardinal 样条公式中同样的解释，即该参数控制曲线段的松紧程度。偏离参数 b 用来调整曲线段在端点处弯曲的数值，因此曲线段可以偏向一个端点或另一个端点（参见图8.32）。参数 c 控制切向量在曲线段边界处的连续性。若 c 取非零值，则曲线在曲线段边界处的斜率上具有不连续性。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Kochanek-Bartels 样条的设计是为了模拟动画路径。特别是当对象运动有突变时，可以通过为参数 c 取非零值而进行模拟。这些运动的改变用于卡通动画，例如当一个卡通角色突然停止、改变方向或与另一对象碰撞时。

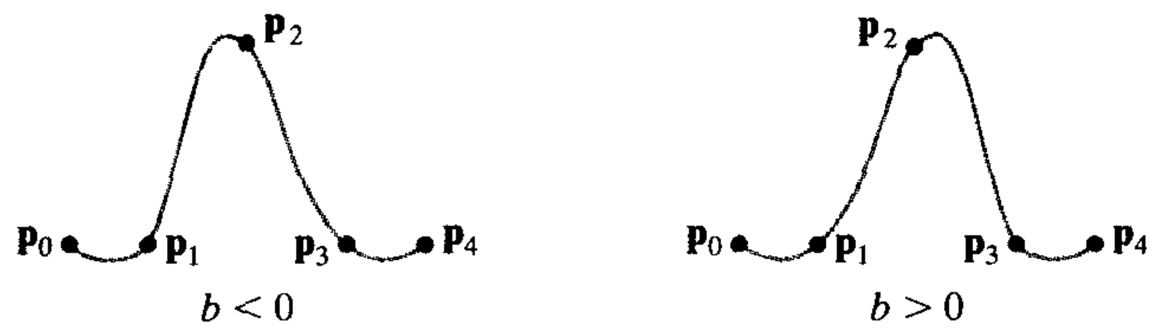


图 8.32　偏离参数 b 在 Kochanek-Bartels 样条段形状中所起的作用

8.10　Bézier 样条曲线

这个样条逼近方法是法国工程师 Pierre Bézier 为雷诺（Renault）公司设计汽车车身而开发的。**Bézier 样条**有很多更好、更方便地用于曲线和曲面设计的性质，并且也更容易实现。基于这些原因，Bézier 样条在各种 CAD 系统中、在大多数图形系统中、在相关的绘图和图形软件包中都有广泛的应用。

尽管有的图形软件包将控制点数限定为 4，但一般来说，Bézier 曲线段可以拟合任何数目的控制点。Bézier 多项式的次数取决于曲线将逼近的控制点数量及相关位置。类似于插值样条，Bézier 曲线可以使用混合函数、特征矩阵或边界条件来指定曲线路径与控制点的接近程度。对于一般对控制点不加约束的 Bézier 曲线，混合函数描述是最方便的表示。

8.10.1　Bézier 曲线公式

假设给出 $n+1$ 个控制点位置：$\mathbf{p}_k=(x_k, y_k, z_k)$，这里 k 可以取 0 到 n。这些坐标点将混合产生下列位置向量 $\mathbf{P}(u)$，用来描述 $\mathbf{p}_0$ 和 $\mathbf{p}_n$ 间逼近 Bézier 多项式函数的路径：

$$\mathbf{P}(u)=\sum_{k=0}^{n}\mathbf{p}_k\,\mathrm{BEZ}_{k,n}(u),\qquad 0\leqslant u\leqslant 1 \tag{8.37}$$

Bézier 混合函数 $\mathrm{BEZ}_{k,n}(u)$是 Bernstein 多项式：

$$\mathrm{BEZ}_{k,n}(u)=C(n,k)u^k(1-u)^{n-k} \tag{8.38}$$

这里，参数 $C(n, k)$是二项式系数：

$$C(n,k)=\frac{n!}{k!(n-k)!} \tag{8.39}$$

向量方程（8.37）表示单个曲线坐标三个参数方程的集合：

$$\begin{aligned}x(u)&=\sum_{k=0}^{n}x_k\,\mathrm{BEZ}_{k,n}(u)\\ y(u)&=\sum_{k=0}^{n}y_k\,\mathrm{BEZ}_{k,n}(u)\\ z(u)&=\sum_{k=0}^{n}z_k\,\mathrm{BEZ}_{k,n}(u)\end{aligned} \tag{8.40}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

多数情况下，Bézier 曲线是一个阶数比控制点数少 1 的多项式：三点生成一个抛物线，四点生成一个三次曲线，依次类推。图 8.33 给出了 xy 平面（$z=0$）上不同数量控制点生成的 Bézier 曲线。而在一定安排的控制点控制下，可获得退化的 Bézier 多项式。例如，使用三个共线控制点生成的 Bézier 曲线是一直线段。而所有控制点都在同一坐标位置时生成的 Bézier "曲线"是一个点。

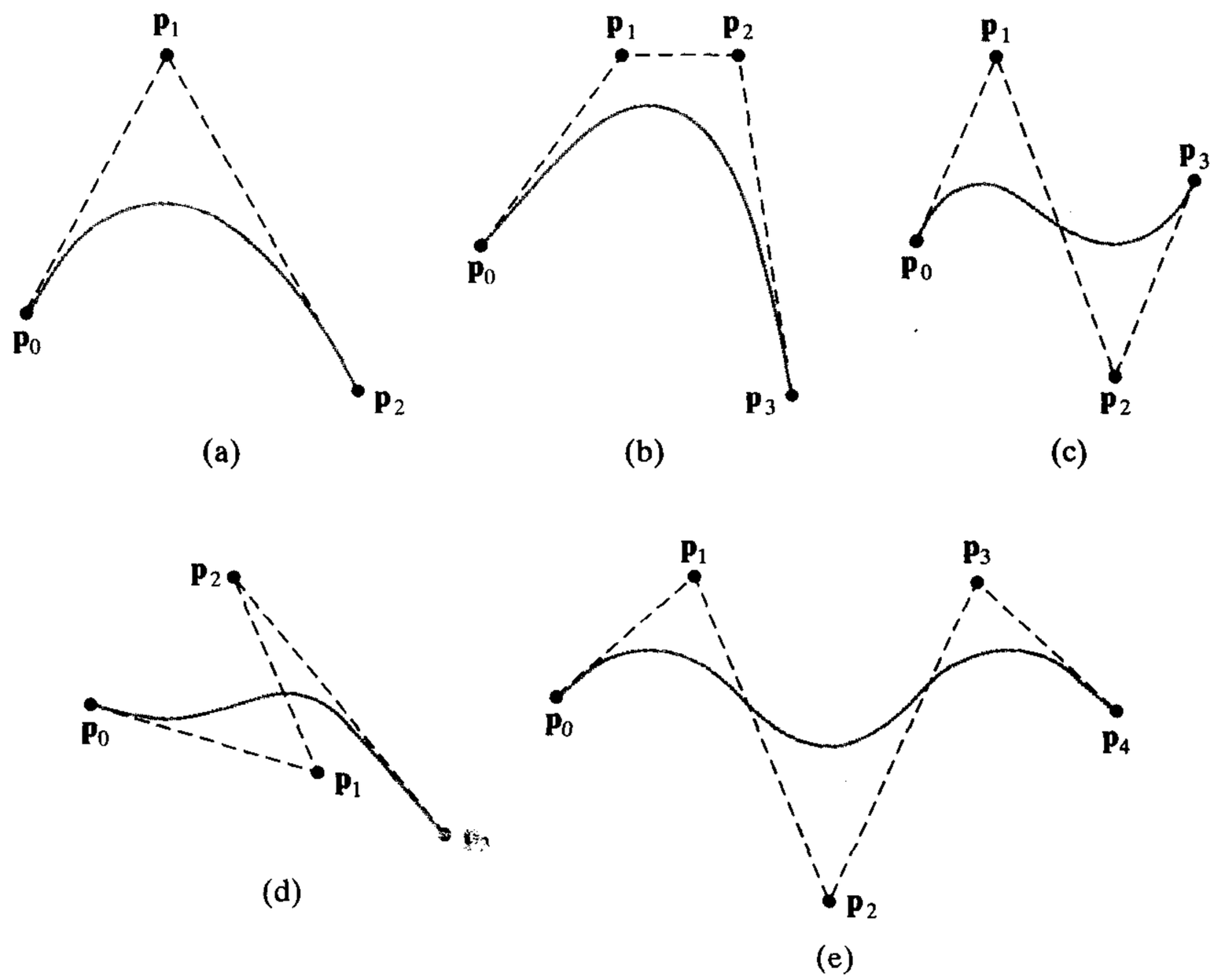


图 8.33 由三个、四个和五个控制点生成的二维 Bézier 曲线的例子，虚线连接了控制点的位置

连续的二项式系数可以用递归计算来得到：

$$C(n,k)=\frac{n-k+1}{k}C(n,k-1) \tag{8.41}$$

且 $n \geqslant k$。Bézier 混合函数也符合递归关系：

$$\mathrm{BEZ}_{k,n}(u)=(1-u)\mathrm{BEZ}_{k,n-1}(u)+u\,\mathrm{BEZ}_{k-1,n-1}(u), \qquad n>k\geqslant 1 \tag{8.42}$$

其中 $\mathrm{BEZ}_{k,k}=u^k$ 且 $\mathrm{BEZ}_{0,k}=(1-u)^k$。

8.10.2 Bézier 曲线生成程序示例

下面的程序实现了生成二维三次 Bézier 样条曲线的 Bézier 混合函数的计算。在 xy 平面上定义四个控制点，使用宽度为 4 的像素绘制沿曲线路径的 1000 个像素位置。二项式系数的值用过程 binomialCoeffs 计算，而沿曲线路径的坐标位置在过程 computeBezPt 中计算。这些值传递给过程 bezier，然后用 OpenGL 画点函数绘出这些像素位置。另一种方法是使用较少的点用直线段逼近曲线路径。8.17 节给出一种沿样条曲线位置生成坐标位置的更有效的方法。在本例中，世界坐标范围设定为仅显示在视口（参见图 8.34）中的曲线点。如果我们还要绘出控制点位置、控制图或凸壳，则可以扩大世界坐标裁剪窗口。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

/*  Set initial size of the display window.  */
GLsizei winWidth = 600, winHeight = 600;

/*  Set size of world-coordinate clipping window.  */
GLfloat xwcMin = -50.0, xwcMax = 50.0;
GLfloat ywcMin = -50.0, ywcMax = 50.0;

class wcPt3D {
   public:
      GLfloat x, y, z;
};

void init (void)
{
   /*  Set color of display window to white.  */
   glClearColor (1.0, 1.0, 1.0, 0.0);
}

void plotPoint (wcPt3D bezCurvePt)
{
    glBegin (GL_POINTS);
        glVertex2f (bezCurvePt.x, bezCurvePt.y);
    glEnd ( );
}

/*  Compute binomial coefficients C for given value of n.  */
void binomialCoeffs (GLint n, GLint * C)
{
   GLint k, j;

   for (k = 0;  k <= n;  k++) {
      /*  Compute n!/(k!(n - k)!).  */
      C [k] = 1;
      for (j = n;  j >= k + 1;  j--)
        C [k] *= j;
      for (j = n - k;  j >= 2;  j--)
        C [k] /= j;
   }
}

void computeBezPt (GLfloat u, wcPt3D * bezPt, GLint nCtrlPts,
                   wcPt3D * ctrlPts, GLint * C)
{
   GLint k, n = nCtrlPts - 1;
   GLfloat bezBlendFcn;

   bezPt->x = bezPt->y = bezPt->z = 0.0;

   /*  Compute blending functions and blend control points. */
   for (k = 0; k < nCtrlPts; k++) {
      bezBlendFcn = C [k] * pow (u, k) * pow (1 - u, n - k);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        bezPt->x += ctrlPts [k].x * bezBlendFcn;
        bezPt->y += ctrlPts [k].y * bezBlendFcn;
        bezPt->z += ctrlPts [k].z * bezBlendFcn;
    }
}

void bezier (wcPt3D * ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
    wcPt3D bezCurvePt;
    GLfloat u;
    GLint *C, k;

    /*  Allocate space for binomial coefficients  */
    C = new GLint [nCtrlPts];

    binomialCoeffs (nCtrlPts - 1, C);
    for (k = 0;  k <= nBezCurvePts;  k++) {
        u = GLfloat (k) / GLfloat (nBezCurvePts);
        computeBezPt (u, &bezCurvePt, nCtrlPts, ctrlPts, C);
        plotPoint (bezCurvePt);
    }
    delete [ ] C;
}

void displayFcn (void)
{
    /*  Set example number of control points and number of
     *  curve positions to be plotted along the Bezier curve.
     */
    GLint nCtrlPts = 4, nBezCurvePts = 1000;

    wcPt3D ctrlPts [4] = { {-40.0, -40.0, 0.0}, {-10.0, 200.0, 0.0},
                           {10.0, -200.0, 0.0}, {40.0, 40.0, 0.0} };

    glClear (GL_COLOR_BUFFER_BIT);   //  Clear display window.
    glPointSize (4);
    glColor3f (1.0, 0.0, 0.0);       //  Set point color to red.

    bezier (ctrlPts, nCtrlPts, nBezCurvePts);
    glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
    /*  Maintain an aspect ratio of 1.0.  */
    glViewport (0, 0, newHeight, newHeight);

    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ( );

    gluOrtho2D (xwcMin, xwcMax, ywcMin, ywcMax);

    glClear (GL_COLOR_BUFFER_BIT);
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (50, 50);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Bezier Curve");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);

   glutMainLoop ( );
}
```

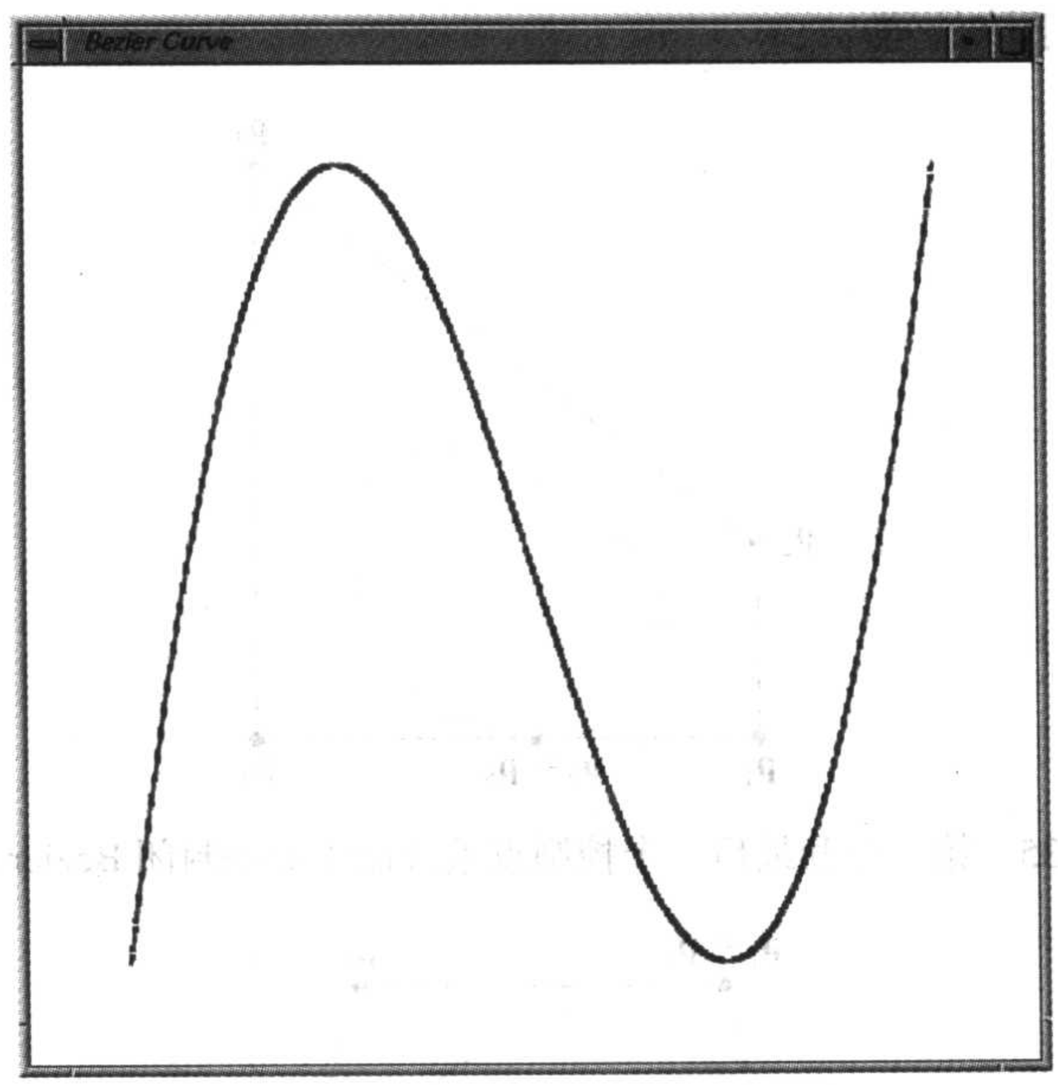


图 8.34　示例程序显示的 Bézier 曲线

8.10.3　Bézier 曲线的特性

Bézier曲线的一个非常有用的性质是该曲线总是通过第一个和最后一个控制点。即曲线在两个端点处的边界条件是

$$\begin{aligned}\mathbf{P}(0) &= \mathbf{p}_0 \\ \mathbf{P}(1) &= \mathbf{p}_n\end{aligned} \tag{8.43}$$

Bézier 曲线在端点处的一阶导数的参数值可以由控制点的坐标进行计算：

$$\begin{aligned}\mathbf{P}'(0) &= -n\mathbf{p}_0 + n\mathbf{p}_1 \\ \mathbf{P}'(1) &= -n\mathbf{p}_{n-1} + n\mathbf{p}_n\end{aligned} \tag{8.44}$$

从这些表达式可看出，曲线在始点处的切线落在前两个控制点的连线上，曲线在终点处的切线落在最后两个控制点的连线上。同样，Bézier 曲线在端点处的二阶导数可以计算为

$$\begin{aligned}\mathbf{P}''(0) &= n(n-1)[(\mathbf{p}_2 - \mathbf{p}_1) - (\mathbf{p}_1 - \mathbf{p}_0)] \\ \mathbf{P}''(1) &= n(n-1)[(\mathbf{p}_{n-2} - \mathbf{p}_{n-1}) - (\mathbf{p}_{n-1} - \mathbf{p}_n)]\end{aligned} \tag{8.45}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Bézier 曲线的另一个重要性质是其落在控制点的凸壳内（凸多边形边界）。这些点由 Bézier 混合函数给出，这些值都是正的且其总和为 1：

$$\sum_{k=0}^{n} \mathrm{BEZ}_{k,n}(u) = 1 \tag{8.46}$$

因此，任一曲线位置仅是控制点位置的加权和。Bézier 曲线的凸壳性质保证了多项式随控制点平稳前进而不会产生摆动。

8.10.4 使用 Bézier 曲线的设计技术

封闭 Bézier 曲线由重合第一个和最后一个控制点的位置而生成，如图 8.35 所示。多个控制点位于同一个位置也会对该位置加以更多的权。在图8.36中，输入一个坐标位置作为两个控制点，这样所产生的曲线更接近于该位置。

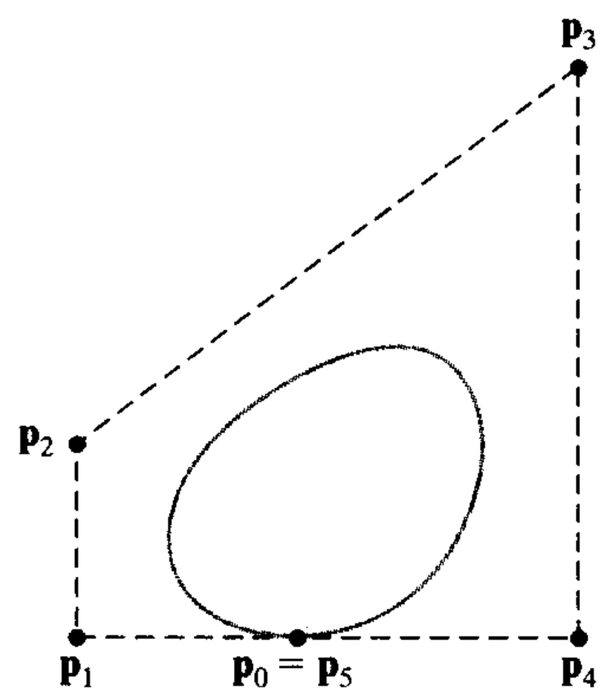


图 8.35 第一个和最后一个控制点重合而生成的封闭 Bézier 曲线

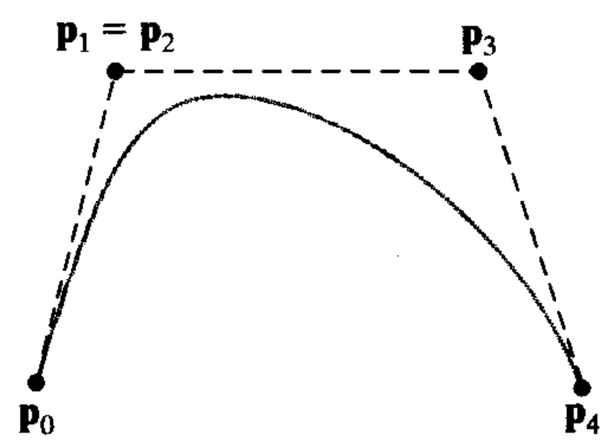


图 8.36 多个控制点具有相同的坐标位置，Bézier 曲线将更靠近该位置

我们可以使用任何数目的控制点来拟合一条Bézier曲线，但这需要计算更高次的多项式。复杂曲线可以由几个较低次数的Bézier曲线段连接而成。较小的曲线段连接也可以使我们更好地控制小区域内的曲线形状。由于Bézier曲线通过端点，因此比较容易匹配曲线段（0阶连续性）。Bézier曲线还有一个重要性质，即曲线在端点处的切向量位于端点和相邻控制点的连线上。因此，为了得到曲线段之间的一阶连续性，使得新段中的控制点 $\mathbf{p}_{0'}$ 和 $\mathbf{p}_{1'}$ 与前段中的控制点 $\mathbf{p}_{n-1}$ 和 $\mathbf{p}_n$ 在同一条直线上（参见图 8.37）。一旦两个曲线段具有相同数目的控制点，则选择前段中最后一个控制点为新段中的第一个控制点，新段中第二个控制点 $\mathbf{p}_{1'}$ 的位置为

$$\mathbf{p}_{1'} = \mathbf{p}_n + \frac{n}{n'}(\mathbf{p}_n - \mathbf{p}_{n-1}) \tag{8.47}$$

为了简化的 $\mathbf{p}_{1'}$ 安排，可以只要求几何连续性，从而将 $\mathbf{p}_{1'}$ 安排在 $\mathbf{p}_{n-1}$ 和 $\mathbf{p}_n$ 之间连线上的任何地方。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

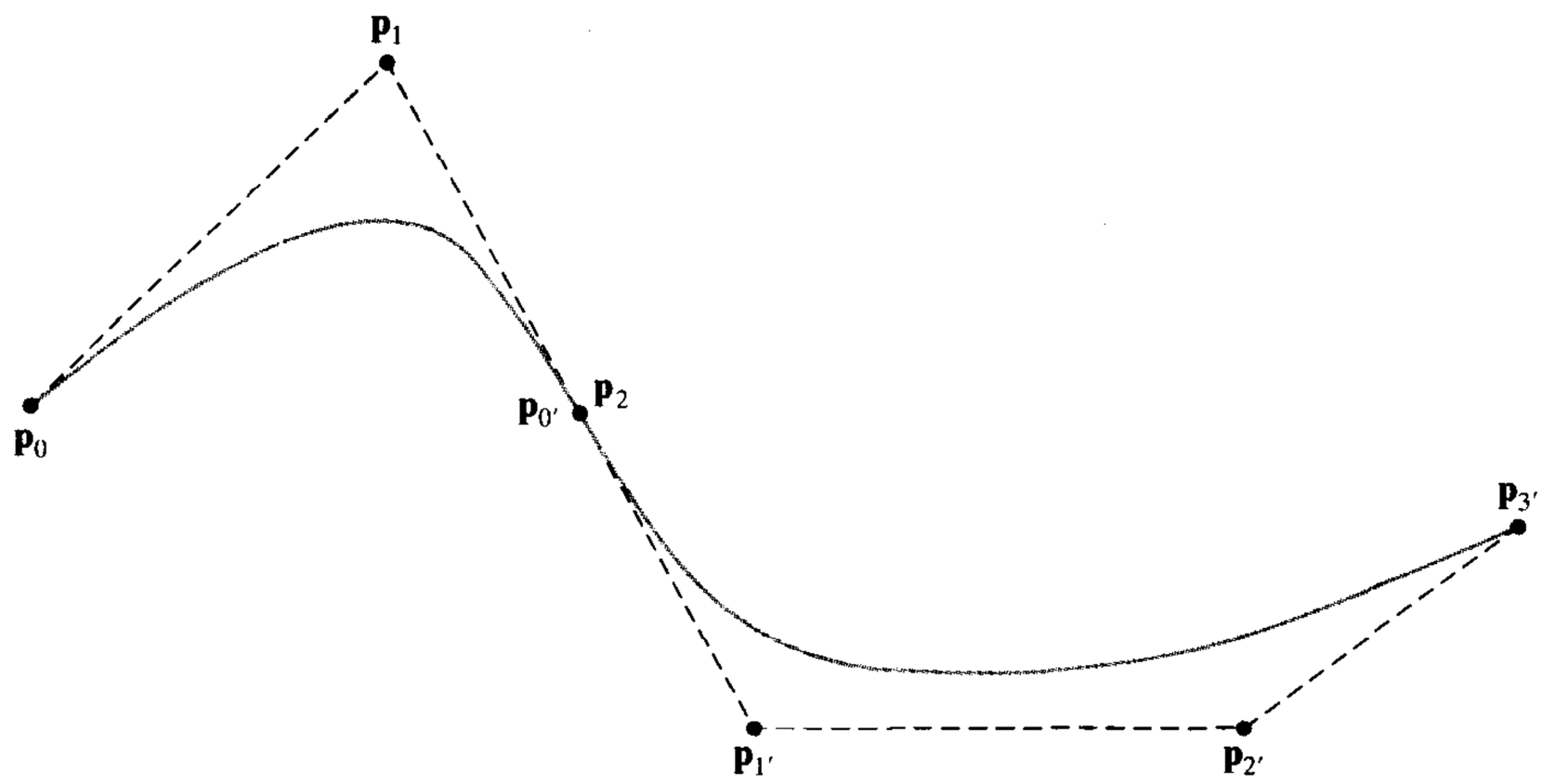


图 8.37　由两个 Bézier 曲线段形成的分段逼近曲线。设 $\mathbf{p}_{0'} = \mathbf{p}_2$，使 $\mathbf{p}_{1'}$、$\mathbf{p}_1$、$\mathbf{p}_2$ 共线，可以得到两条曲线段之间的 0 阶和一阶连续性

使用公式（8.45）中的表达式可以在两个相邻Bézier曲线段之间得到C^2连续性。这建立了除$\mathbf{p}_{0'}$和$\mathbf{p}_{1'}$外的一个控制点$\mathbf{p}_{2'}$，以满足C^0和C^1连续性的需要。但是，对Bézier曲线段不一定要求有两次连续性。对每段有四个控制点的三次曲线来说尤其是这样。此时，二次连续性固定了前三个控制点位置，只留下了一个点可用于调整该曲线段形状。

8.10.5　三次 Bézier 曲线

很多图形软件包只有三次样条显示函数。这不仅带来了设计上的方便性，同时避免了由于高阶多项式而使计算量增加。三次 Bézier 曲线由四个控制点生成。将 $n = 3$ 代入方程（8.38），得到三次 Bézier 曲线的四个混合函数，如下所示：

$$\begin{aligned} \mathrm{BEZ}_{0,3} &= (1-u)^3 \\ \mathrm{BEZ}_{1,3} &= 3u(1-u)^2 \\ \mathrm{BEZ}_{2,3} &= 3u^2(1-u) \\ \mathrm{BEZ}_{3,3} &= u^3 \end{aligned} \tag{8.48}$$

图 8.38 给出了四个三次 Bézier 混合函数的图示。混合函数的形式决定了控制点如何影响曲线的形状，此时参数 u 的取值范围从 0 到 1。$u = 0$ 时，非零的混合函数是 $\mathrm{BEZ}_{0,3}$，其值为 1。$u = 1$ 时，非零混合函数是 $\mathrm{BEZ}_{3,3}(1) = 1$。这样，三次 Bézier 曲线总是通过控制点 $\mathbf{p}_0$ 和 $\mathbf{p}_3$。其他两个函数 $\mathrm{BEZ}_{1,3}$ 和 $\mathrm{BEZ}_{2,3}$ 影响参数 u 取中间值时的曲线形状，因此，生成的曲线靠近 $\mathbf{p}_1$ 和 $\mathbf{p}_2$。混合函数 $\mathrm{BEZ}_{1,3}$ 当 $u = 1/3$ 时取到最大值，而 $\mathrm{BEZ}_{2,3}$ 当 $u = 2/3$ 时取到最大值。

注意到在图 8.38 中，每个混合函数在参值 u 的整个取值范围内不为 0。这样，Bézier 曲线不可能对曲线形状进行局部控制。如果改变任意一个控制点的位置，那么整个曲线将受到影响。

在三次 Bézier 曲线的端点处，一阶导数（斜率）为

$$\mathbf{P}'(0) = 3(\mathbf{p}_1 - \mathbf{p}_0), \qquad \mathbf{P}'(1) = 3(\mathbf{p}_3 - \mathbf{p}_2)$$

二阶导数为

$$\mathbf{P}''(0) = 6(\mathbf{p}_0 - 2\mathbf{p}_1 + \mathbf{p}_2), \qquad \mathbf{P}''(1) = 6(\mathbf{p}_1 - 2\mathbf{p}_2 + \mathbf{p}_3)$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

使用一系列三次Bézier曲线段可构造复杂的样条曲线。利用导数表达式，可以在两条曲线段之间构造具有 C^1 或 C^2 连续性的分段曲线，尽管这时我们无法利用前三个控制点的位置。

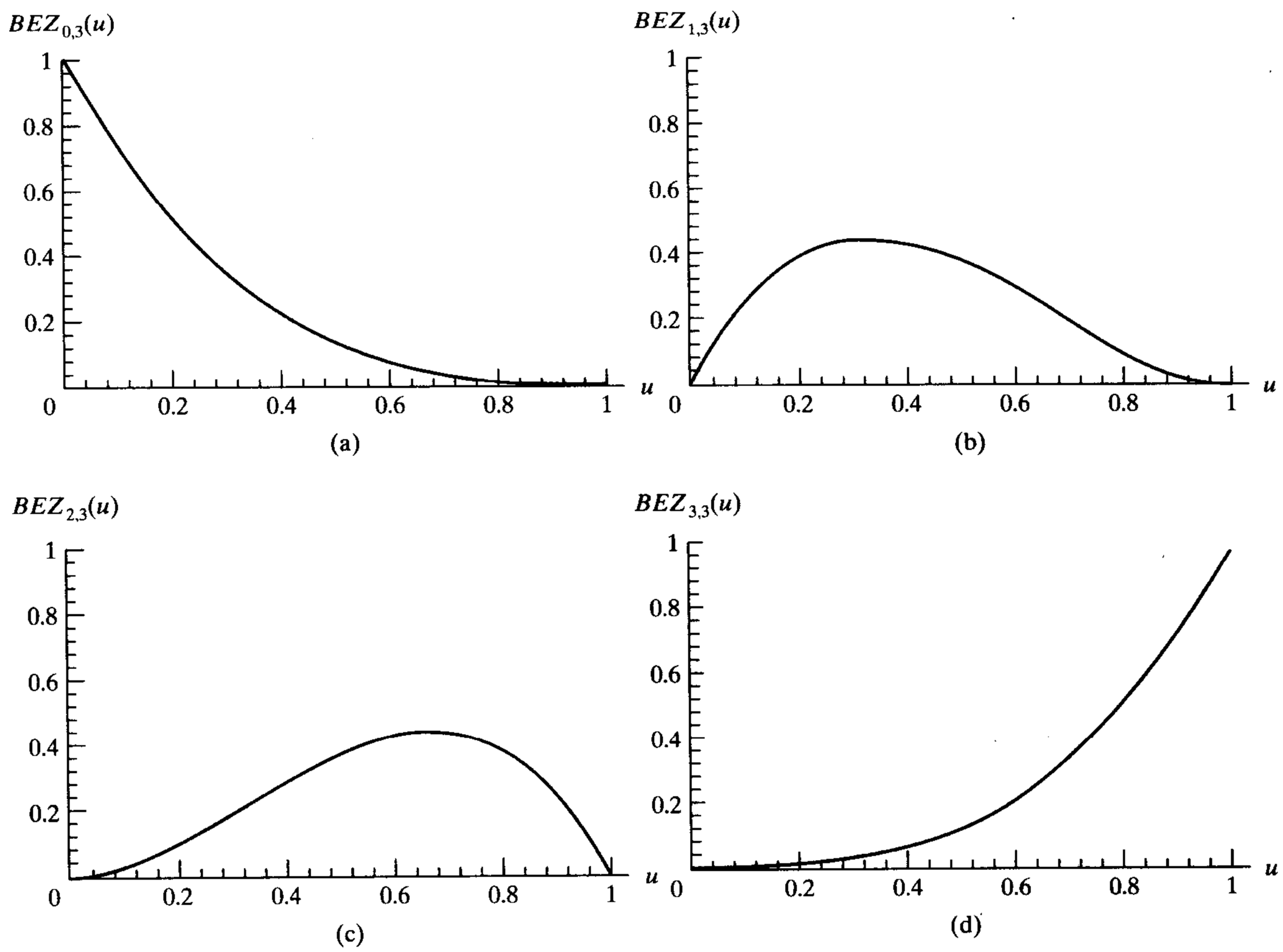


图 8.38 三次曲线的四个 Bézier 混合函数

将混合函数和约束扩展为多项式表达式，那么三次 Bézier 函数可以写成矩阵形式：

$$\mathbf{P}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \mathbf{M}_{\text{Bez}} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \tag{8.49}$$

其中，Bézier 矩阵为

$$\mathbf{M}_{\text{Bez}} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{8.50}$$

类似于插值样条中的处理，可以引入额外的参数来调整曲线的“张量”和“偏离”程度。但作用更大的 B 样条、β 样条都提供了这一功能。

8.11 Bézier 曲面

可以使用两组正交的 Bézier 曲线来设计一个对象曲面。Bézier 曲面的参数向量函数可以按照 Bézier 混合函数的笛卡儿积形式来描述：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{P}(u, v) = \sum_{j=0}^{m} \sum_{k=0}^{n} \mathbf{p}_{j,k} \, \mathrm{BEZ}_{j,m}(v) \, \mathrm{BEZ}_{k,n}(u) \tag{8.51}$$

其中的 $\mathbf{p}_{j,k}$ 指定了 $(m+1)\times(n+1)$ 个控制点的位置。

图 8.39 给出了两个 Bézier 曲面。控制点由虚线连接，实线表示常数 u 的曲线和常数 v 的曲线。常数 u 的曲线是通过 u 取单位区间内的一定值并将 v 从 0 变到 1 而绘制出的。常数 v 的曲线绘制方法也类似。

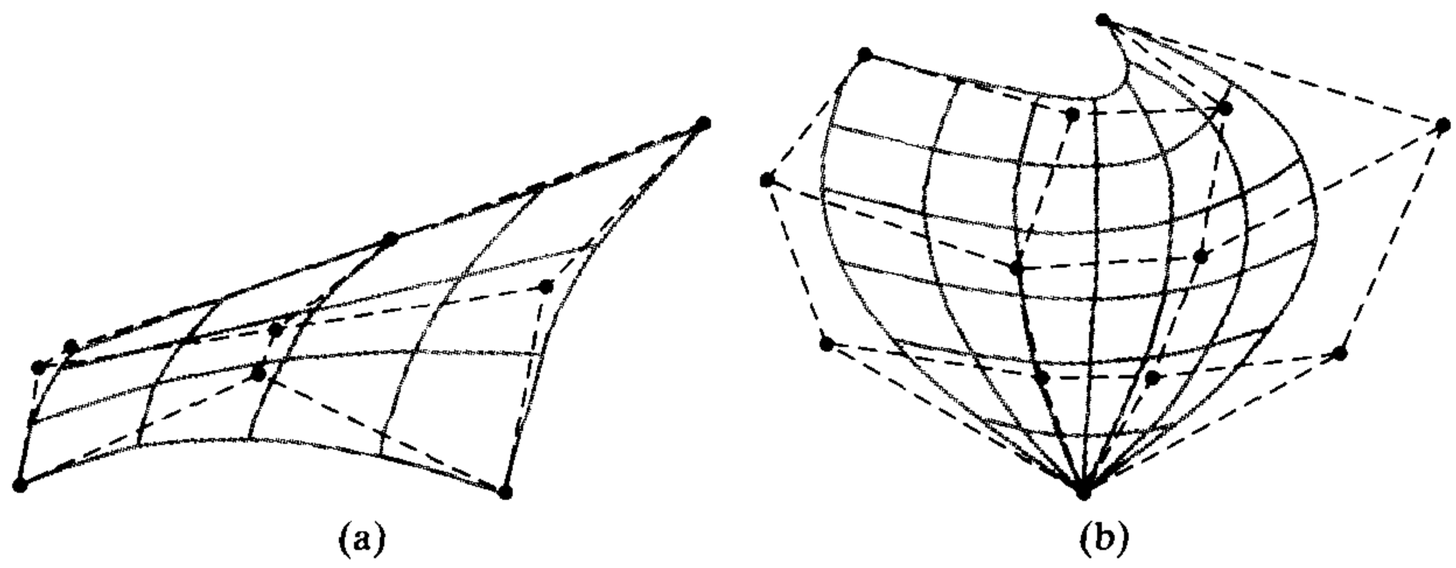


图 8.39　线框式 Bézier 曲面由(a) $m=3$、$n=3$ 和 (b) $m=4$、$n=4$ 所构造。虚线连接控制点

Bézier 曲面与 Bézier 曲线有相同的性质，可以提供用于交互设计应用的便捷方法。对于每一个曲面片，选择在 xy “地” 平面上的控制点网格，然后根据控制点的 z 坐标值在地平面上选择高度。而这些曲面片可以通过边界约束而连接。曲面面片可用多边形和第10章讨论的浓淡绘制技术表示。

图8.40给出了用两个Bézier曲面形成的一个曲面。类似于在曲线中通过在边界线上建立0阶和一阶连续性，可以确保从一个部分平滑转换到另一个部分。只要在边界上匹配控制点就可以获得0阶连续性。一阶连续性的获得需要选取穿过边界的直线上的控制点，并且对穿过边界的一组指定控制点形成的共线线段保持一个常数比例。

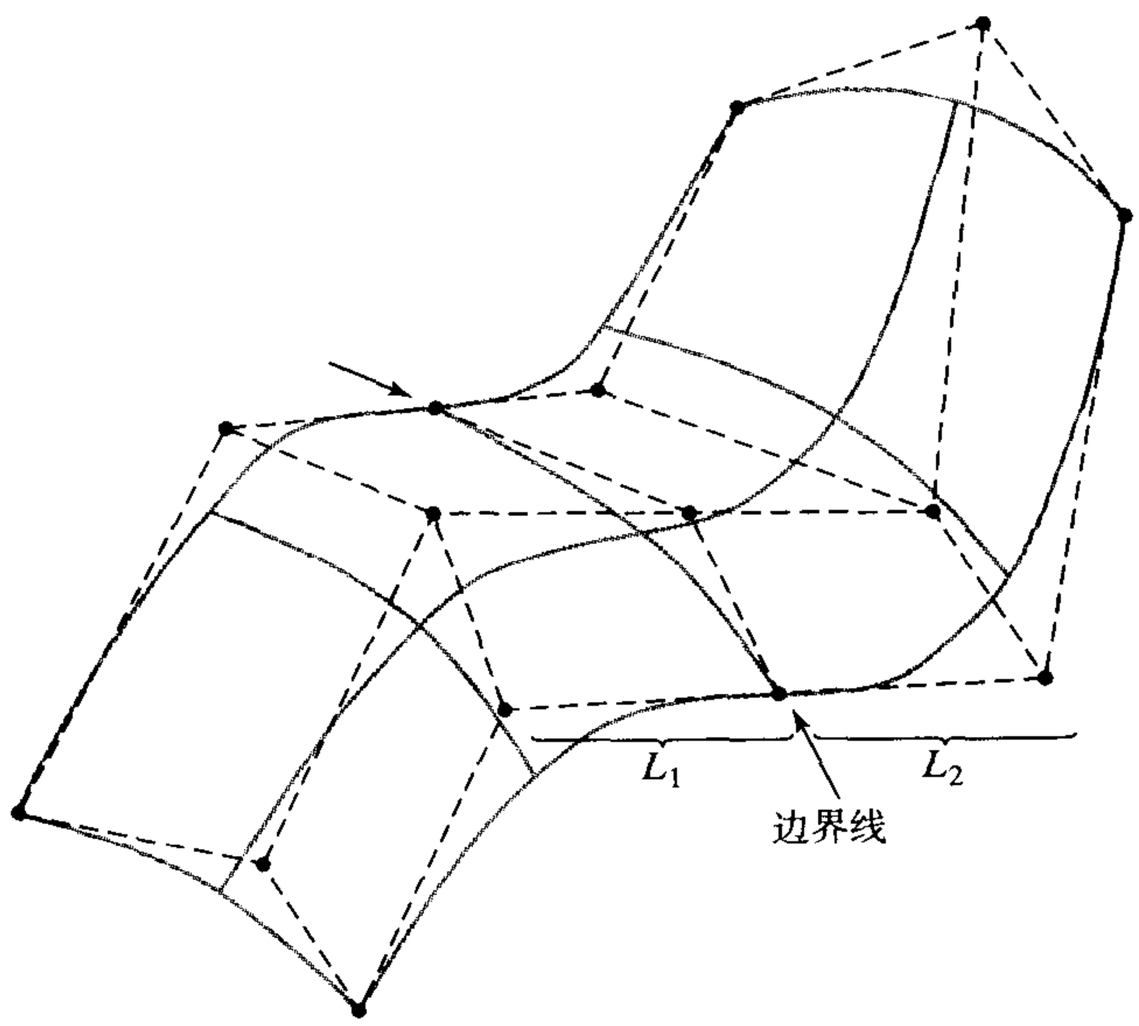


图 8.40　由两个Bézier曲面组成的Bézier曲面，在指定边界线处连接。虚线连接控制点。一阶连续性，则根据在两曲面间穿过边界的共线控制点所形成的线段长度 L_1 与线段长度 L_2 之比为常数而得到

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.12 B样条曲线

下面将要介绍使用更为广泛的逼近样条类。**B样条**函数广泛应用于CAD系统和许多图形程序设计软件包中。与Bézier样条一样，B样条也通过逼近一组控制点来生成。但是B样条有两个Bézier样条所不具备的优点：(1) B样条多项式次数可独立于控制点数目（有一定限制），(2) B样条允许局部控制曲线或曲面。缺点是B样条比Bézier样条更复杂。

8.12.1 B样条曲线

我们可以把沿B样条曲线的坐标位置的计算表示写成混合函数公式的表达式：

$$\mathbf{P}(u)=\sum_{k=0}^{n}\mathbf{p}_k B_{k,d}(u),\qquad u_{\min}\leqslant u\leqslant u_{\max},\quad 2\leqslant d\leqslant n+1 \tag{8.52}$$

其中，$\mathbf{p}_k$ 是输入的一组 $n+1$ 个控制点。B样条公式与Bézier样条公式相比有几个不同点。参数 u 的范围取决于B样条其他参数的选取。B样条混合函数 $B_{k,d}$ 是次数为 d–1 的多项式，其中参数 d 是一个**阶参数**（degree parameter）。（有时参数 d 暗指多项式的“幂次”，但这可能被误解，因为术语幂次常用来简单地表示多项式的次数，即 d–1。）阶参数 d 可赋以2到控制点个数 $n+1$ 之间的任意整数。实际上，也可以设 d 的值为1，但这时“曲线”正好是该控制点本身。B样条的局部控制可以由定义在 u 取值范围中子区间上的混合函数来实现。

B样条曲线的混合函数由Cox-deBoor递归公式定义为

$$\begin{aligned}B_{k,1}(u)&=\begin{cases}1 & ,\ u_k\leqslant u\leqslant u_{k+1}\\ 0 & ,\ \text{其他}\end{cases}\\ B_{k,d}(u)&=\frac{u-u_k}{u_{k+d-1}-u_k}B_{k,d-1}(u)+\frac{u_{k+d}-u}{u_{k+d}-u_{k+1}}B_{k+1,d-1}(u)\end{aligned} \tag{8.53}$$

其中，每个混合函数定义在 u 的取值范围的 d 子区间上。所选的一组子区间端点 u_j 称为**节点**（knot），而选定的一组子区间端点整体称为**节点向量**（knot vector）。可以选取满足 $u_j\leqslant u_{j+1}$ 的任意值作为子区间端点。$u_{\min}$ 和 $u_{\max}$ 的值取决于所选的控制点个数、参数 d 的取值、如何建立子区间（节点向量）。由于可以选取节点向量的元素，因此在Cox-deBoor计算中的某些分母值为0，该公式假定求解诸如0/0的值时，其结果为0。

图8.41表示了B样条的局部控制特征。除了局部控制，B样条允许通过改变控制点的个数来设计一条曲线，而不需改变多项式的次数，也可以增加或修改控制点的数目来控制曲线形状。类似地，可以增加节点向量的值从而辅助曲线的设计。但是在进行这种处理时，也需要增加控制点，因为节点向量的大小取决于参数 n。

B样条具有下列性质：

- 在 u 取值范围内，多项式曲线的次数为 d–1，并且具有 C^{d-2} 连续性；
- 对于 n+1 个控制点，曲线由 $n+1$ 个混合函数进行描述；
- 每个混合函数 $\mathrm{B}_{k,d}$ 定义在 u 取值范围的 d 子区间上，以节点向量值 u_k 为起点；
- 参数 u 的取值范围由 $n+d+1$ 个节点向量中指定的值分成 $n+d$ 个子区间；

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

- 节点值记为 $\{u_0, u_1, \ldots, u_{n+d}\}$，所生成的 B 样条曲线仅定义在从节点值 u_{d-1} 到节点值 u_{n+1} 的区间上；
- 每个样条曲线段（在两个相邻节点值间）受 d 个控制点影响；
- 任意一个控制点可以影响最多 d 个曲线段的形状。

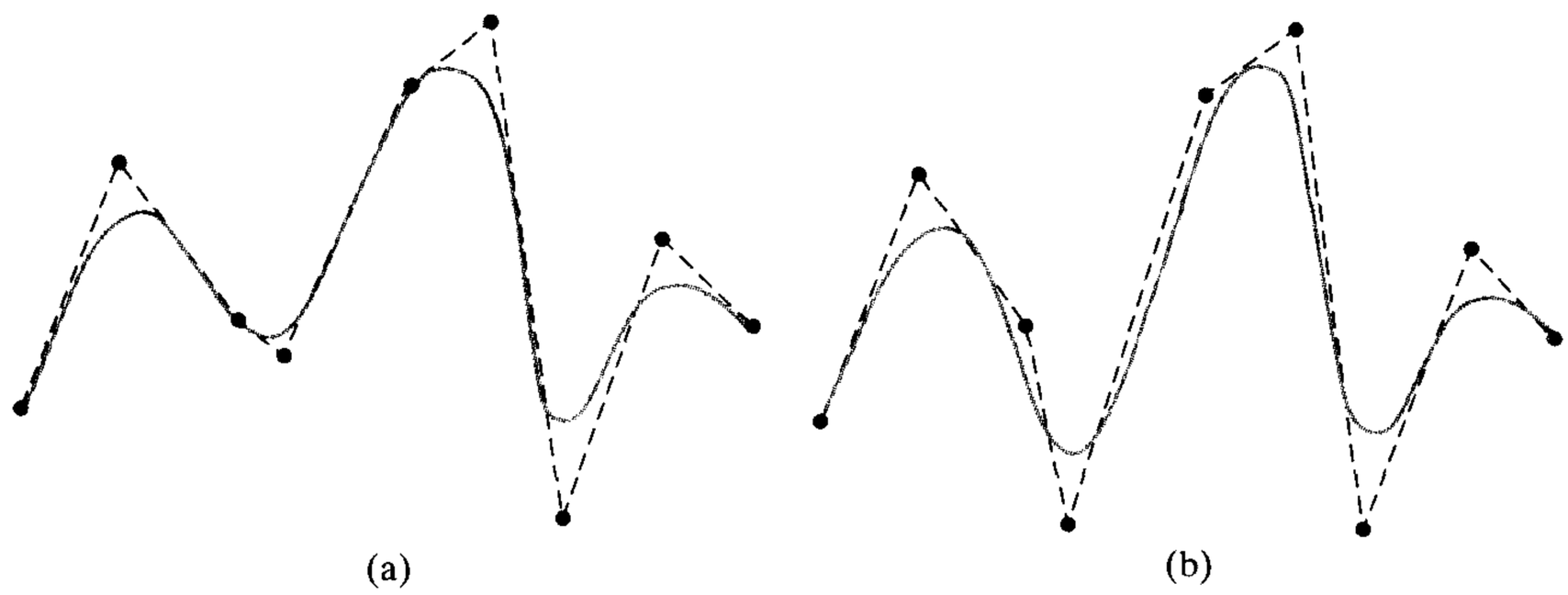


图 8.41　B 样条曲线的局部调整。(a)中改变一个控制点生成曲线 (b)，这仅改变了该控制点附近的一些部分

另外，B 样条曲线位于最多由 $d+1$ 个控制点所形成的凸壳内，因此 B 样条与控制点位置密切关联。对从节点值 u_{d-1} 到 u_{n+1} 区间上的任意值 u，所有的基本函数之和为 1：

$$\sum_{k=0}^{n} B_{k,d}(u) = 1 \tag{8.54}$$

给出控制点位置和参数 d 的值，则需要指定节点值，并使用等式（8.53）中的递归关系来获得混合函数。节点向量有三种分类：均匀的（uniform）、开放均匀的（open uniform）和非均匀的（nonuniform）。B 样条通常根据所选的节点向量类型进行描述。

8.12.2　均匀周期性 B 样条曲线

当节点值间的距离为常数时，所生成的曲线称为均匀 B 样条。例如，可以建立均匀节点向量为

$$\{-1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0\}$$

通常节点值的标准取值范围介于 0 和 1 之间，例如：

$$\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$$

在很多应用中建立起以 0 为初始值、1 为间距的均匀节点值是比较方便的，其节点向量为

$$\{0, 1, 2, 3, 4, 5, 6, 7\}$$

均匀 B 样条具有**周期性**（periodic）混合函数。即给定 n 和 d 值，所有的混合函数具有相同的形状。每个后继混合函数仅仅是前面函数平移的结果：

$$B_{k,d}(u) = B_{k+1,d}(u + \Delta u) = B_{k+2,d}(u + 2\Delta u) \tag{8.55}$$

其中，Δu 是相邻节点值间的区间。图 8.42 给出了在下例中为四个控制点的曲线生成的二次均匀 B 样条混合函数。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

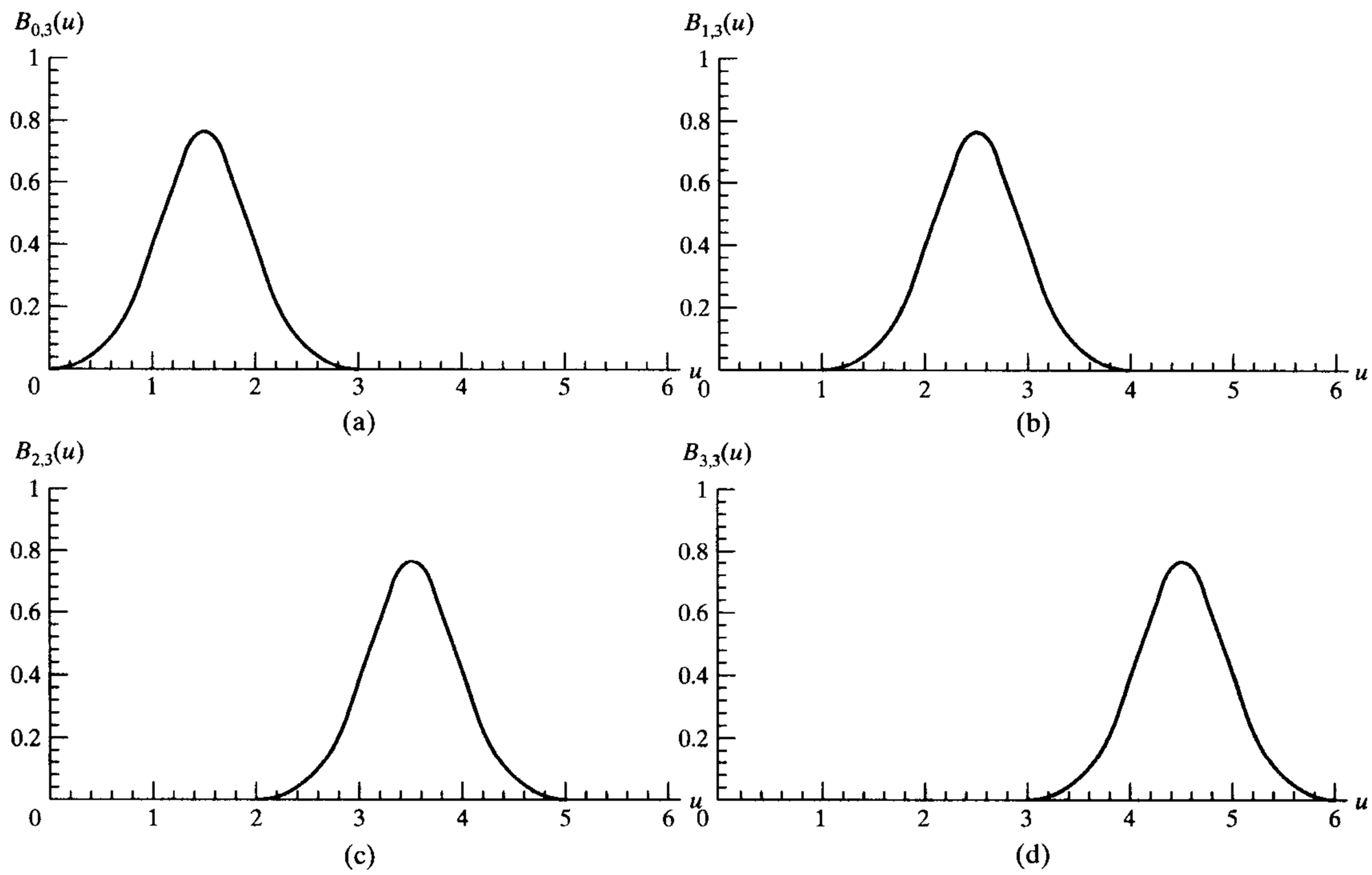


图 8.42 周期性 B 样条混合函数，并且 $n=d=3$，具有均匀的整数节点向量

例 8.1 均匀、二次 B 样条

为了展示均匀、整数节点值的 B 样条混合函数，选择参数 $d=n=3$。则节点向量包括 $n+d+1=7$ 个节点：

$$\{0, 1, 2, 3, 4, 5, 6\}$$

而参数 u 的范围从 0 到 6，有 $n+d=6$ 个子区间。

四个混合函数中的每一个定义在整个 u 范围中 $d=3$ 个子区间内。使用等式（8.53）中的递归关系，可以获得第一个混合函数如下：

$$B_{0,3}(u)=\begin{cases}\frac{1}{2}u^2, & 0\leqslant u<1\\ \frac{1}{2}u(2-u)+\frac{1}{2}(u-1)(3-u), & 1\leqslant u<2\\ \frac{1}{2}(3-u)^2, & 2\leqslant u<3\end{cases}$$

使用公式（8.55）中的关系，用 u–1 代替 $B_{0,3}$ 中的 u，并将开始位置移到 1：

$$B_{1,3}(u)=\begin{cases}\frac{1}{2}(u-1)^2, & 1\leqslant u<2\\ \frac{1}{2}(u-1)(3-u)+\frac{1}{2}(u-2)(4-u), & 2\leqslant u<3\\ \frac{1}{2}(4-u)^2, & 3\leqslant u<4\end{cases}$$

类似地，通过连续将 $B_{1,3}$ 向右移动，可得余下两个周期性函数：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$B_{2,3}(u)=\begin{cases}\frac{1}{2}(u-2)^2, & 2\leqslant u<3\\ \frac{1}{2}(u-2)(4-u)+\frac{1}{2}(u-3)(5-u), & 3\leqslant u<4\\ \frac{1}{2}(5-u)^2, & 4\leqslant u<5\end{cases}$$

$$B_{3,3}(u)=\begin{cases}\frac{1}{2}(u-3)^2, & 3\leqslant u<4\\ \frac{1}{2}(u-3)(5-u)+\frac{1}{2}(u-4)(6-u), & 4\leqslant u<5\\ \frac{1}{2}(6-u)^2, & 5\leqslant u<6\end{cases}$$

图 8.42 给出了表达 B 样条局部特征的四个周期性二次混合函数的图示。第一个控制点与混合函数 $B_{0,3}(u)$ 相乘。因此，改变第一个控制点的位置最多仅影响 $u=3$ 之前的曲线形状。同样，最后一个控制点影响样条曲线定义在 $B_{3,3}(u)$ 的区间上的部分。

图 8.42 给出了本例中 B 样条曲线的限制。所有混合函数都在 $u_{d-1}=2$ 到 $u_{n+1}=4$ 的区间上出现。但不是所有混合函数都会在 2 以下和 4 以上的区域中出现。2 到 4 的区域是多项式曲线的范围，且方程（8.54）在该区域是有效的。因此，在该区间中的所有混合函数总和为 1。在此区间之外，不能对所有的混合函数求和，因为函数不是全部定义在2以下和4以上的区间内。

由于所得多项式曲线的取值从 2 到 4，通过求解混合函数在这些点的值，可以确定曲线的起点和终点位置，得到

$$\mathbf{P}_{\text{start}}=\frac{1}{2}(\mathbf{p}_0+\mathbf{p}_1),\qquad \mathbf{P}_{\text{end}}=\frac{1}{2}(\mathbf{p}_2+\mathbf{p}_3)$$

这样，曲线起始于前两个控制点的中间点位置，终止于后两个控制点的中间点位置。

也可以确定曲线的起点和终点处的导数。对混合函数求导，以端点值替换参数 u，可以发现：

$$\mathbf{P}'_{\text{start}}=\mathbf{p}_1-\mathbf{p}_0,\qquad \mathbf{P}'_{\text{end}}=\mathbf{p}_3-\mathbf{p}_2$$

曲线在起点处的斜率平行于前两个控制点的连线，在终点处的斜率平行于后两个控制点的连线。

图 8.43 给出了 xy 平面上四个控制点确定的二次周期性 B 样条曲线。

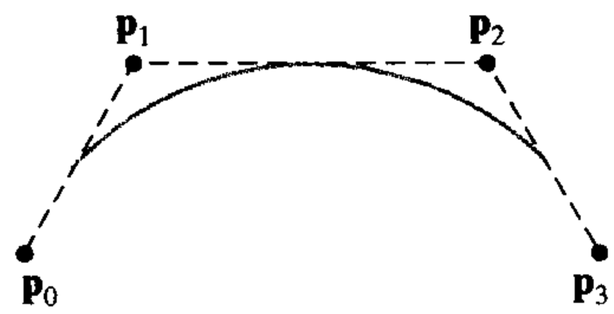


图 8.43　三次周期性 B 样条曲线拟合 xy 平面上的四个控制点

在上例中，注意到二次曲线起始于前两个控制点之间，结束于后两个控制点之间。对于任意数目的不同控制点拟合的二次周期性 B 样条，这一结果也是正确的。一般来说，对于高阶多项式，起点和终点是 d-1 个控制点的加权平均值点。只要多次指定某一位置，则样条曲线将更加接近该控制点的位置。

重新定义混合函数的参数，可以得到周期性 B 样条曲线边界条件的一般表达式。参数 u 映射到从 0 到 1 的单位区间上。只要取 $u=0$ 和 $u=1$ 即可获得起始和终止条件。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.12.3 三次周期性B样条曲线

由于三次周期性B样条曲线普遍用于图形软件包，因此我们考虑这类样条的公式。周期性样条对于生成某些封闭曲线特别有用。例如，图8.44中的封闭图形可以通过循环指定六个控制点中的四个控制点而分段获得。若三个连续控制点相重合，则曲线通过该坐标位置。

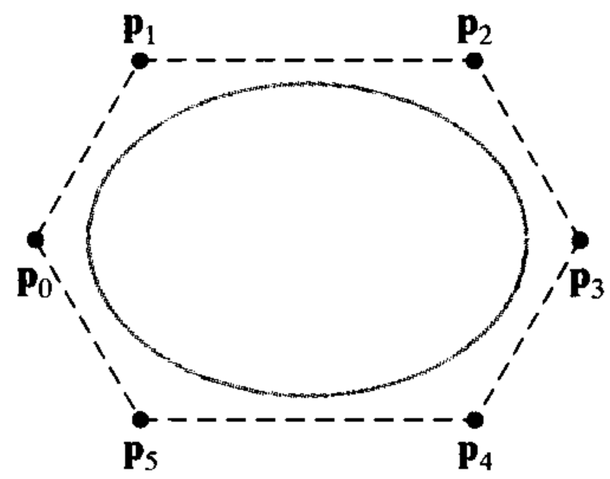


图 8.44 封闭、周期性、分段三次 B 样条，通过循环取六个控制点中的四个而得到

对于三次曲线，$d=4$，而且每个混合函数定义在 u 取值范围的 4 个子区间上。如果将三次曲线拟合在四个控制点上，则可以使用整数节点向量：

$$\{0, 1, 2, 3, 4, 5, 6, 7\}$$

类似于上一节中的二次周期性 B 样条，递归关系式（8.53）用来获得周期性混合函数。

在本节中，考虑周期性三次B样条的另一个公式。从边界条件开始入手，并将混合函数定义在区间 $0 \leqslant u \leqslant 1$。通过这一公式，也可以很容易地得到特征矩阵。周期性三次 B 样条在四个连续控制点 $\mathbf{p}_0$、$\mathbf{p}_1$、$\mathbf{p}_2$、$\mathbf{p}_3$ 上的边界条件是

$$\begin{aligned}
\mathbf{P}(0) &= \frac{1}{6}(\mathbf{p}_0 + 4\mathbf{p}_1 + \mathbf{p}_2) \\
\mathbf{P}(1) &= \frac{1}{6}(\mathbf{p}_1 + 4\mathbf{p}_2 + \mathbf{p}_3) \\
\mathbf{P}'(0) &= \frac{1}{2}(\mathbf{p}_2 - \mathbf{p}_0) \\
\mathbf{P}'(1) &= \frac{1}{2}(\mathbf{p}_3 - \mathbf{p}_1)
\end{aligned} \tag{8.56}$$

这些边界条件类似于cardinal样条中的边界条件：曲线段由四个控制点给出，在每个曲线段两个端点处的导数（斜率）平行于相邻控制点的连线。B 样条从靠近 $\mathbf{p}_1$ 点位置开始到靠近 $\mathbf{p}_2$ 处结束。

具有四个控制点的三次周期性 B 样条的矩阵公式可以写为

$$\mathbf{P}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \mathbf{M}_B \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \tag{8.57}$$

其中，周期性三次多项式的 B 样条矩阵是

$$\mathbf{M}_B = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{8.58}$$

使用给定的四个边界条件求解一个有关三次多项式系数的方程组，可以得到以上矩阵。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们也可以调整 B 样条方程，从而包含一个张量参数 t（类似于 cardinal 样条）。带有张量参数 t 的周期性三次 B 样条矩阵具有下列形式：

$$\mathbf{M}_{B_t} = \frac{1}{6}\begin{bmatrix} -t & 12-9t & 9t-12 & t \\ 3t & 12t-18 & 18-15t & 0 \\ -3t & 0 & 3t & 0 \\ t & 6-2t & t & 0 \end{bmatrix} \tag{8.59}$$

当 $t=1$ 时将得到 M_B。

将矩阵表达式写成多项式形式，可以得到参数取值为 0 到 1 的周期性三次 B 样条混合函数。例如，当张量值 $t=1$ 时，可以有

$$\begin{aligned} B_{0,3}(u) &= \frac{1}{6}(1-u)^3, \qquad 0 \leqslant u \leqslant 1 \\ B_{1,3}(u) &= \frac{1}{6}(3u^3 - 6u^2 + 4) \\ B_{2,3}(u) &= \frac{1}{6}(-3u^3 + 3u^2 + 3u + 1) \\ B_{3,3}(u) &= \frac{1}{6}u^3 \end{aligned} \tag{8.60}$$

8.12.4　开放均匀的 B 样条曲线

这类 B 样条是均匀的 B 样条曲线和非均匀的 B 样条曲线的重叠部分。有时将这类曲线看成特殊的均匀 B 样条类型，有时又看成属于非均匀 B 样条类型。**开放均匀**（open uniform）的 B 样条曲线，简称**开放**（open）B 样条曲线，除了在两端的节点值重复 d 次以外，其节点间距是均匀的。

下面是两个开放均匀的、从 0 开始的整型节点向量的例子：

$$\begin{aligned} \{0, 0, 1, 2, 3, 3\} \qquad &, d = 2 \text{ 和 } n = 3 \\ \{0, 0, 0, 0, 1, 2, 2, 2, 2\} \qquad &, d = 4 \text{ 和 } n = 4 \end{aligned} \tag{8.61}$$

我们将节点向量规范到从 0 到 1 的单位区间：

$$\begin{aligned} \{0, 0, 0.33, 0.67, 1, 1\} \qquad &, d = 2 \text{ 和 } n = 3 \\ \{0, 0, 0, 0, 0.5, 1, 1, 1, 1\} \qquad &, d = 4 \text{ 和 } n = 4 \end{aligned} \tag{8.62}$$

对于任意的参值 d 和 n，通过下列计算，可以生成开放均匀的具有整型节点的向量：

$$u_j = \begin{cases} 0 & , 0 \leqslant j < d \\ j - d + 1 & , d \leqslant j \leqslant n \\ n - d + 2 & , j > n \end{cases} \tag{8.63}$$

j 的值从 0 到 $n+d$。前 d 个节点向量值设为 0，后 d 个值设为 $n-d+2$。

开放均匀的 B 样条具有与 Bézier 样条非常类似的特性。事实上，当 $d=n+1$（多项式次数为 n）时，开放 B 样条变回到 Bézier 样条，所有的节点值为 0 或为 1。例如，三次开放 B 样条（$d=4$），具有 4 个控制点，则节点向量是

$$\{0, 0, 0, 0, 1, 1, 1, 1\}$$

开放B样条的多项式曲线通过第一个和最后一个控制点。参数曲线在第一个控制点处的斜率也平行于前两个控制点的连线。最后一个控制点处的斜率平行于最后两个控制点的连线。因此，连接曲线段的几何约束也与 Bézier 曲线相同。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

类似于Bézier曲线，在同一个坐标位置指定多个控制点会将B样条曲线拉向该点。由于开放B样条起始于第一控制点，终止于最后一个控制点，因此把第一个和最后一个控制点指定于同一位置将生成封闭曲线。

例 8.2　开放均匀的二次 B 样条

由条件（8.63），取 $d=3$、$n=4$（5 个控制点），可以得到下列节点向量的 8 个值：

$$\{u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7\} = \{0, 0, 0, 1, 2, 3, 3, 3\}$$

u 的整个取值范围分成 7 个子区间，且 5 个混合函数 $B_{k,3}$ 中的每一个函数从节点位置 u_k 开始定义在 3 个子区间上。这样，$B_{0,3}$ 定义在 $u_0=0$ 到 $u_3=1$ 上，$B_{1,3}$ 是在 $u_1=0$ 到 $u_4=2$ 上，$B_{4,3}$ 定义在 $u_4=2$ 到 $u_7=3$ 上。由递归关系式（8.53）得到混合函数的多项式表达式，如下所示：

$$B_{0,3}(u) = (1-u)^2 \qquad 0 \leqslant u < 1$$

$$B_{1,3}(u) = \begin{cases} \frac{1}{2}u(4-3u) & 0 \leqslant u < 1 \\ \frac{1}{2}(2-u)^2 & 1 \leqslant u < 2 \end{cases}$$

$$B_{2,3}(u) = \begin{cases} \frac{1}{2}u^2 & 0 \leqslant u < 1 \\ \frac{1}{2}u(2-u) + \frac{1}{2}(u-1)(3-u) & 1 \leqslant u < 2 \\ \frac{1}{2}(3-u)^2 & 2 \leqslant u < 3 \end{cases}$$

$$B_{3,3}(u) = \begin{cases} \frac{1}{2}(u-1)^2 & 1 \leqslant u < 2 \\ \frac{1}{2}(3-u)(3u-5) & 2 \leqslant u < 3 \end{cases}$$

$$B_{4,3}(u) = (u-2)^2 \qquad 2 \leqslant u < 3$$

图 8.45 表示了这 5 个混合函数的形状。再一次展示了 B 样条曲线的局部特征。混合函数 $B_{0,3}$ 仅在从 0 到 1 的子区间上取非零值，因此第一个控制点仅影响该区间上的曲线。同样，函数 $B_{4,3}$ 在 2 到 3 的区间以外取零值，最后一个控制点位置不影响曲线的开头和中间部分的形状。

开放 B 样条的矩阵公式不像在均匀的周期性 B 样条中那样容易得到。这是由于节点值在节点向量的开始和结束部分重复多次的缘故。

8.12.5　非均匀 B 样条曲线

对于这类样条，可以对节点向量的值和间距指定任何值。对非均匀B样条，可以选择多个内节点值并且在节点值之间选择不等的间距。例如

$$\{0, 1, 2, 3, 3, 4\}$$
$$\{0, 2, 2, 3, 3, 6\}$$
$$\{0, 0, 0, 1, 1, 3, 3, 3\}$$
$$\{0, 0.2, 0.6, 0.9, 1.0\}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

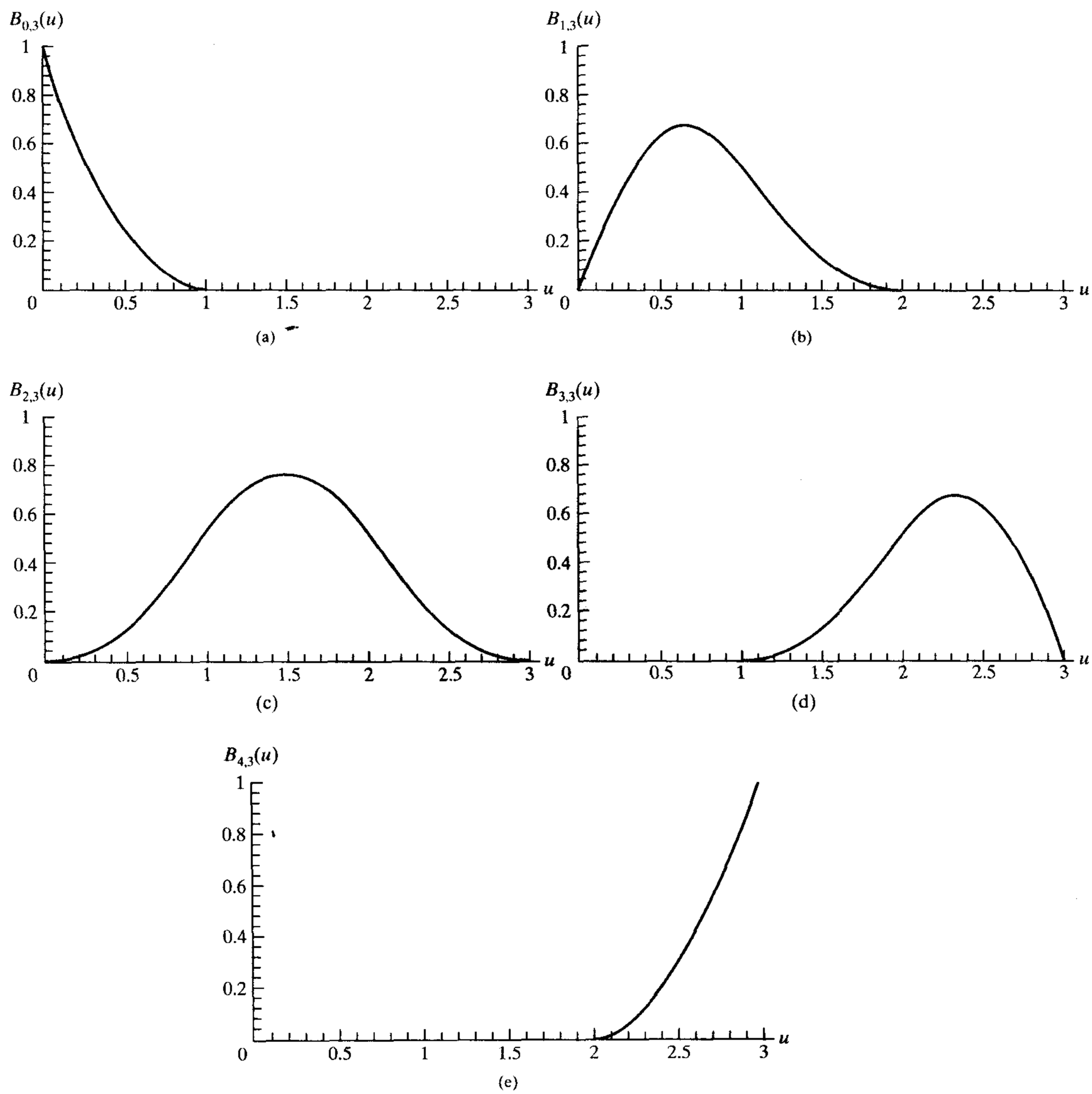


图 8.45　$n=4, d=3$ 的开放均匀 B 样条混合函数

非均匀B样条曲线在控制曲线形状方面提供了更多的便利。通过在节点向量中取不同的间距，我们可以在不同的区间上得到不同的混合函数形状，从而用来调整曲线形状。增加节点值的多样性，可以在曲线中产生细微的摆动，甚至导致不连续性。重节点值也可能因为每次重复一个特殊值而使连续性降一次。

使用类似于均匀和开放 B 样条曲线中使用的方法，可以得到非均匀 B 样条曲线的混合函数。给定 $n+1$ 个控制点集，设定多项式次数及选定节点值。然后使用递归关系，既可得到混合函数集，又可求解曲线位置从而直接显示曲线。图形软件包经常限制节点值的间距为 0 或为 1 以减少计算量。这样，可以存储特征矩阵集，以便用于计算样条曲线上的值而无需在绘制每个曲线点时求解递归关系。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.13 B样条曲面

B样条曲面公式类似于Bézier样条公式。使用如下所示的B样条混合函数的笛卡儿积，可以得到B样条曲面上的向量函数：

$$\mathbf{P}(u, v) = \sum_{k_u=0}^{n_u} \sum_{k_v=0}^{n_v} \mathbf{p}_{k_u,k_v} B_{k_u,d_u}(u) B_{k_v,d_v}(v) \tag{8.64}$$

其中，向量值 $\mathbf{p}_{k_u,k_v}$ 指定了$(n_u+1)\times(n_v+1)$个控制点位置。

B样条曲面具有与B样条曲线相同的性质。选定参数 d_u 和 d_v 的值，该值确定所使用的多项式的次数为 d_u-1 和 d_v-1。我们还为每一个曲面参数 u 和 v 选择节点向量，用于确定混合函数的参数范围。图8.46给出了一个用B样条曲面建模的对象。

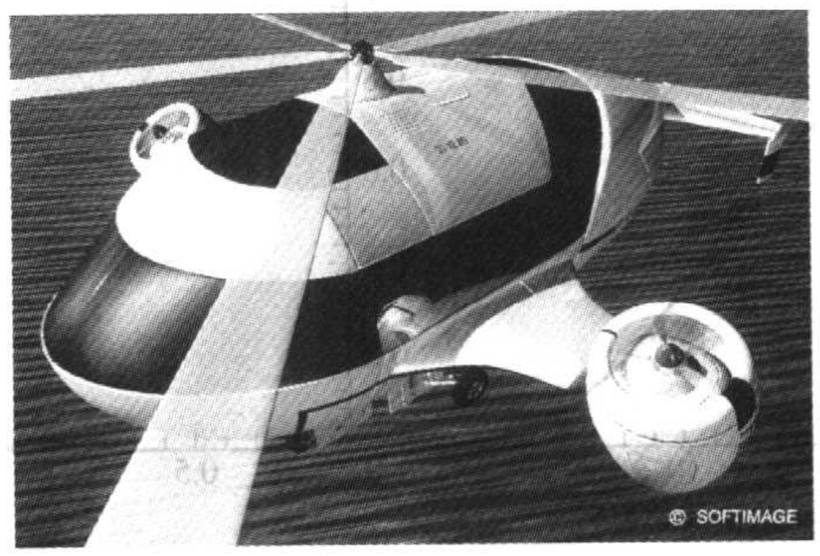


图8.46 一架原型直升飞机，由SOFTIMAGE公司的Daniel Langlois设计并进行模拟，使用了180 000个B样条曲面片。然后使用光线追踪、凸映射和反射映射方法绘制这一场景（SGI公司提供）

8.14 beta样条

B样条的一般化是beta样条，也记为**β样条**，它是在一阶和二阶导数上加上几何条件而形成的。beta样条的连续性参数称为 β 参数（β parameter）。

8.14.1 beta样条连续性条件

对于一个指定的节点向量，我们将一个确定节点 u_j 的左右样条段命名为位置向量 $\mathbf{P}_{j-1}(u)$ 和 $\mathbf{P}_j(u)$（参见图8.47）。在 u_j 处获得0阶连续性（位置连续性）G^0，需要：

$$\mathbf{P}_{j-1}(u_j) = \mathbf{P}_j(u_j) \tag{8.65}$$

取得一阶连续性（单位切向量连续）G^1，则需要切向量成比例：

$$\beta_1 \mathbf{P}'_{j-1}(u_j) = \mathbf{P}'_j(u_j), \qquad \beta_1 > 0 \tag{8.66}$$

其中，一阶导数成比例，单位切向量在节点处是连续的。

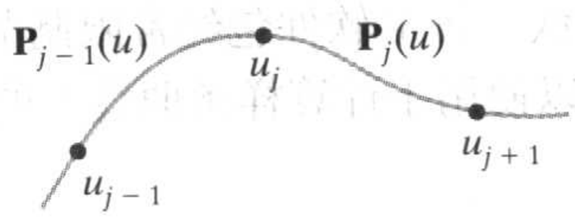


图8.47 曲面上到节点 u_j 的左右部分的位置向量

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

二阶连续性（曲率向量连续）G^2，需要满足条件：

$$\beta_1^2\mathbf{P}''_{j-1}(u_j)+\beta_2\mathbf{P}'_{j-1}(u_j)=\mathbf{P}''_j(u_j) \tag{8.67}$$

其中，β_2是给定的任意实数，且$\beta_1>0$。曲率向量给出了曲线在位置u_j上弯曲程度的度量。当$\beta_1=0$且$\beta_2=0$时，beta 样条退化到 B 样条。

参数β_1称为偏离参数，因为它控制曲线的倾斜度。当$\beta_1>1$，曲面偏向节点处单位切向量的右侧。当$0<\beta_1<1$时，曲线偏向节点处单位切向量的左侧。β_1对样条曲线形状的影响如图8.48所示。

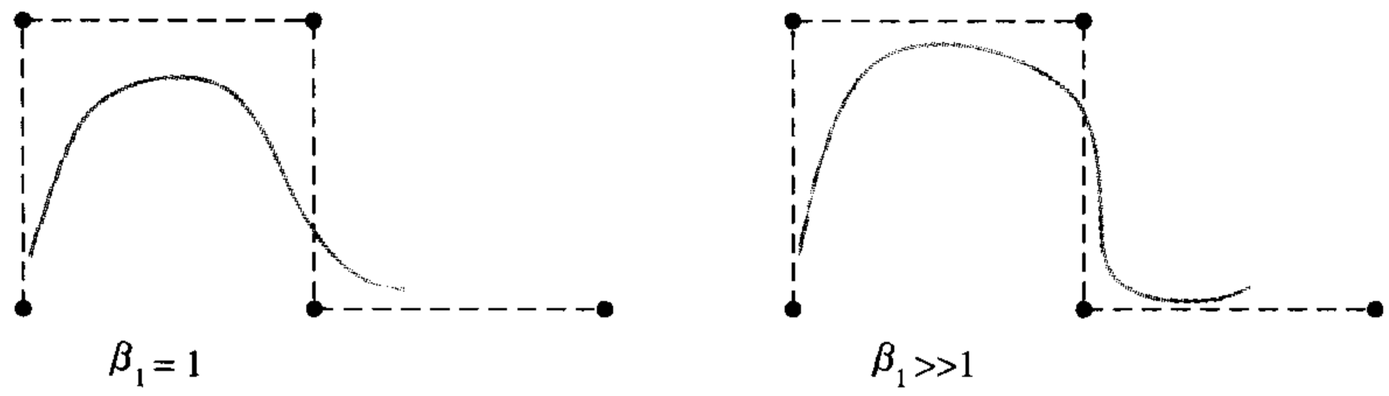


图 8.48　参数β_1对 beta 样条曲线的影响

参数β_2称为张量参数，因为其控制样条拟合控制图的松紧程度。当β_2增加时，曲线接近控制图的形状，如图 8.49 所示。

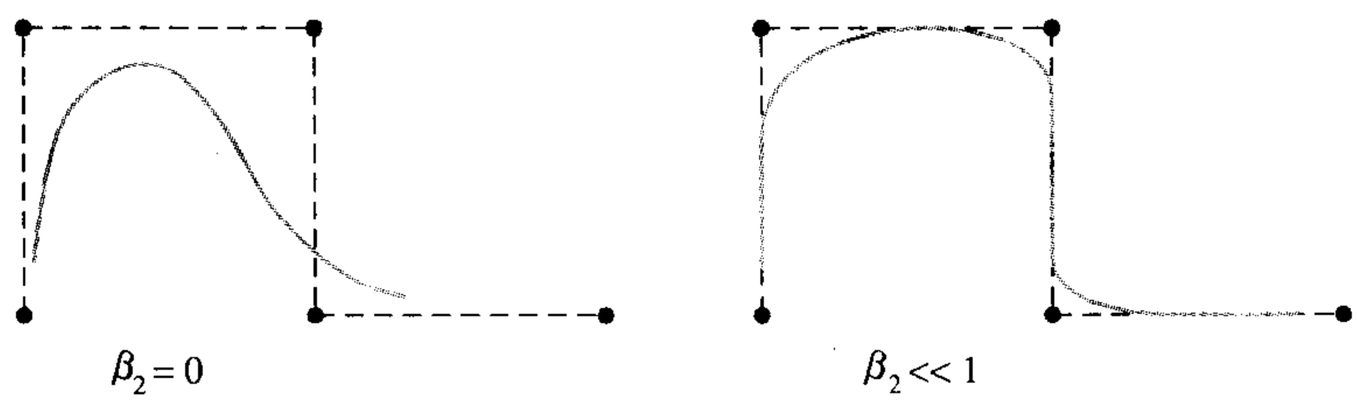


图 8.49　参数β_2对 beta 样条曲线的影响

8.14.2　三次周期性 beta 样条曲线的矩阵表示

在均匀节点向量的三次多项式上应用 beta 样条曲线的边界条件，可以得到下列周期性 beta 样条曲线的矩阵表示：

$$\mathbf{M}_\beta=\frac{1}{\delta}\begin{bmatrix}-2\beta_1^3 & 2(\beta_2+\beta_1^3+\beta_1^2+\beta_1) & -2(\beta_2+\beta_1^2+\beta_1+1) & 2\\ 6\beta_1^3 & -3(\beta_2+2\beta_1^3+2\beta_1^2) & 3(\beta_2+2\beta_1^2) & 0\\ -6\beta_1^3 & 6(\beta_1^3-\beta_1) & 6\beta_1 & 0\\ 2\beta_1^3 & \beta_2+4(\beta_1^2+\beta_1) & 2 & 0\end{bmatrix} \tag{8.68}$$

其中，$\delta=\beta_2+2\beta_1^3+4\beta_1^2+4\beta_1+2$。

当$\beta_1=1$且$\beta_2=0$时，得到 B 样条曲线矩阵M_B。并当

$$\beta_1=1,\qquad \beta_2=\frac{12}{t}(1-t)$$

时，得到 B 样条的张量矩阵M_{B_t}（参见等式 8.59）。

8.15　有理样条

有理函数是两个多项式之比。因此，**有理样条**（rational spline）是两个样条函数之比。例如，有理 B 样条曲线可以使用向量描述为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{P}(u)=\frac{\sum_{k=0}^{n}\omega_k\mathbf{p}_k B_{k,d}(u)}{\sum_{k=0}^{n}\omega_k B_{k,d}(u)} \tag{8.69}$$

其中，$\mathbf{p}_k$ 是 $n+1$ 个控制点位置。参数 ω_k 是控制点的加权因子。一个特定的 ω_k 值越大，曲线越靠近该控制点 $\mathbf{p}_k$。当所有加权因子都设为 1 时，得到标准 B 样条曲线，因为这时方程（8.69）中的分母仅为混和函数之和，即为 1（参见等式（8.54））。

有理样条与非有理样条相比有两个重要的优点。第一，提供了二次曲线的精确表达式，如圆和椭圆。非有理样条的表达式为多项式，仅能逼近二次曲线。这样，使得图形软件包使用一个表达式——有理样条——来模拟所有的曲线形状，无需使用一个曲线函数库来处理不同的设计形状。有理样条的另一个优点是，对于透视观察变换是不会变化的（参见7.8节）。这表示我们可以对有理曲线上的控制点应用一个透视观察变换，从而得到曲线的正确视图。而另一方面，非有理样条关于透视观察变换是可变的。通常，图形设计软件包使用非均匀节点向量表达式来构造有理B样条。这些样条称为 NURBS（nonuniform rational B-spline）或 NURB。

齐次坐标表达式用于有理样条，这是因为分母可以看成是在控制点四维表达式中的齐次因子h。这样，一个有理样条可以认为是将四维非有理样条投影到三维空间中。

构造有理B样条表达式与构造非有理表达式有着相同的步骤。给定控制点集、多项式次数、加权因子、节点向量，使用递归关系即可得到混合函数。在某些 CAD 系统中，可通过在一段弧上指定三个点来构造一个二次曲线段。然后，通过计算可生成所选二次曲线段的控制点位置来确定一个有理齐次坐标样条的表达式。

作为使用有理样条描述二次曲线段的一个例子，我们可以利用二次样条函数（$d=3$）、三个控制点及开放节点向量：

$$\{0,0,0,1,1,1\}$$

来实现。这种处理与二次 Bézier 样条一样。然后设置加权函数为下列值：

$$\begin{aligned}\omega_0&=\omega_2=1\\ \omega_1&=\frac{r}{1-r},\qquad 0\leqslant r<1\end{aligned} \tag{8.70}$$

有理 B 样条的表达式是

$$\mathbf{P}(u)=\frac{\mathbf{p}_0B_{0,3}(u)+[r/(1-r)]\mathbf{p}_1B_{1,3}(u)+\mathbf{p}_2B_{2,3}(u)}{B_{0,3}(u)+[r/(1-r)]B_{1,3}(u)+B_{2,3}(u)} \tag{8.71}$$

然后利用下列参数 r 值，得到各种二次曲线（参见图 8.50）：

$r>1/2,\ \omega_1>1$ 双曲线段
$r=1/2,\ \omega_1=1$ 抛物线段
$r<1/2,\ \omega_1<1$ 椭圆段
$r=0,\ \omega_1=0$ 直线段

设 $\omega_1=\cos\phi$，并选择控制点：

$$\mathbf{p}_0=(0,1),\qquad \mathbf{p}_1=(1,1),\qquad \mathbf{p}_2=(1,0)$$

可以产生 xy 平面上单位圆在第一象限中的弧，如图 8.51 所示。单位圆的其他部分可以根据不同控制点位置而得到。利用几何变换可以产生 xy 平面上的整个圆。例如，可以将 1/4 圆弧关于 x 和 y 轴反射，从而产生其他三个象限中的圆弧。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

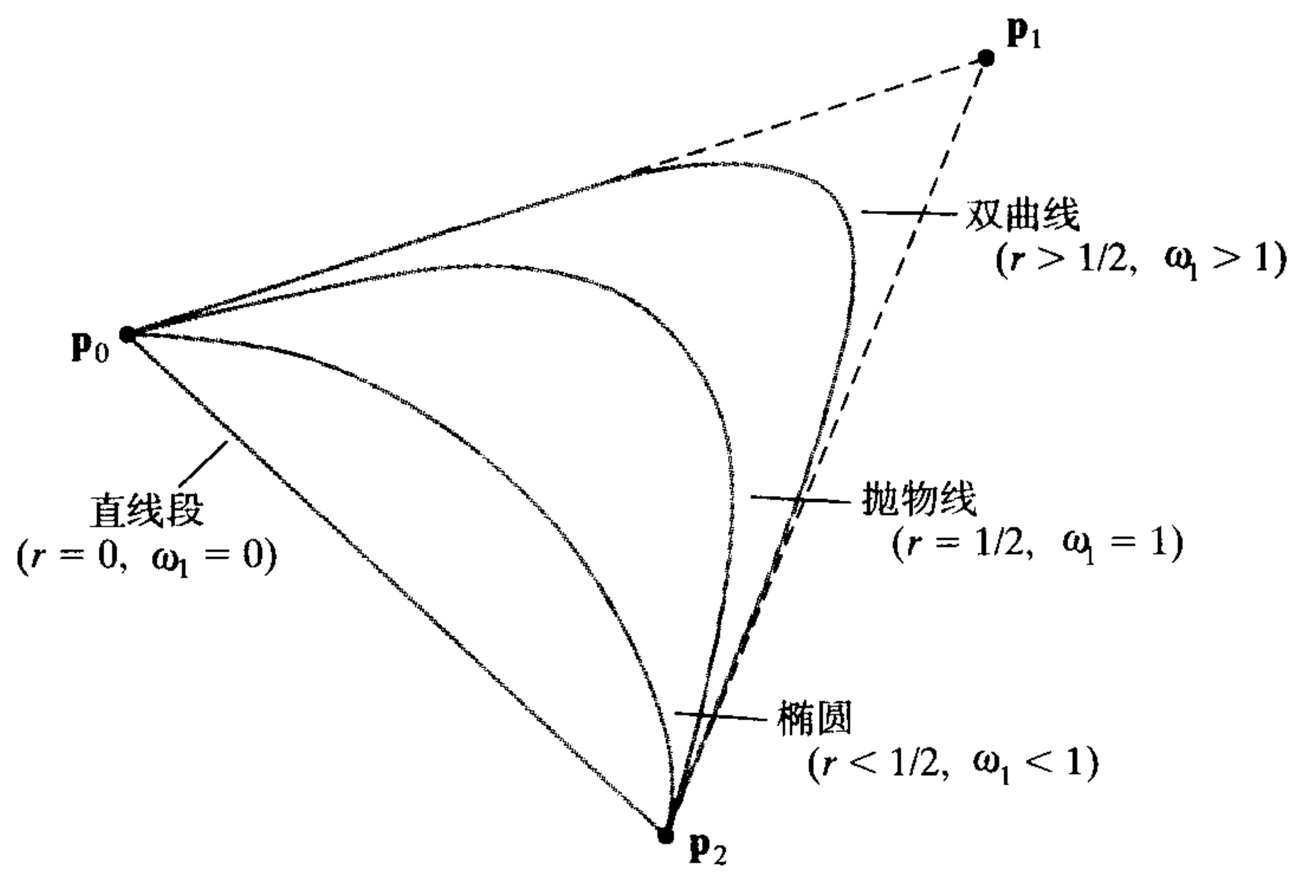


图 8.50　由不同有理样条的加权因子 ω_1 值生成的二次曲线段

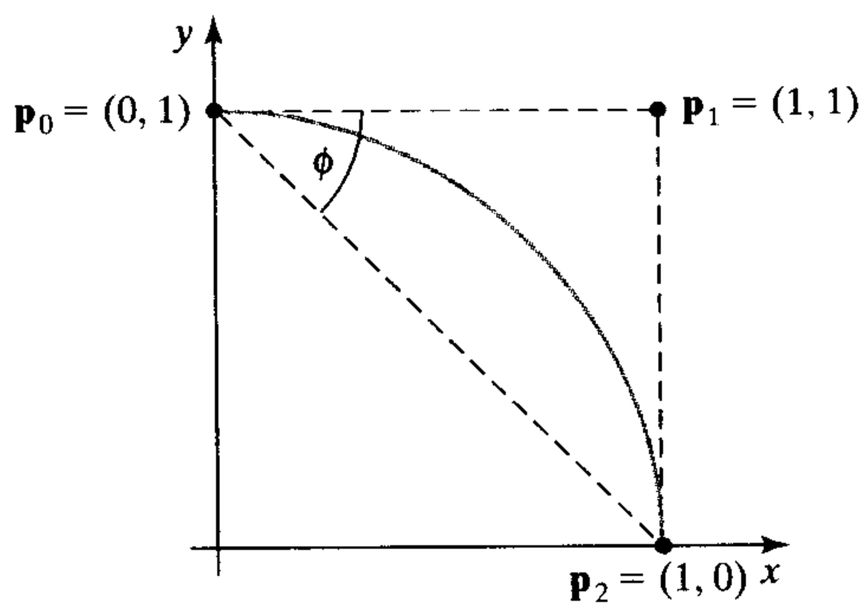


图 8.51　xy 平面中第一象限上的圆弧

在 xy 平面上第一象限中单位圆弧的齐次表达式是

$$\begin{bmatrix} x_h(u) \\ y_h(u) \\ z_h(u) \\ h(u) \end{bmatrix} = \begin{bmatrix} 1-u^2 \\ 2u \\ 0 \\ 1+u^2 \end{bmatrix} \tag{8.72}$$

该齐次表达式给出了如下的第一象限参数圆方程：

$$\begin{aligned} x &= \frac{x_h(u)}{h(u)} = \frac{1-u^2}{1+u^2} \\ y &= \frac{y_h(u)}{h(u)} = \frac{2u}{1+u^2} \end{aligned} \tag{8.73}$$

8.16　样条表示之间的转换

有时，希望能从一个样条表达式转变到另一个样条表达式。例如，Bézier 表达式对分割样条曲线是最方便的，而 B 样条表达式提供了更大的设计灵活性。因此，可以用 B 样条段来设计一条曲线，然后转换到等价的 Bézier 表达式，利用递归分割程序以对沿曲线的坐标位置定位，从而显示对象。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

假设一对象的样条描述可以表示成下列矩阵乘积：

$$\mathbf{P}(u) = \mathbf{U} \cdot \mathbf{M}_{\text{spline1}} \cdot \mathbf{M}_{\text{geom1}} \tag{8.74}$$

其中，$\mathbf{M}_{\text{spline1}}$ 是描述样条表达式的矩阵，$\mathbf{M}_{\text{geom1}}$ 是几何约束的列矩阵（如控制点坐标）。为了变换到具有样条矩阵 $\mathbf{M}_{\text{spline2}}$ 的第二个表达式，需要确定几何约束矩阵 $\mathbf{M}_{\text{geom2}}$，即

$$\mathbf{P}(u) = \mathbf{U} \cdot \mathbf{M}_{\text{spline2}} \cdot \mathbf{M}_{\text{geom2}} \tag{8.75}$$

或

$$\mathbf{U} \cdot \mathbf{M}_{\text{spline2}} \cdot \mathbf{M}_{\text{geom2}} = \mathbf{U} \cdot \mathbf{M}_{\text{spline1}} \cdot \mathbf{M}_{\text{geom1}} \tag{8.76}$$

求解 $\mathbf{M}_{\text{geom2}}$ 得到

$$\begin{aligned} \mathbf{M}_{\text{geom2}} &= \mathbf{M}_{\text{spline2}}^{-1} \cdot \mathbf{M}_{\text{spline1}} \cdot \mathbf{M}_{\text{geom1}} \\ &= \mathbf{M}_{s1,s2} \cdot \mathbf{M}_{\text{geom1}} \end{aligned} \tag{8.77}$$

从第一个样条表达式转变到第二个表达式的变换矩阵可以计算为

$$\mathbf{M}_{s1,s2} = \mathbf{M}_{\text{spline2}}^{-1} \cdot \mathbf{M}_{\text{spline1}} \tag{8.78}$$

非均匀 B 样条不能使用一般样条矩阵进行描述，但是可以再安排一个节点序列，将非均匀 B 样条转变到 Bézier 表达式。然后 Bézier 矩阵可以转变到任何其他形式。

下例计算了将周期性三次 B 样条表达式变换到三次 Bézier 样条表达式的变换矩阵：

$$\begin{aligned} \mathbf{M}_{B,\text{Bez}} &= \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}^{-1} \cdot \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{bmatrix} \end{aligned} \tag{8.79}$$

将三次 Bézier 表达式变换到周期性三次 B 样条表达式的变换矩阵是

$$\begin{aligned} \mathbf{M}_{\text{Bez},B} &= \begin{bmatrix} -\frac{1}{6} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{6} \\ \frac{1}{2} & -1 & \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 6 & -7 & 2 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 2 & -7 & 6 \end{bmatrix} \end{aligned} \tag{8.80}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.17　样条曲线和曲面的显示

为了显示一个样条曲线或曲面，必须确定将要投影到显示设备上像素位置的曲线或曲面上的坐标位置。这说明必须在函数值域内，以一定增量求参数多项式的样条函数的值。在样条曲线或曲面范围上，可以有几种方法用来计算位置。

8.17.1　Horner 规则

最简单的求多项式值的方法是 Horner 规则（Horner's rule）。该规则不是逐次计算每一项，而是通过逐次分解因子来进行计算。每一步需要进行一次乘法和一次加法运算。对于次数为 n 的多项式，共有 n 步计算。

假设有一个三次样条表达式，其坐标位置表示为

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x \tag{8.81}$$

对 y 和 z 坐标有类似的表达式。对于参数 u 的一个确定值，我们根据下列分解因子的顺序来求多项式的值：

$$x(u) = [(a_x u + b_x)u + c_x]u + d_x \tag{8.82}$$

每一个 x 值计算需要三次乘法和三次加法，因此沿三次样条曲线确定每个坐标位置 (x, y, z) 需要九次乘法和九次加法。

利用 Horner 规则，另外的分解因子的方法可以用来减少计算量，特别是对于高阶多项式（次数大于3）。但是对于样条函数值域上坐标位置的重复定位，使用向前差分计算方法或样条细分方法的计算速度会更快。

8.17.2　向前差分计算

求解多项式函数值的最快方法是，通过将一个增量加到前一次计算的值上，递归地生成后面的值，例如：

$$x_{k+1} = x_k + \Delta x_k \tag{8.83}$$

这样，一旦知道了任意一步的 x_k 值和增量，那么通过这一步的值加上增量就可以得到下一个值。每步的增量 Δx_k 称为向前差分（forward difference）。如果将 u 的取值范围分成具有固定大小 δ 的子区间，则两个相邻 x 坐标为 $x_k = x(u_k)$ 和 $x_{k+1} = x(u_{k+1})$，其中

$$u_{k+1} = u_k + \delta, \qquad k = 0, 1, 2, \ldots \tag{8.84}$$

并且 $u_0 = 0$。

为了说明这种方法，假设有线性样条表达式 $x(u) = a_x u + b_x$，两个相邻的 x 坐标位置表示为

$$\begin{aligned} x_k &= a_x u_k + b_x \\ x_{k+1} &= a_x(u_k + \delta) + b_x \end{aligned} \tag{8.85}$$

两方程相减得到向前差分：

$$\Delta x_k = x_{k+1} - x_k = a_x \delta \tag{8.86}$$

在这种情况下，该向前差分是一个常数。对于高阶多项式，向前差分本身是参数 u 的多项式函数，其次数比原多项式小 1。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于方程（8.81）中的三次样条表达式，两个相邻 x 坐标具有多项式表达式：

$$\begin{aligned} x_k &= a_x u_k^3 + b_x u_k^2 + c_x u_k + d_x \\ x_{k+1} &= a_x(u_k+\delta)^3 + b_x(u_k+\delta)^2 + c_x(u_k+\delta) + d_x \end{aligned} \tag{8.87}$$

可以计算向前差分：

$$\Delta x_k = 3a_x\delta u_k^2 + (3a_x\delta^2 + 2b_x\delta)u_k + (a_x\delta^3 + b_x\delta^2 + c_x\delta) \tag{8.88}$$

这是参数 u_k 的二次函数。由于 Δx_k 是 u 的多项式函数，所以可以使用同样的增量过程来得到 Δx_k 的后继值。即

$$\Delta x_{k+1} = \Delta x_k + \Delta_2 x_k \tag{8.89}$$

其中第二个向前差分是线性函数：

$$\Delta_2 x_k = 6a_x\delta^2 u_k + 6a_x\delta^3 + 2b_x\delta^2 \tag{8.90}$$

再次重复这个过程，记为

$$\Delta_2 x_{k+1} = \Delta_2 x_k + \Delta_3 x_k \tag{8.91}$$

其中第三个向前差分是常数：

$$\Delta_3 x_k = 6a_x\delta^3 \tag{8.92}$$

方程（8.83）、（8.89）、（8.91）和（8.92）给出了沿三次曲线的点的递增向前差分计算。从 $u_0 = 0$ 开始，每步长为 δ，得到 x 坐标的初始值以及第一组的两个向前差分：

$$\begin{aligned} x_0 &= d_x \\ \Delta x_0 &= a_x\delta^3 + b_x\delta^2 + c_x\delta \\ \Delta_2 x_0 &= 6a_x\delta^3 + 2b_x\delta^2 \end{aligned} \tag{8.93}$$

一旦算出初始值，则每个后继 x 坐标位置的计算仅需三次加法。

我们运用向前差分方法来对次数为n的曲线上的点进行定位。每一后继坐标位置(x, y, z)通过$3n$次加法序列进行计算。对于曲面而言，增量计算既应用于参数 u 又应用于参数 v。

8.17.3 细分方法

递归样条细分（recursive spline-subdivision）过程可以用来反复地平分曲线段，每一步都将增加控制点的数目。细分方法对于显示逼近样条十分有用，因为可以重复细分过程直到控制图逼近曲线路径。然后将控制点坐标作为曲线位置而绘出整个图形。细分的另一个作用是为绘制曲线造形而生成更多的控制点。这样可以使用少量的控制点来设计曲线形状，然后利用细分过程以得到附加的控制点。利用附加的控制点，可以对曲线的某些小段进行精细的调整。

样条细分方法最容易应用于Bézier曲线，因为该曲线通过第一个和最后一个控制点，并且参数 u 的范围总是在 0 和 1 之间，比较容易确定何时控制点“充分靠近”曲线路径。通过下列操作，可以将 Bézier 细分应用到其他样条表达式：

(1) 将所用的样条表达式转换到 Bézier 表达式；
(2) 应用 Bézier 细分算法；
(3) 再将 Bézier 表达式变回到原来的样条表达式。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图 8.52 表示了三次 Bézier 曲线段中递归细分的第一步。Bézier 曲线使用参数点函数 $\mathbf{P}(u)$ $(0 \leqslant u \leqslant 1)$进行描述。在第一步细分中，利用中点 $\mathbf{P}(0.5)$将原曲线分成两部分。这时，第一段函数记为 $\mathbf{P}_1(s)$，第二段函数记为 $\mathbf{P}_2(t)$。其中

$$\begin{aligned} s &= 2u, \qquad 0.0 \leqslant u \leqslant 0.5 \\ t &= 2u - 1, \qquad 0.5 \leqslant u \leqslant 1.0 \end{aligned} \tag{8.94}$$

两条新曲线段与原始曲线段具有相同数目的控制点。在每个新曲线段中，两个端点处的边界条件（位置和斜率）也必与原曲线 $\mathbf{P}(u)$的位置和斜率相同。这为我们确定每个曲线段的控制点位置给出了四个条件。对于第一部分曲线，这四个新控制点是

$$\begin{aligned} \mathbf{p}_{1,0} &= \mathbf{p}_0 \\ \mathbf{p}_{1,1} &= \frac{1}{2}(\mathbf{p}_0 + \mathbf{p}_1) \\ \mathbf{p}_{1,2} &= \frac{1}{4}(\mathbf{p}_0 + 2\mathbf{p}_1 + \mathbf{p}_2) \\ \mathbf{p}_{1,3} &= \frac{1}{8}(\mathbf{p}_0 + 3\mathbf{p}_1 + 3\mathbf{p}_2 + \mathbf{p}_3) \end{aligned} \tag{8.95}$$

对于后半个曲线，这四个控制点为

$$\begin{aligned} \mathbf{p}_{2,0} &= \frac{1}{8}(\mathbf{p}_0 + 3\mathbf{p}_1 + 3\mathbf{p}_2 + \mathbf{p}_3) \\ \mathbf{p}_{2,1} &= \frac{1}{4}(\mathbf{p}_1 + 2\mathbf{p}_2 + \mathbf{p}_3) \\ \mathbf{p}_{2,2} &= \frac{1}{2}(\mathbf{p}_2 + \mathbf{p}_3) \\ \mathbf{p}_{2,3} &= \mathbf{p}_3 \end{aligned} \tag{8.96}$$

计算新控制点仅需要加法和移位（除以 2）操作：

$$\begin{aligned} \mathbf{p}_{1,0} &= \mathbf{p}_0 \\ \mathbf{p}_{1,1} &= \frac{1}{2}(\mathbf{p}_0 + \mathbf{p}_1) \\ \mathbf{T} &= \frac{1}{2}(\mathbf{p}_1 + \mathbf{p}_2) \\ \mathbf{p}_{1,2} &= \frac{1}{2}(\mathbf{p}_{1,1} + \mathbf{T}) \\ \mathbf{p}_{2,3} &= \mathbf{p}_3 \\ \mathbf{p}_{2,2} &= \frac{1}{2}(\mathbf{p}_2 + \mathbf{p}_3) \\ \mathbf{p}_{2,1} &= \frac{1}{2}(\mathbf{T} + \mathbf{p}_{2,2}) \\ \mathbf{p}_{2,0} &= \frac{1}{2}(\mathbf{p}_{1,2} + \mathbf{p}_{2,1}) \\ \mathbf{p}_{1,3} &= \mathbf{p}_{2,0} \end{aligned} \tag{8.97}$$

这些步骤可以重复多次，取决于是否需要进一步细分曲线以得到更多的控制点，或取决于是否要定位逼近曲线。一旦通过细分得到了一组显示点，则当曲线段很小时，可以结束细分过程。决定

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

结束细分过程的一种方法是，检查每段中第一个和最后一个控制点之间的距离。若这些距离“充分”小，我们可以停止细分。另一种测试是检查相邻控制点的距离。或者当每段的控制点集几乎共线时，可以停止细分过程。

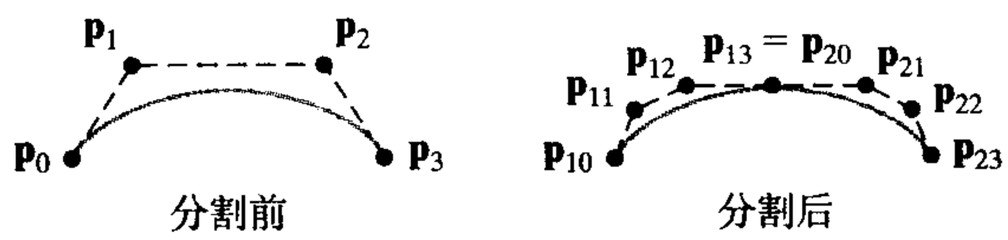


图 8.52 将一条 Bézier 曲线细分成两条，每条有四个控制点

细分方法可以应用于任意次数的Bézier曲线。对于次数为$n-1$的Bézier多项式，第一次细分步骤后，每一对初始曲线上的$2n$个控制点是

$$\mathbf{p}_{1,k} = \frac{1}{2^k}\sum_{j=0}^{k} C(k, j)\mathbf{p}_j, \qquad k = 0, 1, 2, \ldots, n$$
$$\mathbf{p}_{2,k} = \frac{1}{2^{n-k}}\sum_{j=k}^{n} C(n-k, n-j)\mathbf{p}_j \tag{8.98}$$

其中，$C(k, j)$和$C(n-k, n-j)$是二项式系数。

可以通过增加节点向量值将细分方法直接用于非均匀B样条。但在一般情况下，这些方法不像Bézier细分那样有效。

8.18 OpenGL的逼近样条函数

使用OpenGL的函数可显示Bézier样条和B样条，以及样条曲面的修剪曲线。核心库包括Bézier函数，而OpenGL实用库（GLU）包括B样条和修剪曲线函数。Bézier函数常用硬件实现，而GLU函数提供一个调用OpenGL画点和画线子程序的B样条接口。

8.18.1 OpenGL的Bézier样条曲线函数

下列OpenGL函数用来指定参数并激活Bézier曲线的显示子程序：

```
glMap1* (GL_MAP1_VERTEX_3, uMin, uMax, stride, nPts, *ctrlPts);
glEnable (GL_MAP1_VERTEX_3);
```

而使该子程序无效的函数是

```
glDisable (GL_MAP1_VERTEX_3);
```

glMap1的后缀码f或d用来指出数值的浮点类型或双精度浮点类型。曲线参数u的最小或最大值在uMin和uMax中指定Bézier曲线，尽管Bézier曲线中的这两个值一般分别置为0和1.0。Bézier曲线控制点的三维浮点笛卡儿坐标值在数组ctrlPts中列出，该数组元素的数量在参数nPts中以正整数给出。参数stride赋以整型位移量，用来指出在数组ctrlPts中一个坐标位置的开始到下一个坐标位置开始的数据值的数量。对于一个三维控制点列表，可设定stride = 3。如果用四维齐次坐标指定控制点或在坐标值之间插入颜色等其他数据，则stride要赋以更高的值。要在四维齐次坐标系(x, y, z, h)表达控制点，仅需改变stride的值并将glMap1和glEnable中的符号常量改为GL_MAP1_VERTEX_4。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在设定 Bézier 参数并激活曲线的生成子程序后，要计算沿样条路径的位置并显示计算出的曲线。沿曲线路径的位置用下列函数计算：

```
glEvalCoord1* (uValue);
```

其中参数 uValue 赋以从 uMin 到 uMax 之间的某个区间值。该函数的后缀可以为 f 或 d，也可为 v 表示该值在一个数组中给出。函数 glEvalCoord1 使用等式（8.37）及下列参数值计算坐标位置：

$$u = \frac{u_{\text{value}} - u_{\text{min}}}{u_{\text{max}} - u_{\text{min}}} \tag{8.99}$$

该公式将 uValue 映射到 0 至 1.0 的区间内。

当 glEvalCoord1 处理曲线参数 u 的值时，生成一个 glVertex3 函数。为了得到一条 Bézier 曲线，需重复引用 glEvalCoord1 函数并使用 uMin 到 uMax 之间的选定数值来生成曲线路径上的一组点。使用直线段连接这些点，就可以利用一条折线逼近该样条曲线。

作为 OpenGL 的 Bézier 曲线子程序示例，下列程序使用 8.10 节中程序的四个控制点来生成二维三次 Bézier 曲线。在本例中，画出了曲线路径上的 50 个点，并且用直线段连接曲线点。该曲线用蓝色显示，其控制点用尺寸为 5 的红色点绘出（参见图 8.53）。

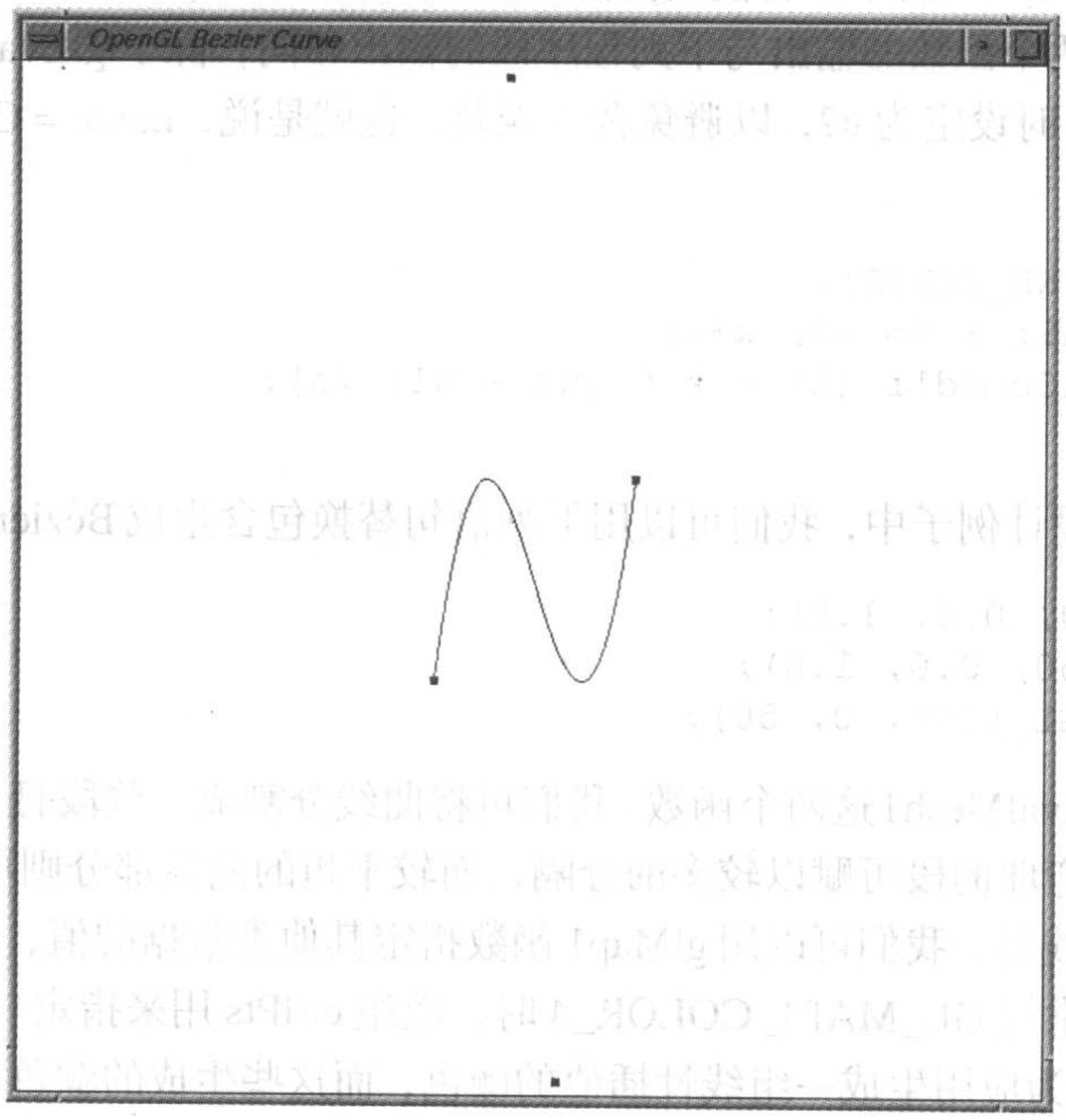


图 8.53　利用 OpenGL 子程序按逼近折线显示的四个控制点生成的 Bézier 曲线

```
GLfloat ctrlPts [4][3] = { {-40.0, 40.0, 0.0}, {-10.0, 200.0, 0.0},
                           {10.0, -200.0, 0.0}, {40.0, 40.0, 0.0} };

glMap1f (GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, *ctrlPts);
glEnable (GL_MAP1_VERTEX_3);

GLint k;

glColor3f (0.0, 0.0, 1.0);          //  Set line color to blue.
glBegin (GL_LINE_STRIP);            //  Generate Bezier "curve".
    for (k = 0; k <= 50; k++)
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        glEvalCoord1f (GLfloat (k) / 50.0);
glEnd ( );


glColor (1.0, 0.0, 0.0);                    //  Set point color to red.
glPointSize (5.0);                          //  Set point size to 5.0.
glBegin (GL_POINTS);                        //  Plot control points.
    for (k = 0; k < 4; k++);
            glVertex3fv (&ctrlPts [k][0]);
glEnd ( );
```

尽管前面的程序使用平均间隔的参数值生成了一条样条曲线，我们可使用glEvalCoord1f函数来对参数u任意安排。然而，样条曲线一般都平均安排参数间隔，而OpenGL提供下列函数生成一组均匀分布的参数值。

```
glMapGrid1* (n, u1, u2);
glEvalMesh1 (mode, n1, n2);
```

glMapGrid1的后缀可以是f或d。参数n指定从u1到u2之间均匀划分的整数量。参数mode赋以GL_POINT或GL_LINE，取决于是否使用离散的点（点状曲线）或直线段来显示曲线。对一条用折线显示的曲线，这两个函数的输出与下列程序段的输出相同，除了glEvalCoord1的变量在$k=0$时设定为u1或在$k=n$时设定为u2，以避免舍入误差。也就是说，mode = GL_LINE时，前面的命令与下列程序等价：

```
glBegin (GL_LINE_STRIP);
    for (k = n1; k <= n2; k++)
        glEvalCoord1f (u1 + k * (u2 - u1) /n);
glEnd ( );
```

因此，在前面的程序设计例子中，我们可以用下列语句替换包含生成Bézier曲线的循环的程序块。

```
glColor3f (0.0, 0.0, 1.0);
glMapGrid1f (50, 0.0, 1.0);
glEvalMesh1 (GL_LINE, 0, 50);
```

使用glMapGrid1和glEvalMesh1这两个函数，我们可将曲线分割成一些段且为每段按其曲率指定参数间隔。因此，比较弯曲的段可赋以较多的分隔，而较平坦的曲线部分则可赋以较少的分隔。

除显示Bézier曲线外，我们可以用glMap1函数指定其他类数据的值，为此，可使用另外七个符号常量。使用符号常量GL_MAP1_COLOR_4时，数组ctrlPts用来指定一个四元颜色（红、绿、蓝、α）列表。然后可为应用生成一组线性插值的颜色，而这些生成的颜色不改变当前的颜色状态设定。类似地，可以使用GL_MAP1_INDEX从一张颜色索引表中指定一组值。而使用符号常量GL_MAP1_NORMAL可在数组ctrlPts中指定一组三维的曲面法向量。其余四个符号常量用于指定一组表面纹理信息。

多个glMap1函数可同时被激活，对glEvalCoord1或glMapGrid1以及glEvalMesh1的调用为激活的每一类数据生成数据点。这使我们可以混合地生成坐标位置、颜色值、曲面法向量和曲面纹理等数据。但是不能够同时激活GL_MAP1_VERTEX_3和GL_MAP1_VERTEX_4，并且在同一时刻只能激活一个曲面纹理生成器。

8.18.2 OpenGL的Bézier样条曲面函数

OpenGL中Bézier样条曲面子程序的激活和参数设定使用下列语句来实现：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glMap2* (GL_MAP2_VERTEX_3, uMin, uMax, uStride, nuPts,
                 vMin, vMax, vStride, nvPts, *ctrlPts);
glEnable (GL_MAP2_VERTEX_3);
```

glMap2 的后缀 f 或 d 用来指定数据类型是浮点数或双精度浮点数。对一个曲面，需为 u 和 v 都指定最小值和最大值。双精度浮点数组 ctrlPts 给出 Bézier 控制点的三维笛卡儿坐标，而数组的整数尺寸在参数 nuPts 和 nvPts 中给出。如果要在四维齐次坐标中指定控制点，则需使用符号常量 GL_MAP2_VERTEX_4 而不是 GL_MAP2_VERTEX_3。控制点 $\mathbf{p}_{j,k}$ 的坐标值的开始和 $\mathbf{p}_{j+1,k}$ 坐标值的开始之间的整数位移在 uStride 中指定。而控制点 $\mathbf{p}_{j,k}$ 的坐标值的开始和 $\mathbf{p}_{j,k+1}$ 坐标值的开始之间的整数位移在 vStride 中指定。这样就允许在坐标数据中可插入其他数据，因此我们需要指定位移来定位坐标值。使用下列函数可使 Bézier 曲面子程序变为无效：

```
glDisable {GL_MAP2_VERTEX_3}
```

Bézier 曲面上的坐标位置可用下列函数计算：

```
glEvalCoord2* (uValue, vValue);
```

或

```
glEvalCoord2*v (uvArray);
```

参数 uValue 赋以 uMin 到 uMax 之间的某个值，而参数 vValue 赋以 vMin 到 vMax 之间的某个值。使用向量版本时，uvArray = (uValue, vValue)。两个函数的后缀码可为 f 或 d。函数 glEvalCoord2 使用等式（8.51）及下列参数值计算一个坐标位置：

$$u = \frac{u\text{Value} - u\text{Min}}{u\text{Max} - u\text{Min}}, \qquad v = \frac{v\text{Value} - v\text{Min}}{v\text{Max} - v\text{Min}} \tag{8.100}$$

它将每一个 uValue 和 vValue 映射到 0 到 1.0 的区间上。

为了显示一个 Bézier 曲面，我们反复引用 glEvalCoord2，以生成一系列 glVertex3 函数。这与生成单一样条曲线的情况类似，除了现在有两个参数，即 u 和 v。例如，用按 4 × 4 排列的 16 个控制点可用下列程序显示一组曲面线。u 方向的坐标值位移量是 3，而 v 方向的坐标值位移量是 12。每一坐标位置用三个值描述，而每一组四个位置的 y 坐标是不变的。

```
GLfloat ctrlPts [4][4][3] = {
   { {-1.5, -1.5,  4.0}, {-0.5, -1.5,  2.0},
     {-0.5, -1.5, -1.0}, { 1.5, -1.5,  2.0} },
   { {-1.5, -0.5,  1.0}, {-0.5, -0.5,  3.0},
     { 0.5, -0.5,  0.0}, { 1.5, -0.5, -1.0} },
   { {-1.5,  0.5,  4.0}, {-0.5,  0.5,  0.0},
     { 0.5,  0.5,  3.0}, { 1.5,  0.5,  4.0} },
   { {-1.5,  1.5, -2.0}, {-0.5,  1.5, -2.0},
     { 0.5,  1.5,  0.0}, { 1.5,  1.5, -1.0} }
};

glMap2f (GL_MAP2_VERTEX_3, 0.0, 1.0, 3, 4,
              0.0, 1.0, 12, 4, &ctrlPts[0][0][0]);
glEnable (GL_MAP2_VERTEX_3);

GLint k, j;

glColor3f (0.0, 0.0, 1.0);
for (k = 0; k <= 8; k++)
{
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    glBegin (GL_LINE_STRIP);  // Generate Bezier surface lines.
    for (j = 0; j <= 40; j++)
        glEvalCoord2f (GLfloat (j) / 40.0, GLfloat (k) / 8.0);
    glEnd ( );
    glBegin (GL_LINE_STRIP);
    for (j = 0; j <= 40; j++)
        glEvalCoord2f (GLfloat (k) / 8.0, GLfloat (j) / 40.0);
    glEnd ( );
}
```

除了使用 glEvalCoord2 函数之外，还可以使用下列语句生成曲面上的均匀间隔参数值：

```
glMapGrid2* (nu, u1, u2, nv, v1, v2);
glEvalMesh2 (mode, nu1, nu2, nv1, nv2);
```

glMapGrid2 的后缀还是 f 或 d，参数 mode 可赋以 GL_POINT、GL_LINE 或 GL_FILL。二维点网格按从 u1 到 u2 均匀地分为 nu 个间隔、从 v1 到 v2 均匀地分为 nv 个间隔来生成。参数 u 的对应范围为从 nu1 到 nu2，而参数 v 的对应范围为从 nv1 到 nv2。

对于一个要用折线网显示的曲面来说，glMapGrid2 和 glEvalMesh2 的输出与下面的程序一样，循环变量开始值和结束值处避免舍入误差的条件除外。在循环的开始，glEvalCoord1 的变量设定为(u1, v1)。而在循环的结束，glEvalCoord1 的变量设定为(u2, v2)。

```
for (k = nu1; k <= nu2; k++) {
    glBegin (GL_LINES);
        for (j = nv1; j <= nv2; j++)
           glEvalCoord2f (u1 + k * (u2 - u1) / nu,
                          v1 + j * (v2 - v1) / nv);
    glEnd ( );
}
for (j = nv1; j <= nv2; j++) {
    glBegin (GL_LINES);
        for (k = nu1; k <= nu2; k++)
           glEvalCoord2f (u1 + k * (u2 - u1) / nu,
                          v1 + j * (v2 - v1) / nv);
    glEnd ( );
}
```

类似地，对一个要用填充多边形面片（模式 = GL_FILL）显示的曲面来说，glMapGrid2 和 glEvalMesh2 的输出和下面的程序一样，循环变量开始值和结束值处避免舍入误差的条件除外。

```
for (k = nu1; k < nu2; k++) {
    glBegin (GL_QUAD_STRIP);
        for (j = nv1; j <= nv2; j ++) {
           glEvalCoord2f (u1 + k * (u2 - u1) / nu,
                          v1 + j * (v2 - v1) / nv);
           glEvalCoord2f (u1 + (k + 1) * (u2 - u1) / nu,
                          v1 + j * (v2 - v1) / nv);
```

我们可以用 glMap2 函数来指定其他类型数据，如同 glMap1 一样。为此，可类似地使用 GL_MAP2_COLOR_4 和 GL_MAP2_NORMAL 等符号常量。并且可以激活多个 glMap2 来生成多数据组合。

8.18.3 GLU 的 B 样条曲线函数

尽管 GLU 的 B 样条子程序称为“nurbs”函数，但它可用来生成既不是非均匀也不是有理的 B 样条。因此我们可以使用这些 GLU 子程序显示具有均匀节点间隔的多项式 B 样条。GLU 子程序还

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

可以用来生成 Bézier 样条，有理或非有理的。为了生成一个 B 样条（或 Bézier 样条），必须先定义该样条的名字，激活 GLU 的 B 样条绘制器，然后定义样条参数。

下面的语句给出了显示一条 B 样条曲线所需的基本调用序列。

```
GLUnurbsObj *curveName;

curveName = gluNewNurbsRenderer ( );
gluBeginCurve (curveName);
    gluNurbsCurve (curveName, nknots, *knotVector, stride, *ctrlPts,
                   degParam, GL_MAP1_VERTEX_3);
gluEndCurve (curveName);
```

在第一条语句中，我们为曲线定义一个名字，然后使用 gluNewNurbsRenderer 命令为该曲线引用 GLU 的 B 样条绘制子程序。如果没有足够的存储容量来创建一条 B 样条曲线，则 curveName 赋值为 0。在函数对 gluBeginCurve/gluEndCurve 之间，使用一个 gluNurbsCurve 函数来说明曲线的属性。这使我们能够建立多个曲线段，每段有不同的曲线名字。参数 knotVector 指定该组浮点类型的节点值，而整型的参数 nknots 指定节点向量中的元素数量。多项式的次数为 degParam−1。包含有 nknots−degParam 元素的数组变量 ctrlPts 给出三维控制点的位置。数组 ctrlPts 中连续坐标位置的开始处之间的位移量在整型参数 stride 中给出。如果控制点位置是连续的（不与其他类数据相间），则 stride 的值设定为 3。如果要消除一个已定义的 B 样条，使用

```
gluDeleteNurbsRenderer (curveName);
```

作为使用 GLU 子程序显示样条曲线的例子，下列程序生成一个三次 Bézier 多项式。为了获得这条三次曲线，设次数参数的值为 4。使用四个控制点，并选定一个 8 元素、开放均匀节点序列，其两端各有 4 个重复值。

```
GLfloat knotVector [8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0};
GLfloat ctrlPts [4][3] = { {-4.0, 0.0, 0.0}, {-2.0, 8.0, 0.0},
                           {2.0, -8.0, 0.0}, {4.0, 0.0, 0.0} };
GLUnurbsObj *cubicBezCurve;

cubicBezCurve = gluNewNurbsRenderer ( );
gluBeginCurve (cubicBezCurve);
    gluNurbsCurve (cubicBezCurve, 8, knotVector, 3, &ctrlPts [0][0],
                   4, GL_MAP1_VERTEX_3);
gluEndCurve(cubicBezCurve);
```

如果要建立一条有理 B 样条曲线，需用 GL_MAP1_VERTEX_4 代替 GL_MAP1_VERTEX_3。然后使用四维齐次坐标（x_h, y_h, z_h, h）指定控制点，结果中的齐次部分生成所要的齐次多项式形式。

我们也可以用 gluNurbsCurve 函数指定一组颜色值、法向量或曲面纹理特性，如在使用 glMap1 和 glMap2 中的一样。任一符号常量，如 GL_MAP1_COLOR_4 或 GL_MAP1_NORMAL，都可以用做 gluNurbsCurve 函数的最后一个参数。每一次调用都在函数对 gluBeginCurve/ gluEndCurve 之间列出，但有两个限制：每一数据类型不能有多个函数且至少包含一个生成 B 样条曲线的函数。

GLU 子程序将 B 样条曲线自动分成一些段并将它显示成折线。但通过对下列 GLU 函数的反复调用可选择 B 样条绘制器的各种选项。

```
gluNurbsProperty (splineName, property, value);
```

参数 splineName 赋以一条 B 样条曲线的名字，参数 property 赋以指示所需绘制特征的 GLU 符号常量，参数 value 赋以一个浮点数值或一个设定所需特性的 GLU 符号常量。在 gluNewNurbsRenderer

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

语句之后可指定多个gluNurbsProperty函数。使用gluNurbsProperty函数设定的许多特征都是曲面参数，如下节所述。

8.18.4 GLU的B样条曲面函数

下面的语句给出了生成一个B样条曲面所需调用函数的基本序列。

```
GLUnurbsObj *surfName

surfName = gluNewNurbsRenderer ( );
gluNurbsProperty (surfName, property1, value1);
gluNurbsProperty (surfName, property2, value2);
gluNurbsProperty (surfName, property3, value3);
    .
    .
    .
gluBeginSurface (surfName);
    gluNurbsSurface (surfName, nuKnots, uKnotVector, nvKnots,
                     vKnotVector, uStride, vStride, &ctrlPts [0][0][0],
                      uDegParam, vDegParam, GL_MAP2_VERTEX_3);
gluEndSurface (surfName);
```

一般情况下，定义一个B样条曲面的GLU语句和参数与定义B样条曲线所需的类似。在使用gluNewNurbsRenderer引入B样条绘制子程序以后，可以指定一些选定的曲面特征值。接着调用gluNurbsSurface函数设定曲面属性。利用此法可以定义多个有不同名字的曲面。如果没有足够的存储容量用来存储一个B样条曲面，则返回变量surfName由系统赋值为0。参数uKnotVector和vKnotVector指定在参数u和v方向的浮点型的节点值。我们用参数nuKnots和nvKnots指定每一节点向量的元素数目。多项式参数u的次数由uDegParam-1给出，而多项式参数v的次数由vDegParam-1给出。浮点型的三维控制点坐标在包含(nuKnots-uDegParam) × (nvKnots-vDegParam)个元素的数组变量ctrlPts中给出。参数u方向的连续控制点的开始位置间的整型位移值用整型参数uStride给出，而参数v方向的连续控制点的开始位置间的整型位移值用整型参数vStride给出。我们使用与B样条曲线相同的函数（gluDeleteNurbsRenderer）来清除一个样条曲面并释放分配的内存。

默认时，一个B样条曲面自动地由GLU子程序显示成一组多边形填充区。B样条曲面可设定9个特征，每个特征有两个或多个可能的值。作为一个特征设定的例子，下列语句指定一个线框式、三角形网格曲面显示：

```
gluNurbsProperty (surfName, GLU_NURBS_MODE, GLU_NURBS_TESSELLATOR);
gluNurbsProperty (surfName, GLU_DISPLAY_MODE, GLU_OUTLINE_POLYGON);
```

GLU网格子程序把曲面分割成一组三角形并按多边形轮廓方式显示每一个三角形。另外，这些三角形图元可用gluNurbsCallback函数来提取。特征GLU_DISPLAY_MODE另外的值是GLU_OUTLINE_PATCH和GLU_FILL（默认值）。使用值GLU_OUTLINE_PATCH也可获得一个线框显示，但曲面并不划分成三角形。原来的曲面仅仅显示其轮廓，包括考虑了已指定的修剪曲线。能设定为特征GLU_NURBS_MODE的值中只有GLU_NURBS_RENDERER（默认值）可对曲面进行绘制而不在回调中产生网格数据。

使用特征GLU_U_STEP和GLU_V_STEP可按每单位长度一个取样点来设定取样点的数目。默认值是100。如果要设定u或v的取样值，必须将特征GLU_SAMPLING_METHOD设定为值GLU_DOMAIN_DISTANCE。特征GLU_SAMPLING_METHOD可使用另外几个值来指定曲面网格

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

怎样实现。特征 GLU_SAMPLING_TOLERANCE 和 GLU_PARAMETRIC_TOLERANCE 用来设定最大取样长度。通过将特征 GLU_CULLING 设定为值 GL_TRUE，可以不对观察体外的对象进行网格化，从而提高绘制性能。GLU 剔除的默认值是 GL_FALSE。而特征 GLU_AUTO_LOAD_MATRIX 允许在其值为 GL_TRUE（默认值）时从 OpenGL 服务器上下载用于观察、投影和视口变换的矩阵。否则，如果设该值为 GL_FALSE，则应用必须使用函数 gluLoadSamplingMatrices 来提供这些矩阵。

要确定 B 样条特征的当前值，需使用下列查询函数。

```
gluGetNurbsProperty (splineName, property, value);
```

对指定的 splineName 和 property，相应的值返回给参数 value。

当特征 GLU_AUTO_LOAD_MATRIX 设定为值 GL_FALSE 时，引入

```
gluLoadSamplingMatrices (splineName, modelviewMat, projMat, viewport);
```

该函数指定了用于一个对象的取样和剔除子程序中的模型观察矩阵、投影矩阵和视口。当前的模型观察和投影矩阵能通过调用 glGetFloatv 来获得，而当前的视口可通过调用 glGetIntegerv 来获得。

样条对象附带的各种事件用下列函数来处理：

```
gluNurbsCallback (splineName, event, fcn);
```

参数 event 赋以 GLU 符号常量，而参数 fcn 指定一个在 GLU 常量对应的事件发生时要引入的函数。例如，如果将参数 event 设定为 GLU_NURBS_ERROR，则当发生错误时调用 fcn。其他事件用于 GLU 样条子程序中返回由网格处理生成的 OpenGL 多边形。符号常量 GL_NURBS_BEGIN 指出线段、三角形或四边形等图元的起始位置，而 GL_NURBS_END 指出图元的结束位置。图元起始位置的函数变量是一个符号常量，如 GL_LINE_STRIP、GL_TRIANGLES 或 GL_QUAD_STRIP。符号常量 GL_NURBS_VERTEX 指出已提供三维坐标数据且已调用顶点函数。另外的常量用于指出颜色值等其他数据。

gluNurbsCallback 函数的数据值由下列函数支持：

```
gluNurbsCallbackData (splineName, dataValues);
```

参数 splineName 赋以要网格化的样条对象的名字，而参数 dataValues 赋以一个数据值列表。

8.18.5　GLU 曲面修剪函数

下列语句用于指定对一个 B 样条曲面进行修剪的一组二维曲线。

```
gluBeginTrim (surfName);
   gluPwlCurve (surfName, nPts, *curvePts, stride, GLU_MAP1_TRIM_2);
   .
   .
   .
   .
gluEndTrim (surfName);
```

参数 surfName 是要修剪的 B 样条曲面的名字。确定修剪曲线的一组浮点坐标在数组参数 curvePts 中指定，其中包含 nPts 个坐标位置。在连续的坐标位置之间的整型位移在参数 stride 中给出。指定的坐标位置用来生成 B 样条的一个分段线性修剪函数。换句话说，生成的修剪“曲线”是一条折线。如果曲线点在三维齐次(u, v, h)参数空间给出，则 gluPwlCurve 中的最后一个变量设定为 GLU 符号常量 GLU_MAP1_TRIM_3。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们也可以使用一个或多个gluNurbsCurve作为一条修剪曲线。还可以构造一条gluPwlCurve函数和gluNurbsCurve混合的修剪曲线。任何指定的GLU修剪“曲线”必须没有自相交，且必须是一条封闭曲线。

下面的程序给出了一个三次Bézier曲面的GLU修剪函数。先为一条最外面的修剪曲线设定坐标点。这些位置以逆时针方向沿单位正方形指定。接下来，分两部分按顺时针方向设定最里面的修剪曲线的坐标点。设定曲面和内部修剪曲线第一部分的节点向量来生成三次Bézier曲线。图8.54给出了在单位正方形上的内部和外部修剪曲线。

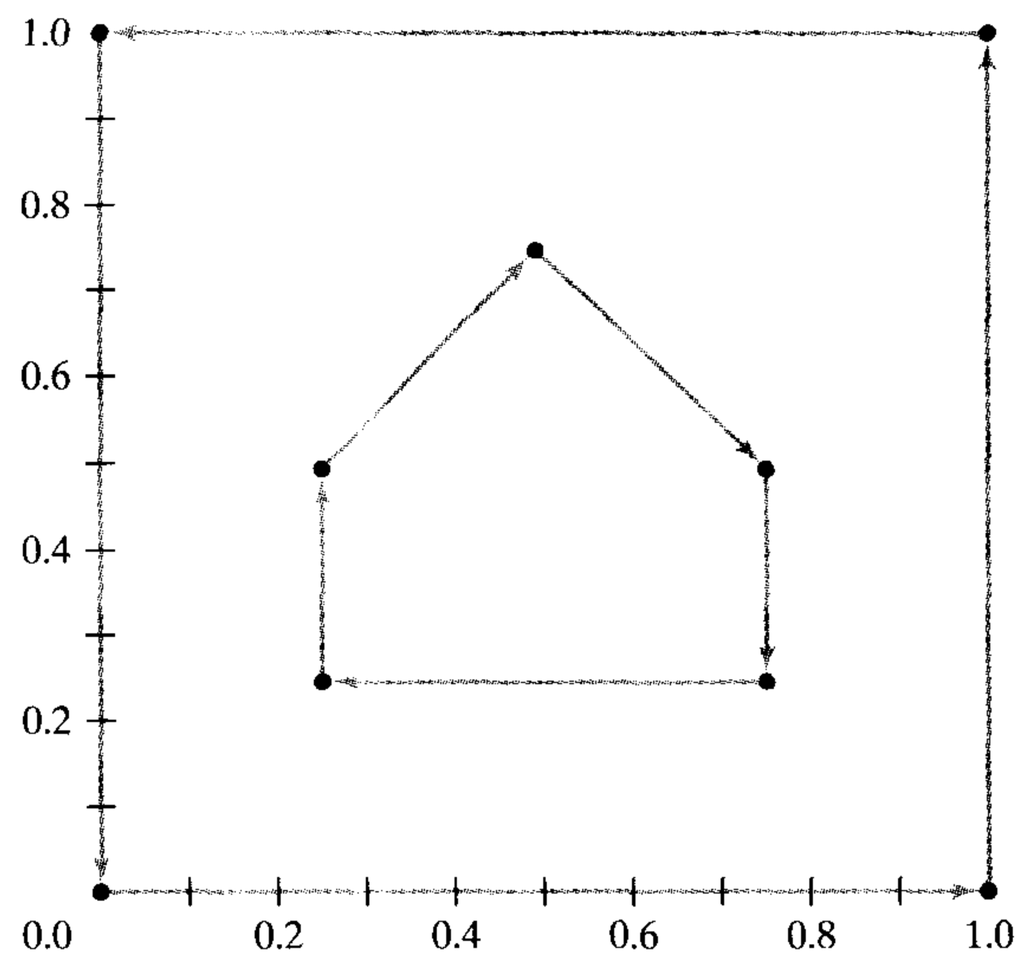


图8.54 沿单位正方形周边的修剪曲线按逆时针方向指定，而内部的修剪曲线段按顺时针方向指定

```
GLUnurbsObj *bezSurface;

GLfloat outerTrimPts [5][2] = { {0.0, 0.0}, {1.0, 0.0}, {1.0, 1.0),
                                {0.0, 1.0}, {0.0, 0.0} };
GLfloat innerTrimPts1 [3][2] = { {0.25, 0.5}, {0.5, 0.75},
                                {0.75, 0.5) };
GLfloat innerTrimPts2 [4][2] = { {0.75, 0.5}, {0.75, 0.25},
                                {0.25, 0.25), {0.25, 0.5} };

GLfloat surfKnots [8] = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0);
GLfloat trimCurveKnots [8] = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0);

bezSurface = gluNewNurbsRenderer ( );
gluBeginSurface (bezSurface);
    gluNurbsSurface (bezSurface, 8, surfKnots, 8, surfKnots, 4 * 3, 3,
                     &ctrlPts [0][0][0], 4, 4, GL_MAP2_VERTEX_3);
    gluBeginTrim (bezSurface);
       /*  Counterclockwise outer trim curve.  */
       gluPwlCurve (bezSurface, 5, &outerTrimPts [0][0], 2,
                        GLU_MAP1_TRIM_2);
    gluEndTrim (bezSurface);
    gluBeginTrim (bezSurface);
       /*  Clockwise inner trim-curve sections.  */
       gluPwlCurve (bezSurface, 3, &innerTrimPts1 [0][0], 2,
                        GLU_MAP1_TRIM_2);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        gluNurbsCurve (bezSurface, 8, trimCurveKnots, 2,
                       &innerTrimPts2 [0][0], 4, GLU_MAP1_TRIM_2):
    gluEndTrim (bezSurface);
gluEndSurface (bezSurface);
```

8.19　扫描表示法

实体造型软件包通常提供多种构造技术。**扫描表示**（sweep representation）通过平移、旋转及其他对称变换来构造三维对象。通过指定一个二维形状以及在空间区域内移动该形状的扫描可以表示该对象。可以将一组二维基本图形，如圆和矩形等作为菜单选项来提供扫描表示。获得二维图形的其他方法有封闭样条曲线构造和实体剖面片。

图 8.55 给出了一个平移扫描。图 8.55(a)中的周期性样条曲线定义了对象的剖面。然后执行一次平移扫描，即沿垂直于剖面的直线路径将控制点 $\mathbf{p}_0$ 到 $\mathbf{p}_3$ 移动一个给定的距离。在沿这条路径的区间上，复制了剖面形状，并且在扫描方向上画出连线，得到如图 8.55(b)所示的线框表示。

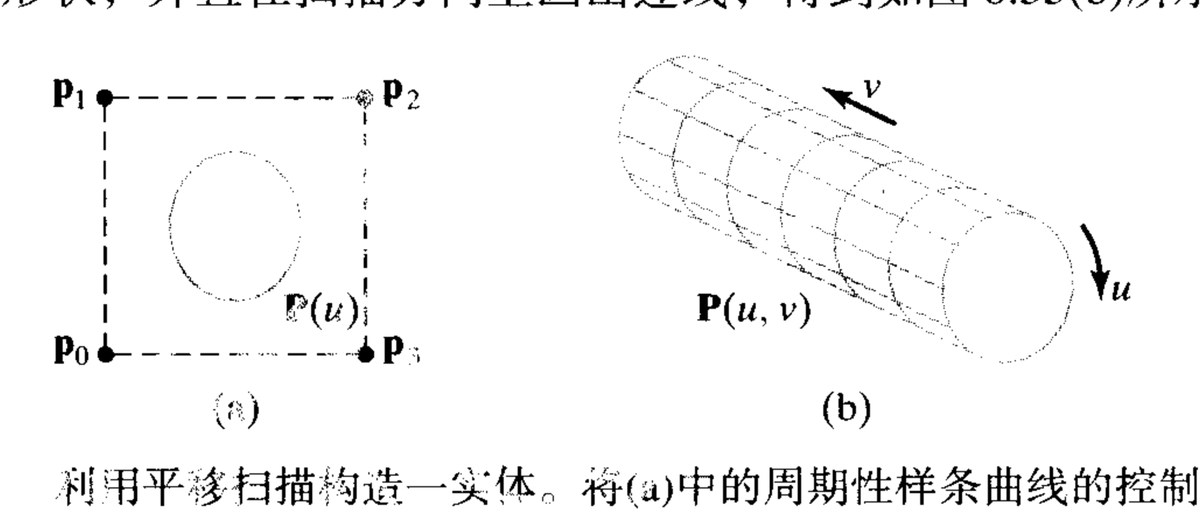


图 8.55　利用平移扫描构造一实体。将(a)中的周期性样条曲线的控制点平移生成(b)中所示的实体，其表面使用向量函数 $\mathbf{P}(u, v)$表示

图 8.56 给出了使用旋转扫描方法设计对象的例子。此时剖面所在平面上的周期性样条曲线绕旋转轴旋转，产生图 8.56(b)中的线框表示。可以选择任意轴进行旋转扫描。如果使用图 8.56(a)中的垂直于样条剖面所在平面的轴旋转，将生成一个二维形状。但是，如果此图中的剖面有一个深度，则使用一个三维对象来生成另一个三维对象。

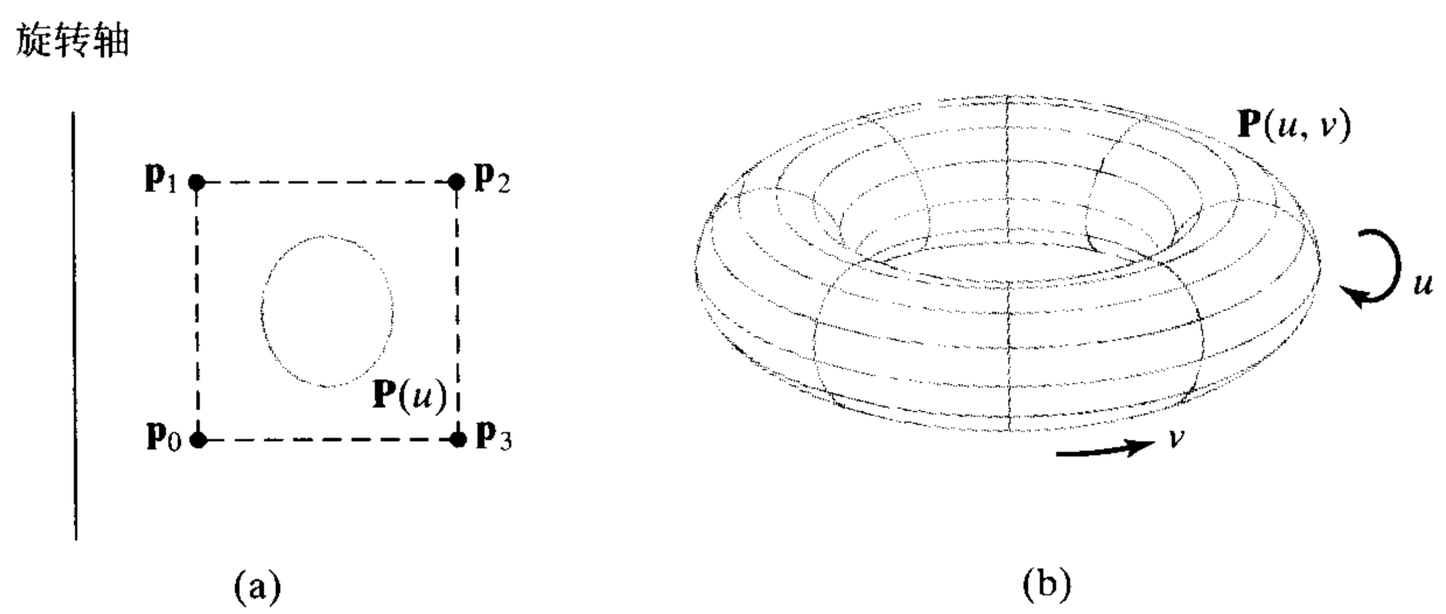


图 8.56　使用旋转扫描构造一实体。对于给定旋转轴旋转(a)中周期性样条曲线的控制点，生成(b)中所示的实体，其表面使用向量函数 $\mathbf{P}(u, v)$表示

一般来说，可以使用任何路径来指定扫描构造。对于旋转扫描，可以沿图形路径从 0 到 360° 的角度范围内移动。对于非圆形路径，可以给定描述路径的曲线函数和沿路径移动的距离。另外，可以沿扫描路径改变剖面的形状和大小。或者在将该形状移过某空间区间时，改变剖面相对于扫描路径的方向。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.20 结构实体几何法

实体建模的另一种技术是使用集合操作来组合两个三维对象。这种建模方法称为**结构实体几何**（constructive solid geometry，CSG）法，通过两个指定对象间的并、交或差操作而生成一个新对象。

图 8.57 和图 8.58 给出了使用集合操作形成新形体的例子。在图 8.57(a)中，将一个长方体和一个棱锥放在相邻的位置。使用并操作，可以得到图8.57(b)中的合并对象。图8.58(a)表示了有重叠部分的一个长方体和一个柱体。使用交操作，可以得到图 8.58(b)中所生成的实体；使用差操作，可以得到图 8.58(c)中的实体。

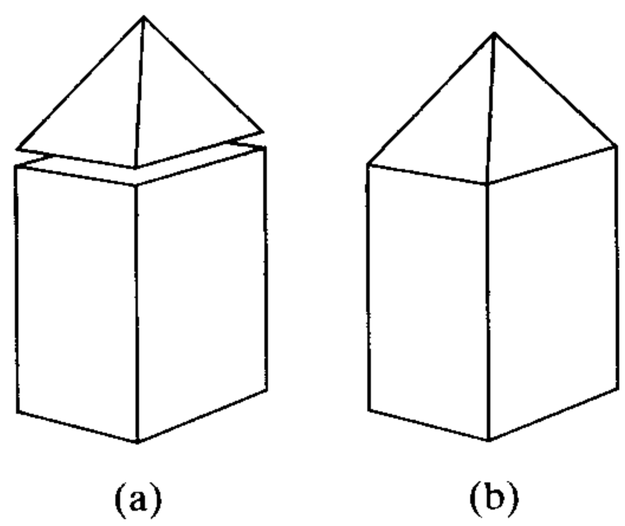


图 8.57 合并(a) 中的两个对象，使用并操作生成(b)中单个组合实体

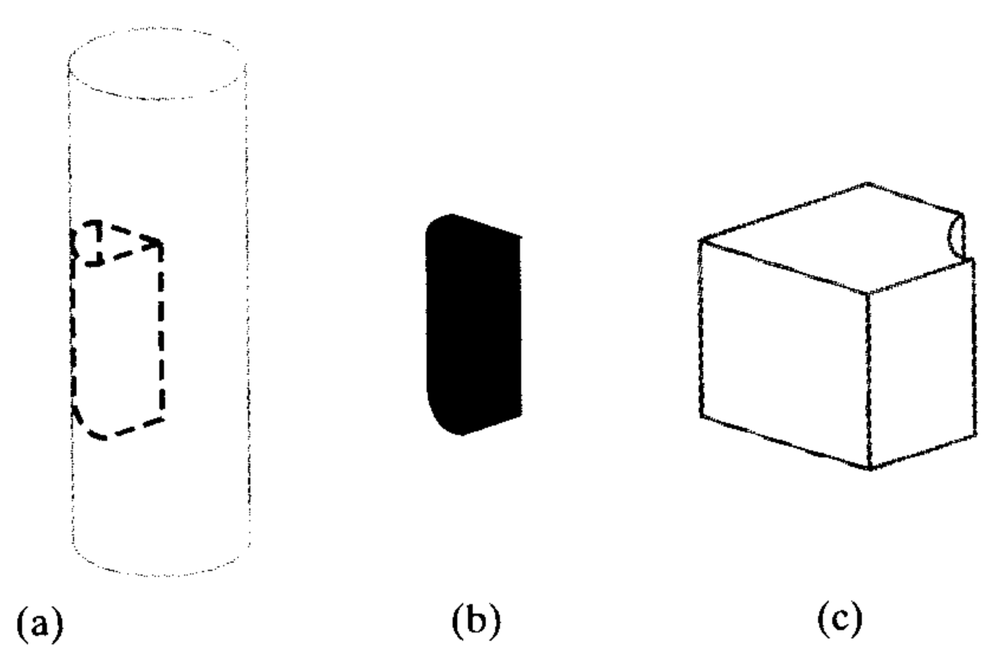


图 8.58 (a)两个重叠对象；(b)由交操作形成的楔形 CGS 对象；(c)利用差操作形成的 CSG 对象从长方体中减去与圆柱体重叠的部分

CSG 的应用开始于三维对象的初始集，称为 CSG 基本图元，如长方体、棱锥、圆柱体、圆锥体、球体和某些使用样条曲面的实体。基本图元可作为 CSG 软件包的菜单选项来提供，基本图元本身可以使用扫描方法、样条构造或其他建模程序形成。使用一个交互式 CSG 软件包，首先选择两个基本图元并拖到某空间位置，并指定一个组合操作（并、交或差）就可创建一个新对象。这个新对象可与一个已存在的形体组合生成另一个新对象。我们可以继续构造新对象，直到形成了所需的对象。利用这一程序设计的对象可以表示成一个二叉树，请参见图 8.59。

当以边界表示描述对象时，通常使用**光线投射**（ray-casting）方法来实现构造实体几何的操作。通过确定与从 xy 平面发出的沿 z 轴方向的一组平行线相交的对象来应用光线投射。该平面称为**射线平面**（firing plane），每一射线从一个像素位置发出，如图 8.60 所示。然后沿每条射线路径确定出表面相交部分，并根据离开射线平面的距离对相交点进行分类。组合对象表面的范围由指定的操作确定。图 8.61 给出了由光线投射确定的 CSG 对象表面范围，给出了两个对象（一个立方体和一个球体）的 yz 剖切面和垂直于射线平面的像素射线路径。对于并操作，新对象合并了两个对象的内

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

部区域。对于交操作，新对象是两个对象内部区域的公共部分。对于差操作，从一个对象中减去属于另一个对象的部分。

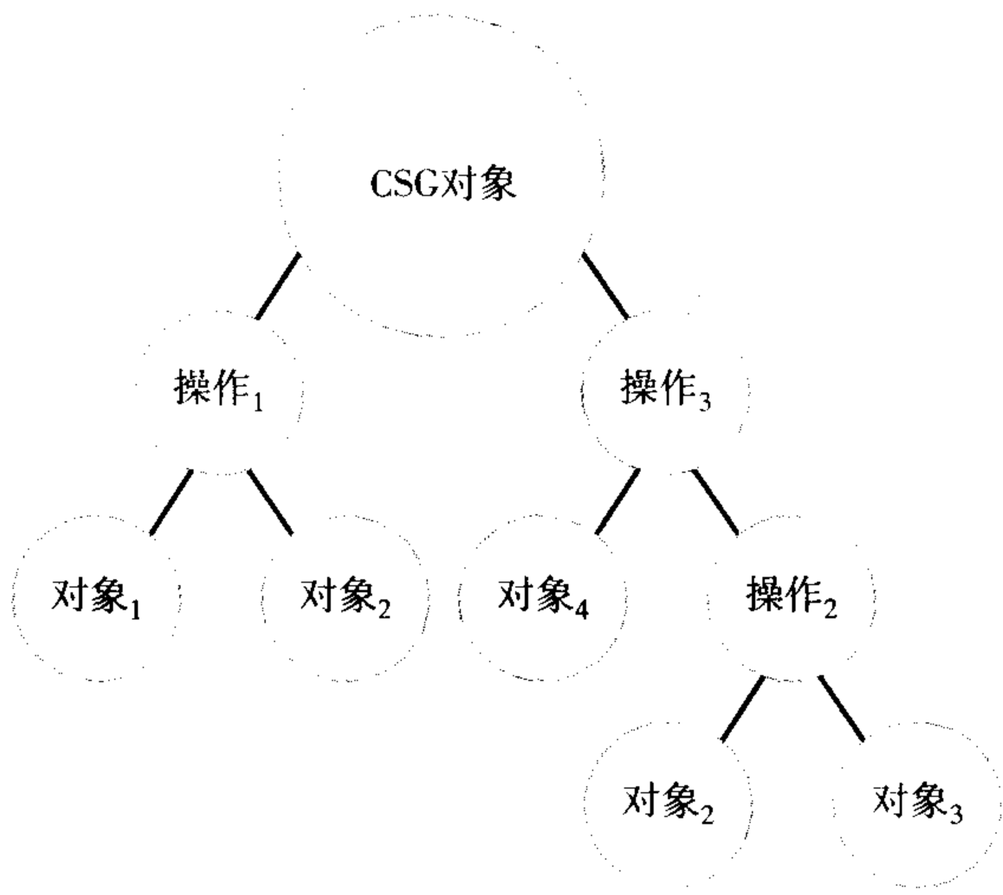


图 8.59　一个对象的 CSG 树形表示

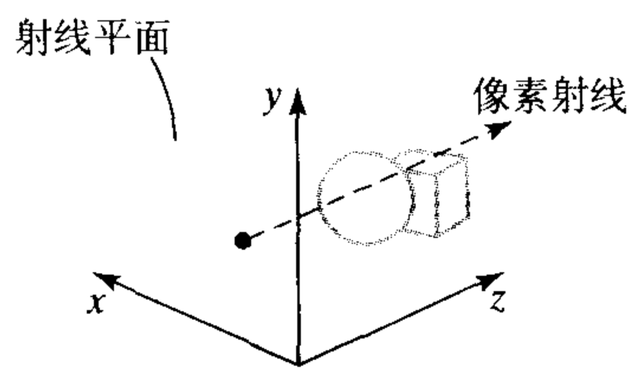
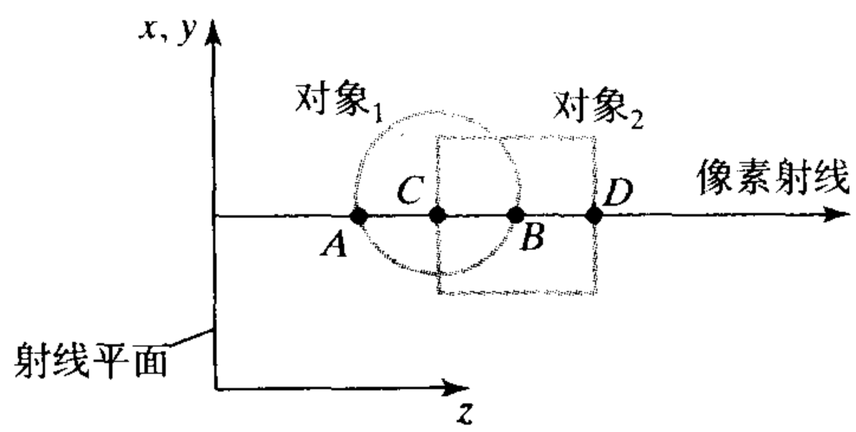


图 8.60　利用光线投射实现 CSG 操作



(a)

操作	表面范围
并	A, D
交	C, B
差（对象$_2$ − 对象$_1$）	B, D

(b)

图 8.61　沿像素射线确定表面范围

每一个 CSG 基本图元可以定义在自己的局部（建模）坐标上。通过建模变换矩阵确定其相应的世界坐标系位置，从而建立与另一对象重叠的位置。这些建模矩阵的逆矩阵，可以用来将像素射线变换到建模坐标系中，在那里对单个基本图元执行表面交计算。然后，对两个重叠对象的表面相交部分沿射线路径的距离进行分类，并且根据指定的集合操作来确定组合对象的范围。以上步骤重复作用于合并在 CSG 树中的每对对象，即形成一个特定对象。

设计了 CSG 对象之后，可以利用光线投射法确定物理性质，如体积和质量。为了确定对象的体积，将射线平面分成很多小方块，如图 8.62 所示。然后，对与从像素(i, j)位置发出的射线 A_{ij} 相交的剖面片计算对象体积 V_{ij}：

$$V_{ij} \approx A_{ij} \Delta z_{ij} \tag{8.101}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

其中Δz_{ij}是沿位置(i, j)的射线在对象中的深度。如果对象内部有一孔，则Δz_{ij}是射线上两对相交点之间的距离之和。这时 CSG 对象的总体积计算为

$$V \approx \sum_{i,j} V_{ij} \tag{8.102}$$

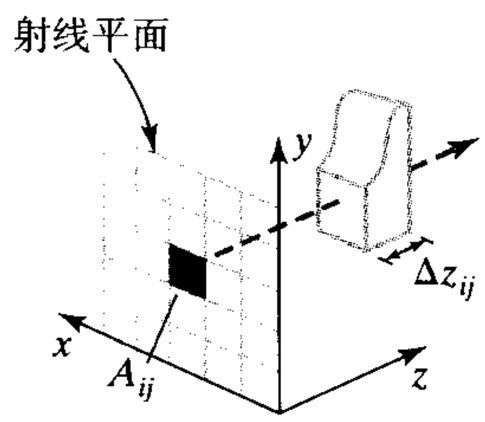


图 8.62 沿射线平面上小区域A_{ij}中的一个射线路径来确定对象体积

给出对象密度函数$\rho(x, y, z)$，可以近似求得经过位置(i, j)上射线的对象的质量：

$$m_{ij} \approx A_{ij} \int \rho(x_{ij}, y_{ij}, z)\, dz \tag{8.103}$$

其中，一维积分通常用来估算而不是真正进行积分，这依赖于密度函数形式。CSG对象的全部质量大约为

$$m \approx \sum_{i,j} m_{ij} \tag{8.104}$$

其他对象性质，如重心和惯性矩，可以利用类似计算而得到。在射线平面上进行更小的细分，可以改进对象性质的近似计算。

如果使用八叉树表示对象，那么可以通过扫描树形结构（描述空间八分圆内容）、使用CSG程序来执行集合操作。下一节将叙述该过程，搜索单位立方体的八分圆和子八分圆，从而定位两个将要组合的对象所占的区域。

8.21 八叉树

分层树形结构称为**八叉树**（octree），用于在某些图形系统中表示实体。需要显示对象剖面的医学图像和其他应用通常使用八叉树进行表示。构造出树形结构后，要求每一个节点对应于一个三维空间区域。这种实体表示利用了空间相关性，从而减少三维对象的存储需求。这种方式也提供了存储有关对象内部信息的方便表示。

三维空间的八叉树编码过程是称为**四叉树**（quadtree）编码的二维空间编码方法的扩展。通过连续将二维区域（通常为正方形）分成四等分而得四叉树。四叉树上的每个节点有四个数据元素，每个象限都在此区域内（参见图8.63）。如果一个象限中所有像素的颜色相同（均质象限），此节点内的数据元素存储该颜色。另外，在数据元素中设定一标记以标明此象限是均质的。假设图8.63的象限2中的所有像素都是红色的，则将红色码放入到节点的数据元素2中。否则，此象限称为非均质的象限，该象限本身再分成四个象限，如图8.64所示。此时节点中相应的数据元素标明此象限为非均质的，并存储指向四叉树中下一个节点的指针。

生成四叉树的一种算法将测试指定区域内赋给对象的像素颜色强度值，并建立相应的四叉树编码。如果原区域中每个象限有单一的颜色规格，则四叉树只有一个节点。对于一个非均质区域，连续细分象限直到所有的象限都是同质的。在图8.65中，大区域内包括一个具有单一颜色的小区域，此颜色不同于大区域中其他部分的颜色。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

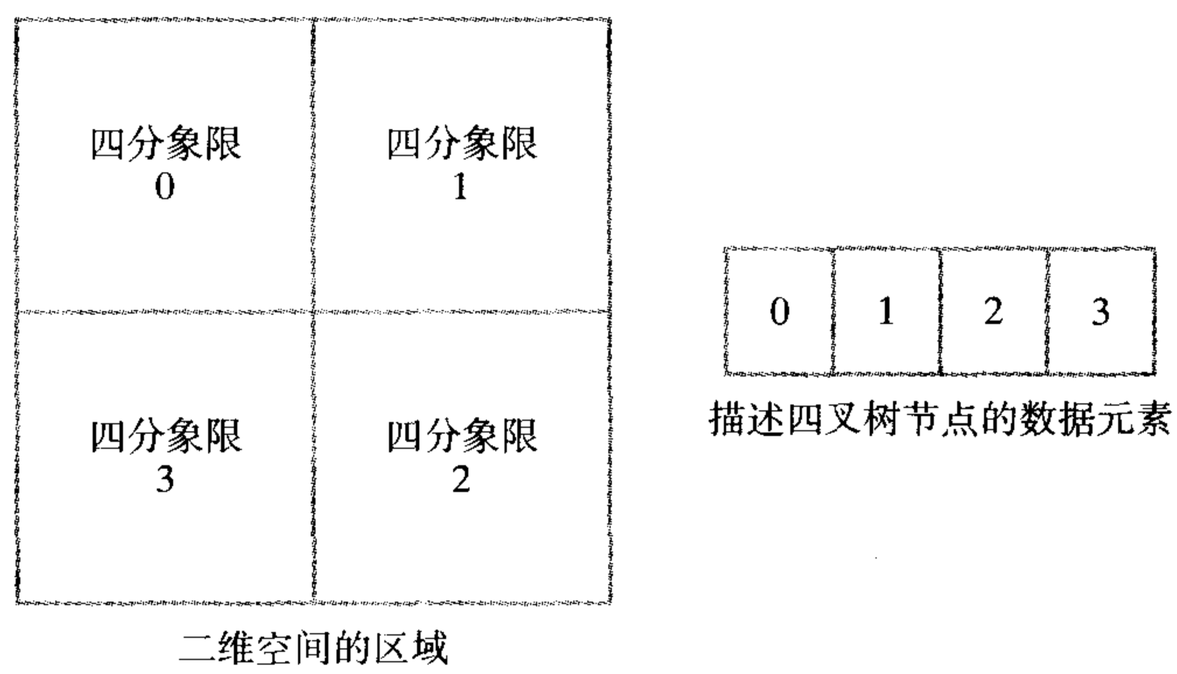


图 8.63　将 xy 平面上的正方形区域分成若干个象限，且相应的四叉树节点有四个数据元素

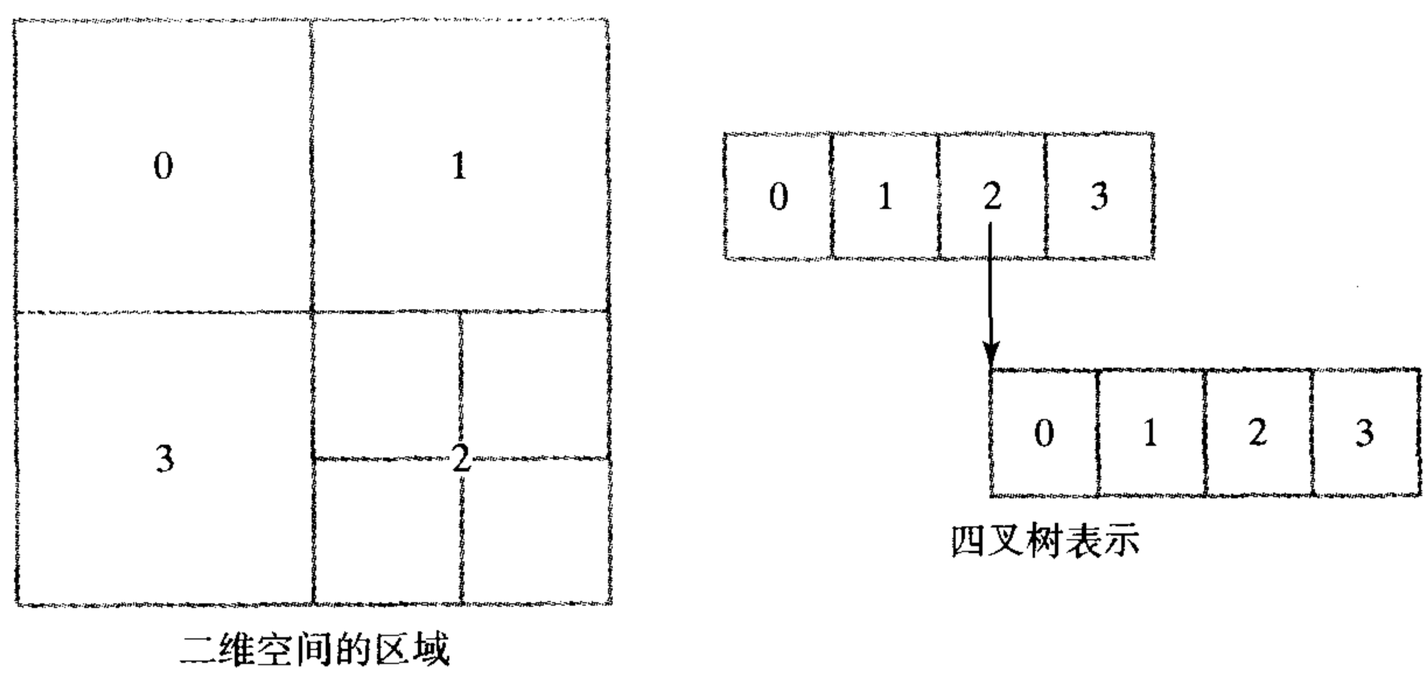


图 8.64　xy 平面上具有两层象限分割及相应的四叉树表示的正方形区域

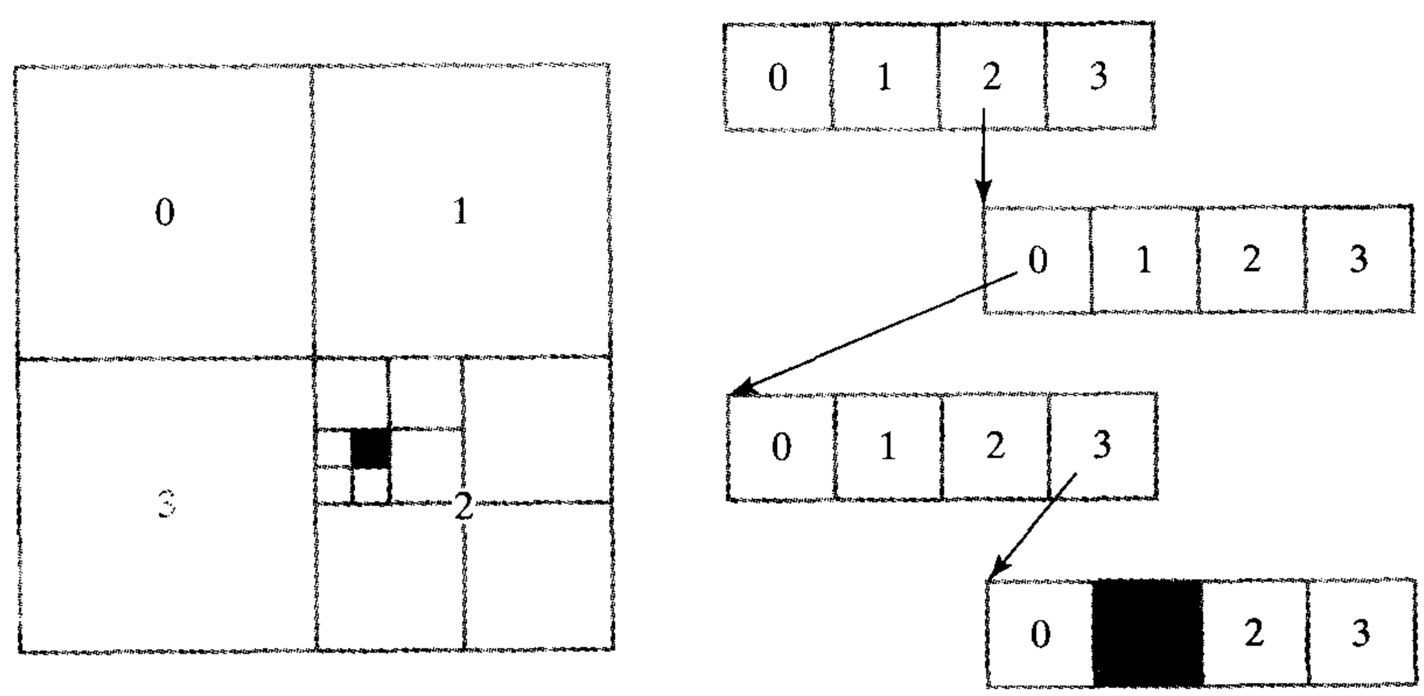


图 8.65　在实体背景中包含一个前景颜色像素区域的四叉树表示

因为每个单色区域使用一个节点表示，所以当空间区域存在较大的单一颜色区域时，四叉树编码能节省存储空间。对于含有 $2n \times 2n$ 个像素的区域，四叉树表示至多有 n 层。四叉树中每个节点至多有 4 个直接后代。

八叉树编码方法将三维空间区域（一般是立方体）分成八等分，并且在树上的每个节点处存储 8 个数据元素，如图 8.66 所示。与矩形显示区域中的像素成分相仿，三维空间的每个元素称为**体素**（volume element）或**体元**（voxel）。八叉树中的一个体元存储一个均质子区域的特征值。空间三维区域中对象的特征包括颜色、材质类型、密度和其他物理特征。例如，空间内选择区域中的对象可包括石头和树或薄纸、骨头和人体组织。空间的空白区域用体元类型“void”表示。使用八叉树表

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

示时，非均质八等分再分成八等分，直到该子区域是均质的。对于一个八叉树而言，树上每个节点有0到8个直接后代。

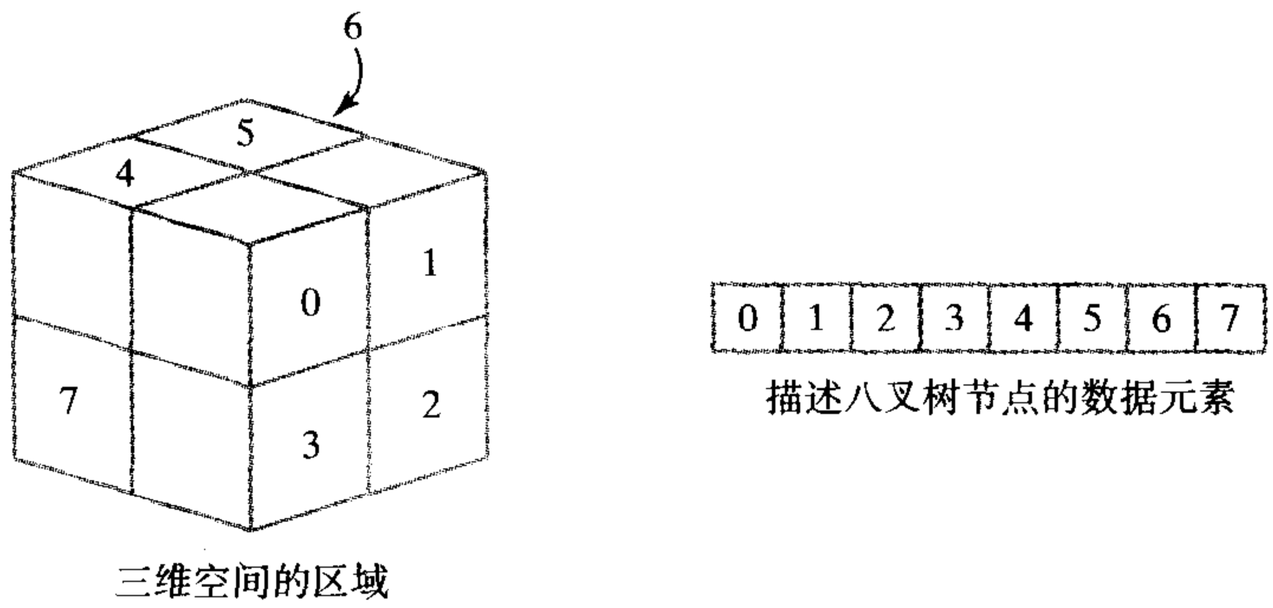


图 8.66 三维空间区域分成若干个八分圆，并且相应的八叉树节点有 8 个数据元素

可以建立为以任何形式定义的对象生成八叉树的算法，如多边形网格、曲面片或实体几何构造。对于单个对象，利用对象的坐标范围确定一个包围盒（平行六面体）来生成八叉树表示。

一旦对实体建立一个八叉树表示，可以将各种操作子程序应用到该实体上。对于同一空间区域的两个八叉树表示，可以应用执行集合操作的算法。对于并操作，新八叉树通过合并同一个区域内的每个输入树来构造。同样，通过寻找两棵输入的八叉树的重叠区域来执行交操作。类似地，存储仅由一个对象所占据而不被另一对象所占据的八等分，从而完成差操作。

已开发出一些其他的八叉树处理算法。例如，三维八叉树的旋转通过对所占八等分应用变换操作而实现。可视面判定则由寻找从前到后的八等分而实现。要确定场景中的可见对象，需先确定前面的八等分是否被占据，如果没有，则处理其后的八等分。该过程一直进行到沿观察方向确定到一个八等分。沿任意观察路径从前往后找到的第一个对象是可视的，则将该对象的有关信息转换成显示用的四叉树表示。

8.22 BSP 树

除了每次将空间分成两部分而不是八部分之外，这种表示方法类似于八叉树编码。利用**二叉空间分割树**（binary space-partitioning，或BSP树），可以每次将一场景根据任意位置和任意方向的平面分为两部分。在八叉树编码中，每次使用与笛卡儿坐标平面对齐的三个相互垂直的平面对场景进行分割。

相对于使用空间细分法，BSP树提供了一种更加有效的分割方法，因为可将分割平面的位置和方向根据适合于对象的空间分布来确定。与八叉树相比，可以减少场景树的表示深度，这样就减少了搜索树的时间。另外，BSP 树可以有效地识别可视面和使用光线追踪算法对空间进行分割。

8.23 分形几何方法

在前面一些章节中讨论的各种对象表示都使用欧氏几何方法，即对象形状由方程来描述。这些方法适用于讨论加工过的对象：具有平滑的表面和规则的形状。但自然景物，如山脉和云，则是不规则或粗糙的，欧氏方法不能真实地模拟这些对象。可以使用**分形几何方法**（fractal-geometry method）来真实地描述自然景物，使用过程而不是使用方程来对对象进行建模。正如我们所期望的，过程描绘的对象其特征远不同于方程描绘的对象。对象的分形几何表示可以用于很多领域，从而描

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

述和解释自然景物的特性。在计算机图形学中，使用分形方法来产生自然景物显示及各种数学和物理系统的可视化。

分形对象有两个基本特征：每点上具有无限的细节以及对象局部和整体特性之间的自相似性（self-similarity）。自相似性可以有不同的形式，这取决于分形表示的选择。我们利用一个过程来描述分形物体，该过程为产生对象局部细节指定了重复操作。自然景物在理论上可以使用重复无限次的过程进行表示。事实上，自然景物的图形显示仅使用有限步生成。

如果放大一连续的欧氏形状，不管其有多复杂，最终可以得出平滑的放大图像。但是，如果在分形对象中放大，则连续地看到原图中出现的更多的细节。从越来越近的位置观察山，会明显地看到类似的锯齿形状（参见图 8.67）。靠近一些再观察山，一个个突出物和石块的小细节呈现在眼前。再靠近一些，我们可以看到岩石轮廓，然后是石头、沙子。每一步，轮廓将显示得更弯曲、更倾斜。如果将沙子放到显微镜下，将会在分子级中看到同样重复的细节。类似的形状有海岸线、树叶和云等。

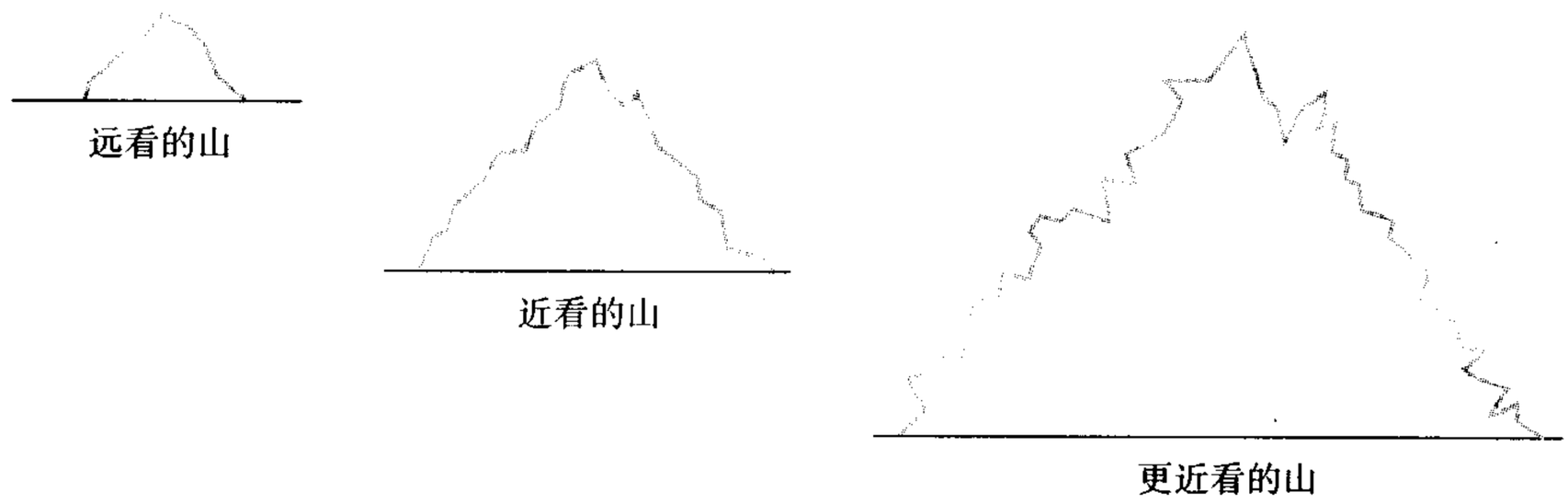


图 8.67　不同放大级别中，山的轮廓的粗糙表面

通过选择一个分形的局部，并且在同样大小的观察区内显示，可以得到分形对象的放大显示。对分形对象的该部分再进行构造操作，可以得到放大级的增加的细节。重复这一处理，就可以显示越来越多的对象细节。由于构造过程中包含无穷性，分形对象没有确定的大小。当我们考虑越来越多的细节时，对象的大小趋于无限，但对象的坐标范围保持在有限的区间内。

可以使用一个数字，称为分形维数（fractal dimension）来描述对象细节的变化。与欧氏维数不同，该数字不一定是整数。对象的分形维数有时称为分数维数（fractional dimension），这是名称“分形”的基础。

已经证明分形方法在模拟多种自然景物时是有用的。在图形学应用中，分形表示用于模拟岩层、云、水、树，以及其他植物、羽毛、毛皮和各种表面纹理，或者仅仅是为了制造漂亮的图案。在其他学科中，分形模型可以用于星体分布、河岸、月球陨石坑、雨地，以及表示股市变化、音乐、交通流量、人口资源利用、数字分析技术收敛区域的边缘。

8.23.1　分形生成过程

通过对一个空间区域内各点重复使用指定的变换函数，可以生成一个分形对象。如果 $\mathbf{P}_0=(x_0, y_0, z_0)$ 是选定的初始点，则每次重复变换函数 F 的计算，可以生成细节后继层：

$$\mathbf{P}_1=F(\mathbf{P}_0),\qquad \mathbf{P}_2=F(\mathbf{P}_1),\qquad \mathbf{P}_3=F(\mathbf{P}_2),\qquad \cdots \tag{8.105}$$

一般情况下，变换函数可以应用于给定的点集，或者将变换函数应用于基本元素的初始集上，如直线、曲线、颜色区或表面。我们既可用固定的也可用随机的生成过程。变换函数也许可以定义成几何变换（对称、平移、旋转），或者利用非线性变换和统计决策参数来建立。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

尽管在定义上分形对象包含无限的细节，但我们运用有限次变换函数。因此，实际显示的对象具有有限维数。当增加变换次数以产生更多的细节时，过程性表示将接近“真正”分形。包括在最终图形显示中的细节数量依赖于重复执行的次数和显示系统的分辨率。我们不可能显示比像素还要小的细节变化。为了看到对象的更多细节，可以选择放大对象的一部分并对其重复变换函数。

8.23.2 分形分类

自相似（self-similar）分形的组成部分是整个对象的收缩形式。从初始形状开始，对整个形体应用缩放参数 s 来构造对象的子部件。对于子部件，同样使用相同的缩放参数 s，或者对对象的不同收缩部分使用不同的缩放因子。如果对收缩部分使用随机变量，则将分形称为统计自相似（statistically self-similar），其各部分具有相同的统计性质。统计自相似分形一般用于模拟树、灌木和其他植物。

自仿射（self-affine）分形的组成部分由不同坐标方向上的不同缩放参数 s_x、s_y、s_z 形成。也可以引入随机变量，从而获得随机自仿射分形。岩层、水和云是使用随机仿射分形构造方法的典型例子。

不变分形集（invariant fractal set）由非线性变换形成。这类分形包括自平方（self-squaring）分形，如Mandelbrot集（在复数空间中使用平方函数而形成）；自逆（self-inverse）分形则由自逆过程形成。

8.23.3 分形的维数

分形对象的细节变化可以使用数字 D 进行描述，D 称为**分形维数**（fractal dimension），它是对象粗糙性或细碎性的度量。有较大锯齿形的对象其分形维数较大。生成分形对象的一种方法是建立一个使用选定的 D 值的交互过程。另一种方法是从构造对象的特性来确定分形维数，尽管一般情况下分形维数较难计算。计算 D 的方法以数学的一个分支——拓扑学中定义的维数概念为基础。

自相似分形的分形维数表达式根据单个缩放因子 s 进行构造，类似于欧氏对象的细分。图8.68表示了缩放因子 s 与单位线段、正方形和立方体的再分数目 n 之间的关系。当 $s = 1/2$ 时，单位线段（参见图8.68(a)）分成两个相同长度的部分。同样，图8.68(b)中的单位正方形分成四个相等的部分。单位立方体（参见图8.68(c)）分成八个相同体积的部分。对于每一个对象，子部分数目与缩放因子的关系是 $n \cdot s^{D_E} = 1$。类似于欧氏对象，自相似对象的分形维数 D 由下列方程得到：

$$ns^D = 1 \tag{8.106}$$

求解有关**分形相似维数**（fractal similarity dimension）D 的表达式，可以有

$$D = \frac{\ln n}{\ln(1/s)} \tag{8.107}$$

对于不同部分由不同缩放因子构造而成的自相似分形，自相似维数可以由下列复杂关系式得到：

$$\sum_{k=1}^{n} s_k^D = 1 \tag{8.108}$$

其中，s_k 是第 k 个子部分的缩放因子。

在图8.68中，考虑了简单对象（线段、正方形、立方体）的细分。如果有更复杂的形状，包括曲线和曲面对象，确定子部分的结构和性质会更加困难。对于一般的对象形状，可以利用拓扑覆盖方法（topological covering method），该方法使用简单形状来逼近对象的子部分。例如，细分曲线可以使用直线段进行逼近；细分样条曲面可以使用小正方形或矩形进行逼近。其他覆盖形状，如

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

圆、球面、圆柱面也可用已经分成很多小部分的对象来逼近。覆盖方法常用于数学中通过对一组更小的对象的特征求和来确定几何性质，如复杂对象的长度、面积、体积，也可以使用覆盖方法来确定某些对象的分形维数 D。

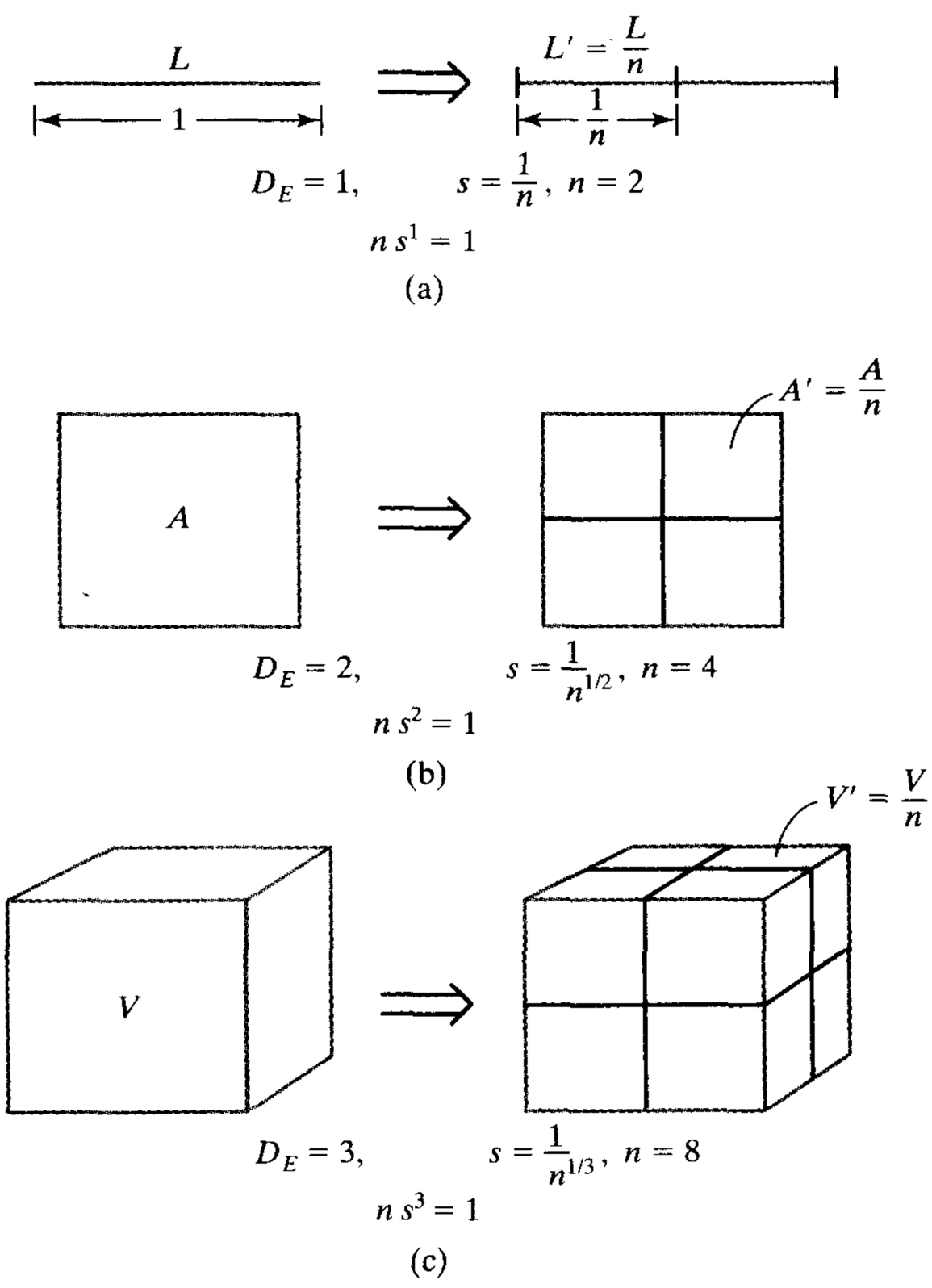


图 8.68　运用缩放因子 $s = 1/2$，通过欧氏维数细分对象：(a) $D_E = 1$，直线；(b) $D_E = 2$，单位正方形；(c) $D_E = 3$，单位立方体

拓扑覆盖概念一开始用于将常见形状的几何性质扩充到非标准形状。运用圆或球的覆盖方法的扩展，产生了Hausdorff-Besicovitch维数或分形维数概念。Hausdorff-Besicovitch维数可以作为某些对象的分形维数，但一般较难求得分形维数的值。更普遍的方法是，对象的分形维数可以利用方框覆盖方法（box-covering methods）进行估算，该方法利用了正方形或平行六面体。图 8.69 说明了方框覆盖的概念。这里，在大的不规则边界内的面积，可以通过小的覆盖正方形面积之和进行估算。

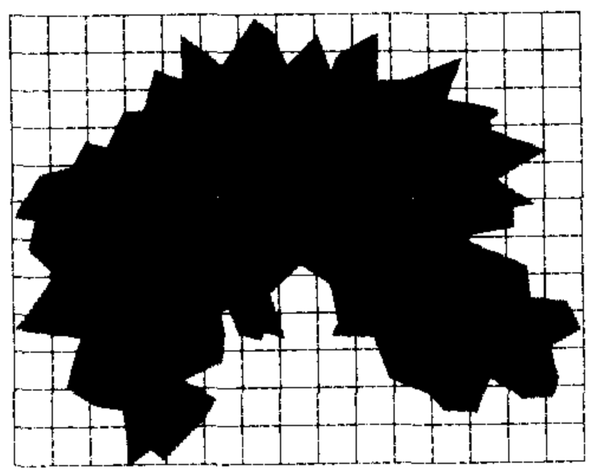


图 8.69　不规则形状的方框覆盖

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

运用方框覆盖方法，首先要确定出对象坐标，然后利用给定的缩放因子将对象分成很多小框。覆盖对象的方框数目 n 称为方框维数（box dimension），n 与对象分形维数 D 有关。对于具有单一缩放因子 s 的统计自相似对象，可以使用矩阵或立方体来覆盖对象。然后计算出覆盖方框数 n，并使用等式（8.107）来估算分形维数。对于自仿射对象，使用矩阵方框来覆盖对象，因为不同的方向有不同的缩放比例。在这种情况下，方框数目 n 及仿射变换参数用来估算分形维数。

对象的分形维数总是大于其欧氏维数（或拓扑维数），拓扑维数仅仅是需要指定对象参数的最少数目。欧氏曲线是一维的，因为可以用一个参数 u 来确定坐标位置。欧氏曲面是二维的，使用曲面参数 u 和 v。需要三个参数来指定坐标的欧氏实体是三维的。

位于一个二维平面内的分形曲线，其分形维数 D 大于 1（曲线的欧氏维数）。D 越靠近 1，则分形曲线越平滑。若 $D=2$，得到 Peano 曲线，即“曲线”完全充满了一个二维空间的区域。若 $2<D<3$，曲线自相交并且可以被覆盖无数次。分形曲线用于模拟自然景物边界，如海岸线。

空间分形曲线（不落在某一平面内）的分形维数 D 也大于 1，但大于 2 时不一定自相交。曲线充满一体积时维数 $D=3$，自相交空间曲线的分形维数 $D>3$。

分形曲面的维数一般在范围 $2<D\leqslant 3$ 内。若 $D=3$，“曲面”充满一体积。若 $D>3$，则存在实体的重叠覆盖。灌木、云、和水都是典型的利用分形曲面进行模拟的例子。

分形实体维数一般在 $3<D\leqslant 4$ 的范围内。当 $D>4$ 时，得到自重叠对象。例如，分形实体可以用来模拟云的特性，如水蒸气密度，或一空间区域内的天气状况。

8.23.4 确定性自相似分形几何构造

确定性（非随机）自相似分形几何构造，开始于一个指定的几何形状，称为初始元（initiator）。然后使用一种模型替代初始元的每部分，该模型称为生成元（generator）。

举一个例子，若使用图 8.70 中的初始元和生成元，可以构造雪花形状或 Koch 曲线，如图 8.71 所示。初始元中的每个直线段每次用四个相等线段替代。然后对该生成元缩放并应用于修改后的初始元的线段，且该过程重复若干步。每一步的缩放因子是 1/3，所以分形维数 $D=\ln 4/\ln 3\approx 1.2619$。每次初始元的线段长度也以 4/3 因子递增，因此当增加更多的细节后，分形曲线长度趋于无限（参见图 8.72）。另一个自相似分形曲线构造如图 8.73 所示。这些例子说明了分形维数越高，其突出状更明显。

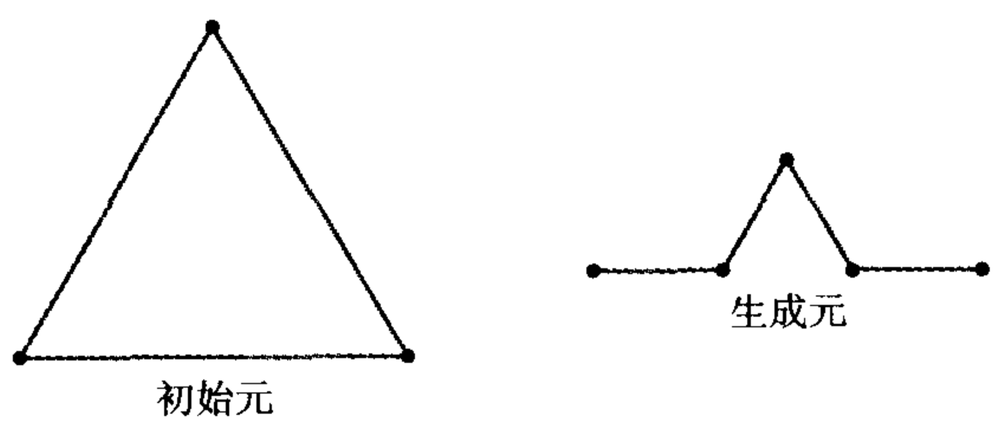


图 8.70 Koch 曲线的初始元和生成元

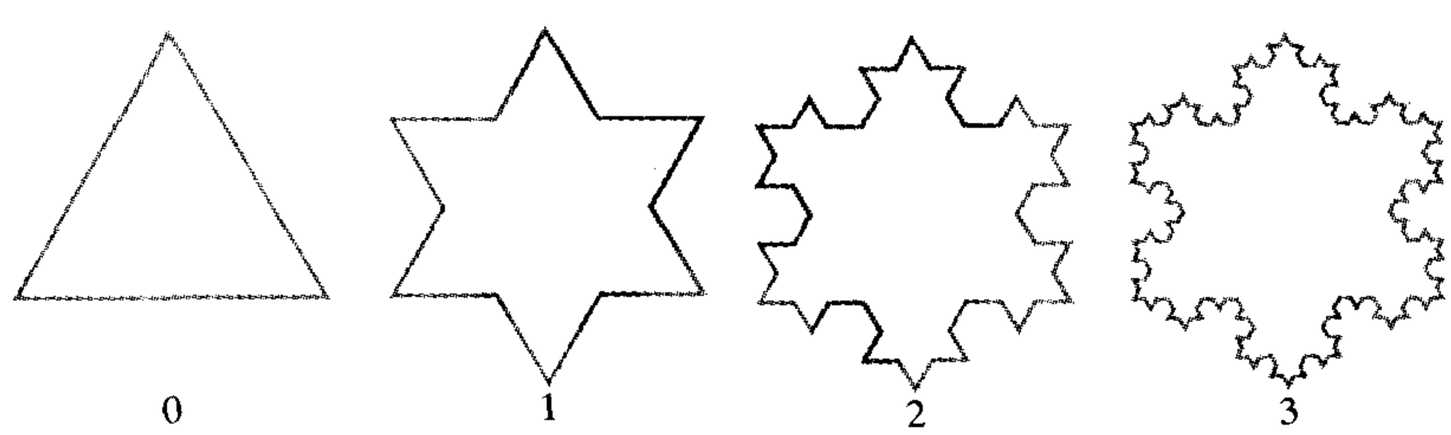


图 8.71 Koch 曲线的前三次迭代

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

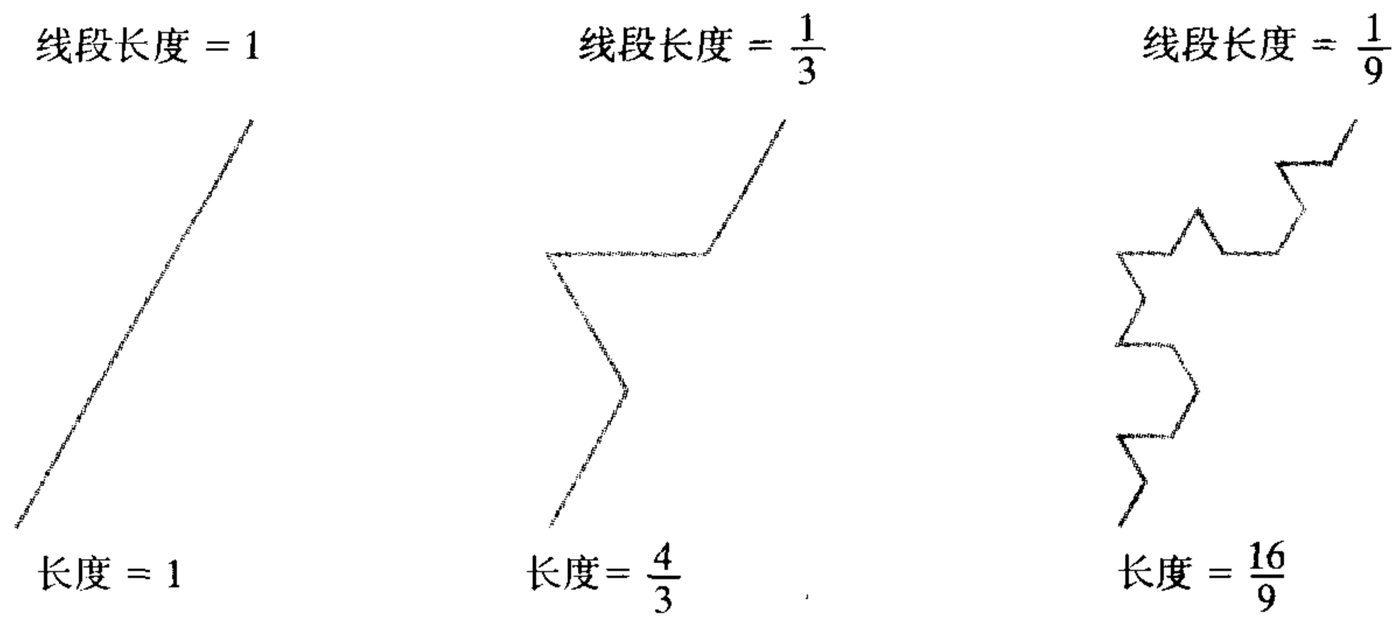


图 8.72　Koch 曲线每边长度每次增加 4/3 倍，其中每条线段的长度减少至 1/3

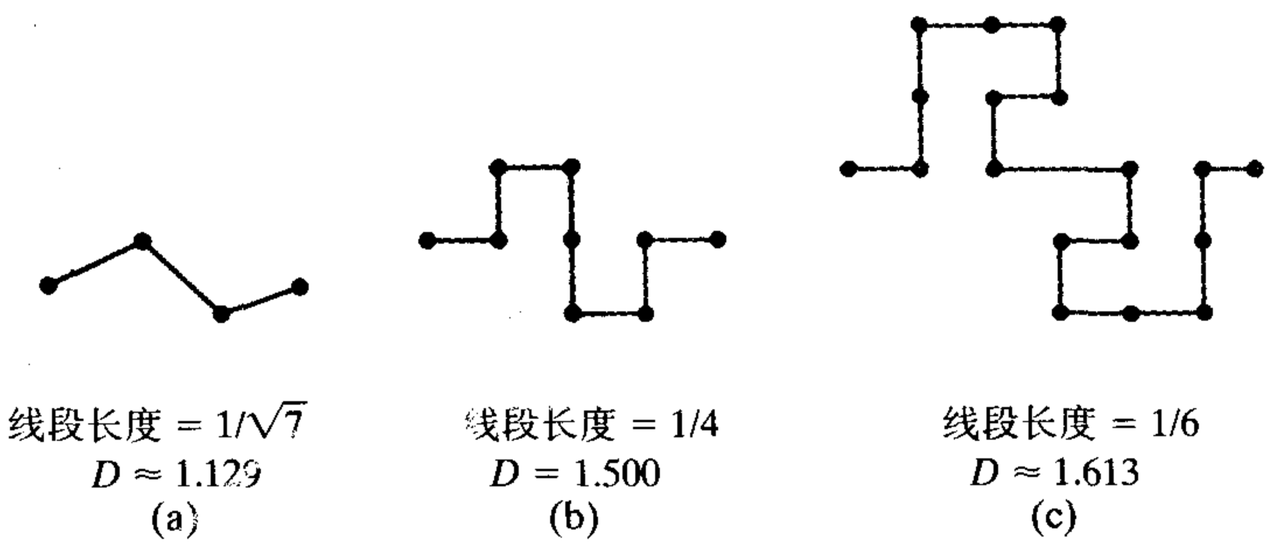


图 8.73　自相似分形曲线生成元构造和相应的分形维数

也可以将多个不相交元素作为生成元。图 8.74 给出了复合生成元的一些例子。使用复合生成元的随机变量，可以模拟带有复合部分的各种自然对象，如沿海岸分布的岛屿。

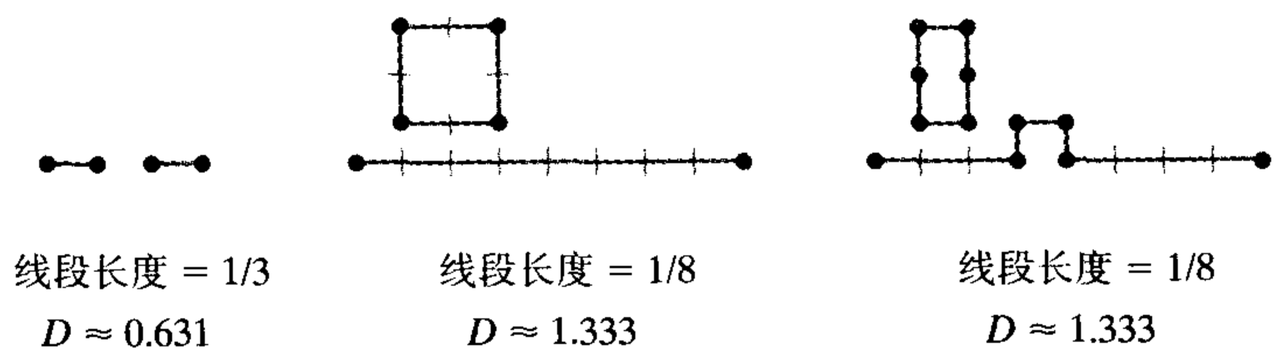


图 8.74　有多个不相连部分的生成元

图 8.75 中包含了不同长度的线段和多个缩放因子的生成元用于构造分形曲线。因此该生成曲线的分形维数由等式（8.108）给出。

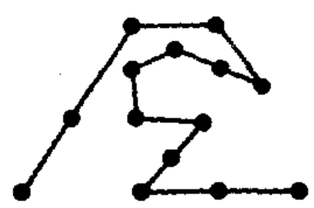


图 8.75　应用该生成元在等边三角形的边生成充满雪花状的 Peano 曲线（也称为 Peano 空间）

树和其他植物的显示可通过自相似几何构造方法实现。图8.76(a)中的蕨类植物外形的每一分支是蕨类植物整体的一个缩放版本。图 8.76(b)中的蕨类植物是对每一分支实施扭转的完全绘制。

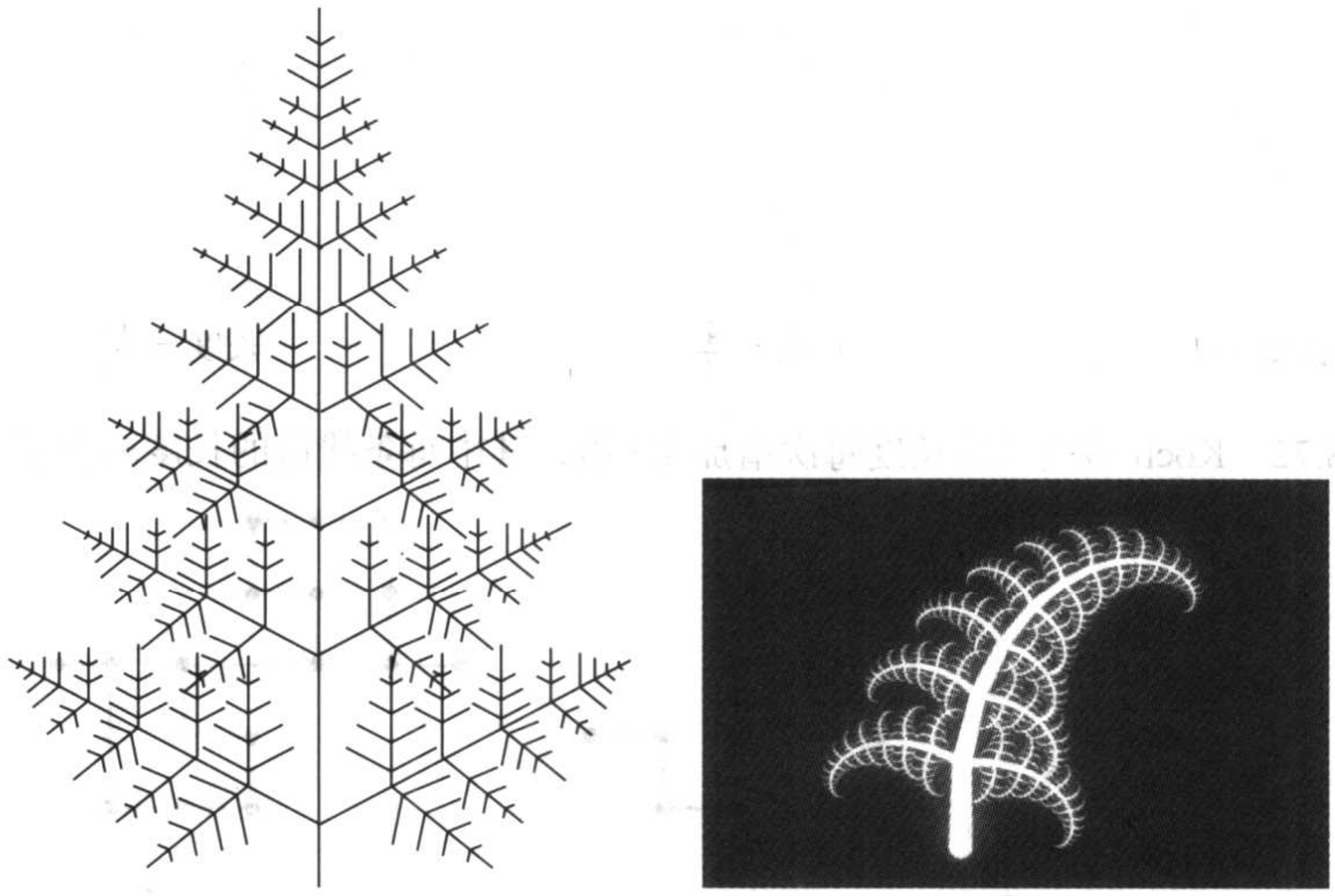
作为曲面自相似分形构造的例子，可以用 1/2 因子来缩放图 8.77 中所示的规则四面体，然后将缩放对象放到四面体的原有四个表面上。四面体的每个面变成 6 个小平面，且原侧面面积增加 3/2 倍。此曲面的分形维数是

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

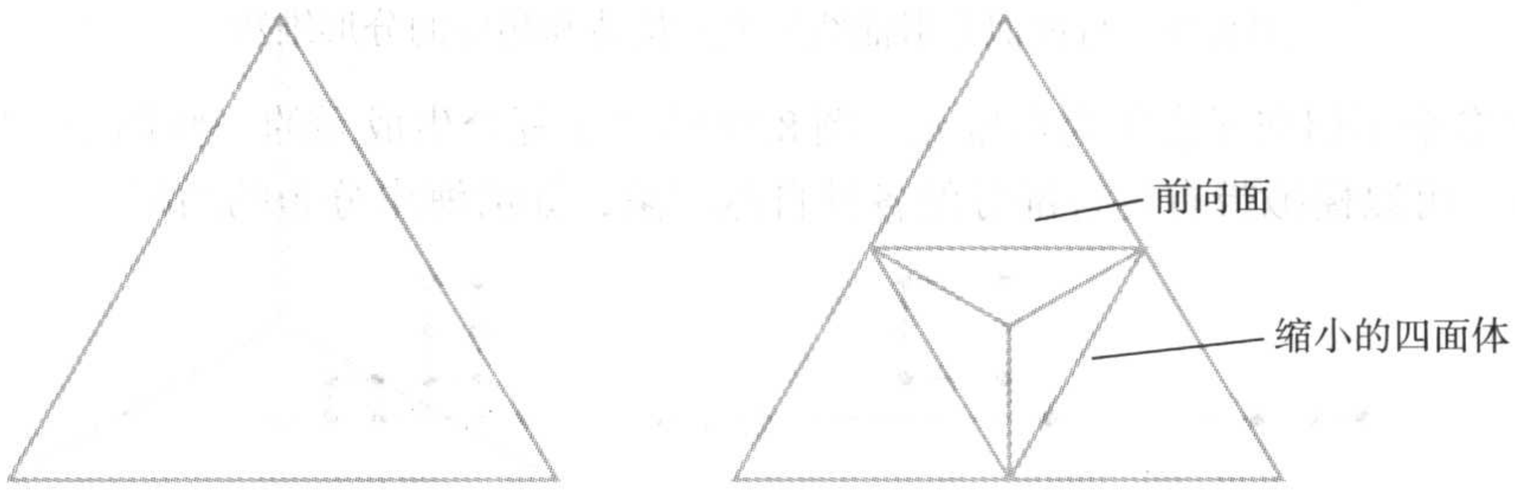
$$D = \frac{\ln 6}{\ln 2} \approx 2.584\,96$$

这给出了一个清晰的细碎表面。



(a) (b)

图 8.76 蕨类植物的自相似结构（纽约技术学院计算机图形学实验室的 Peter Oppenheimer 提供）



(a) (b)

图 8.77 以 1/2 倍缩放(a)中的四面体，将缩放对象放到原四面体的一个侧面来生成分形表面(b)

生成自相似分形对象的另一种方法是，在给定的初始元中穿孔，而不是增加更多的表面面积。图 8.78 给出了以此方法生成的分形对象。

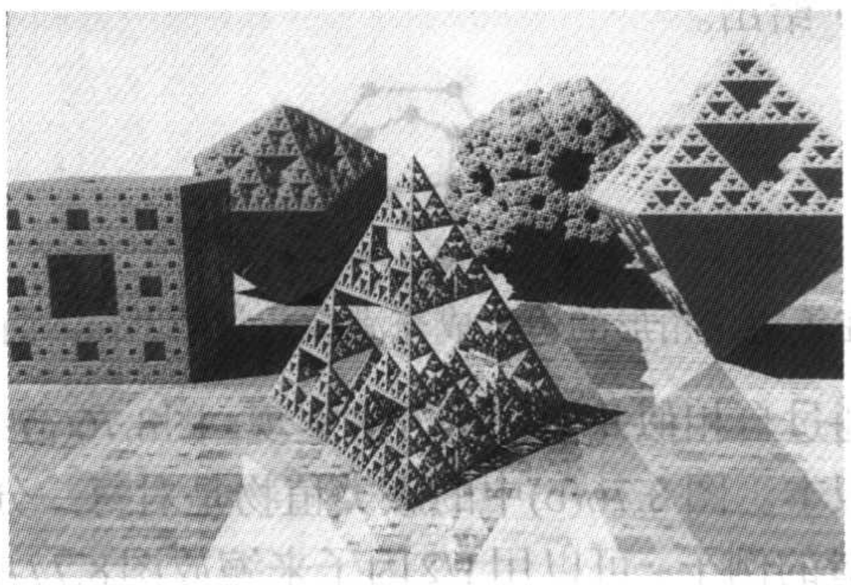


图 8.78 使用从初始元减去子部分的生成元所形成的自相似三维分形（伊利诺伊大学计算机科学系的 John C. Hart 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.23.5　统计自相似分形的几何构造

将某种随机性引入到自相似分形几何构造的一种方法是，每一步从预先确定好的图案菜单中随机选取一个作为生成元。另一种生成随机自相似对象的方法是随机计算坐标位移。例如，在图 8.79 中，每步随机选取中点位移距离，从而生成一种随机雪花模式。

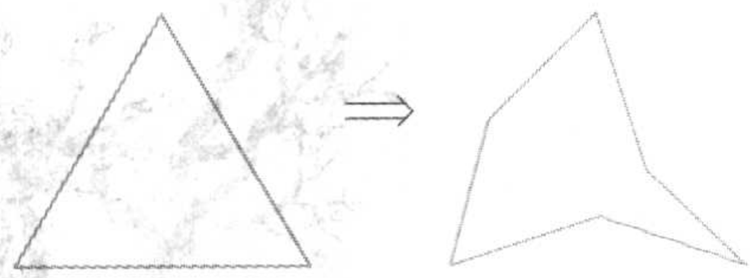


图 8.79　使用随机中点位移法来模拟“雪花”模型

图 8.80 给出了该方法的另一个例子。该显示中使用了随机缩放参数和分支方向来模拟树叶上的叶脉。

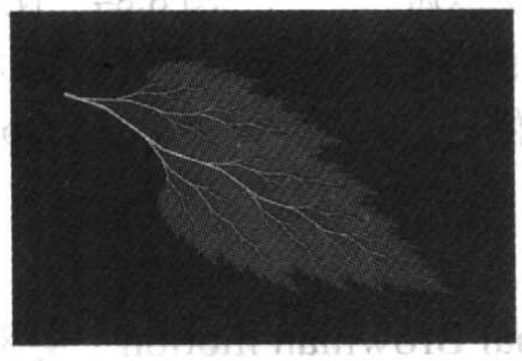


图 8.80　落叶上叶脉的随机自相似构造。树叶边界是叶脉生长的范围（纽约技术学院计算机图形学实验室的Peter Oppenheimer提供）

一旦创建了分形对象，可以使用几个变换后的对象实例来对一个场景建模。图 8.81 给出了一分形树的随机旋转例子。在图 8.82 中，显示了使用各种随机变换的分形树林。

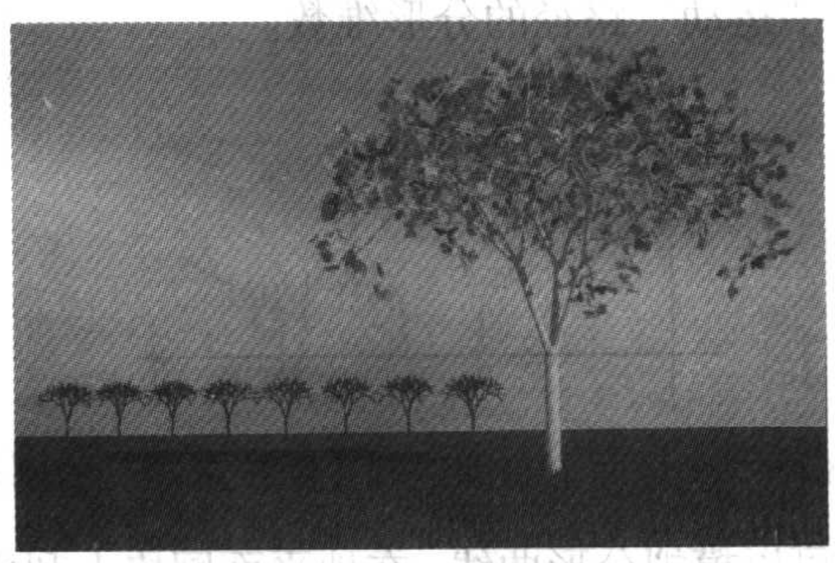


图 8.81　使用多个物体来模拟一场景，分形叶附在一棵树上，几棵树形成树丛。草丛用多个绿色锥体来模拟（伊利诺伊大学的John C. Hart提供）

图 8.82　分形树林由叶、松针、草地和树茎等多个物体模拟生成（伊利诺伊大学的 John C. Hart 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

为了模拟一些树的多节和扭曲形状，可以使用弯曲函数及缩放变换来生成随机自相似分支。这一技术在图8.83中说明。从图左部的锥形柱体开始，变换到产生（连续从左到右）一个卷体、螺旋和随机弯曲模型。随机弯曲的模拟树如图8.84所示。树茎显示利用凸映射和凸模型上的分形布朗颤动（Brownian variation）进行模拟，凸映射方法将在10.17节中讨论。

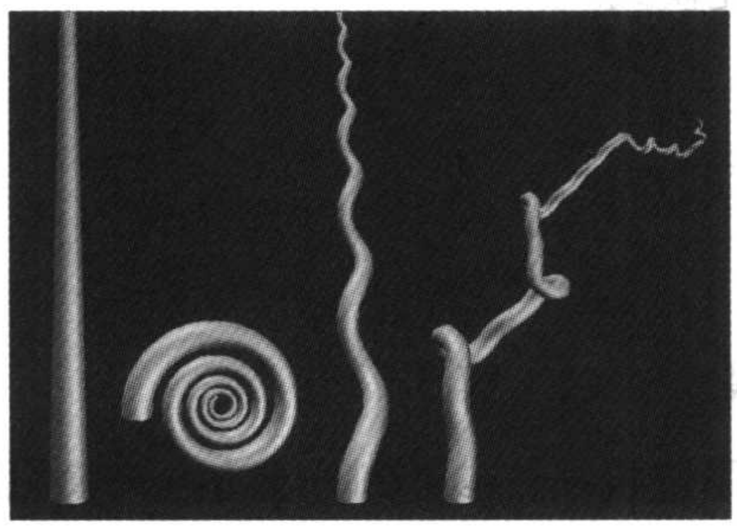


图8.83 使用卷体、螺旋和随机弯曲模拟树枝（纽约技术学院计算机图形学实验室的Peter Oppenheimer提供）

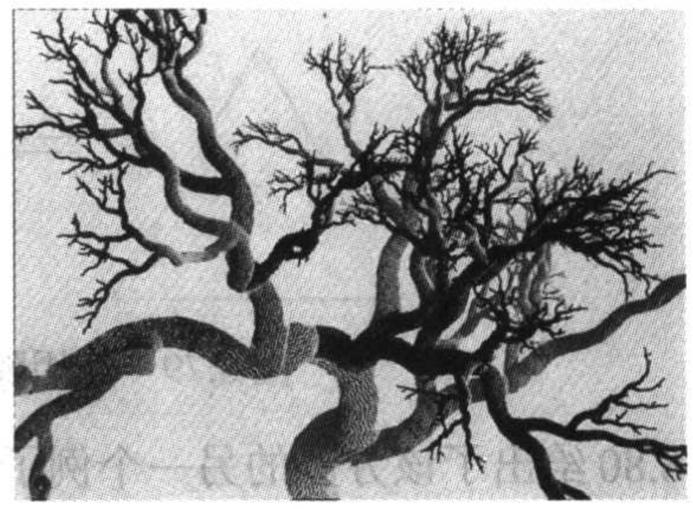


图8.84 使用随机弯曲的线模拟的树枝（纽约技术学院计算机图形学实验室的Peter Oppenheimer提供）

8.23.6 仿射分形构造方法

利用模拟分形布朗运动（fractional Brownian motion）等对象特性的仿射分形方法，可以获得地面和其他自然景物的较高真实感表示。这是标准布朗运动的扩展，是一种“随机走动”形式，描述了空气中或其他液体中分子运动的无规则性、曲折性。图8.85给出了在xy平面上的随机走动路径。从一给定位置开始，以一种随机方向和随机长度生成一条线段。然后移到第一个线段的终点并按所需的线段数重复此过程。通过在描述布朗运动的统计分布上增加一个额外参数，可以实现分形布朗运动。该附加参数建立了“运动”路径的分形维数。

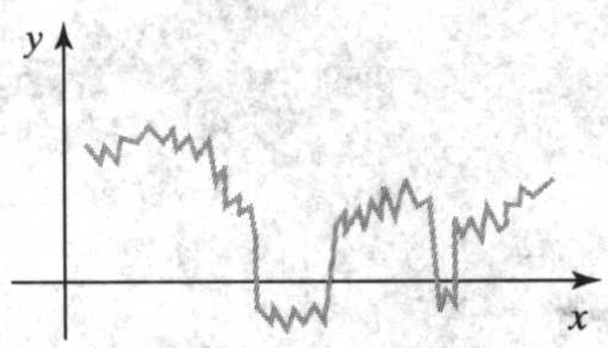


图8.85 xy平面上布朗运动（随机走动）的例子

单个分形布朗运动的路径可以模拟分形曲线。在地平面网格上利用随机分形布朗高度的二维数组，可以通过连接这些高度形成一组多边形面片来模拟山的表面。如果在球面上产生随机高度，可以模拟星球上的山、峡谷和海洋。在图8.86中，布朗运动用于生成星球上的高度变化。然后对高度进行颜色编码，最低高度画上蓝色（海洋），最高高度则画上白色（山上的雪）。分形布朗运动用于生成景色中的地面特性。陨石坑则利用随机直径和随机位置生成，运用仿射分形过程可以真实地描绘可见的陨石坑分布，以及河谷、雨和对象的其他类似系统。

调整分形布朗运动计算中的分形维数，可以变化地面特征的粗糙性。分形维数值在$D \approx 2.15$时可产生具有真实感的山脉特征，$D \approx 3.0$的高值用来产生非同寻常的特殊球面景色。也可以缩放计算高度以加深峡谷深度或增加山脉高度。图8.87给出了用分形过程模拟地面特征的几个例子。图8.88给出了分形山上的分形云的模拟景色。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

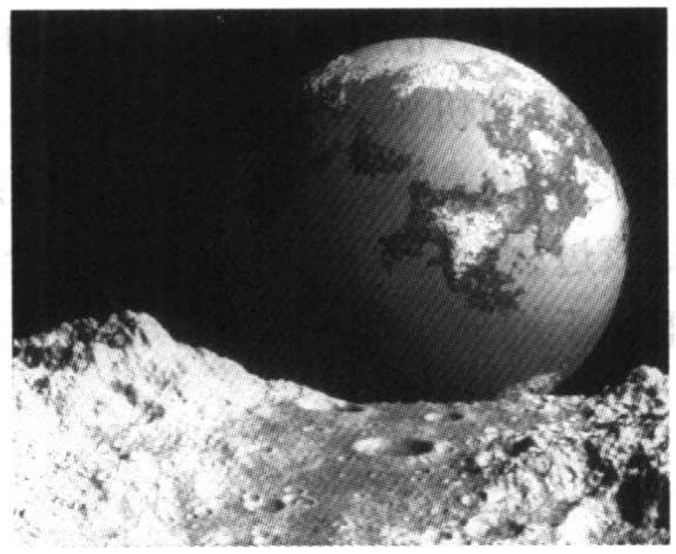
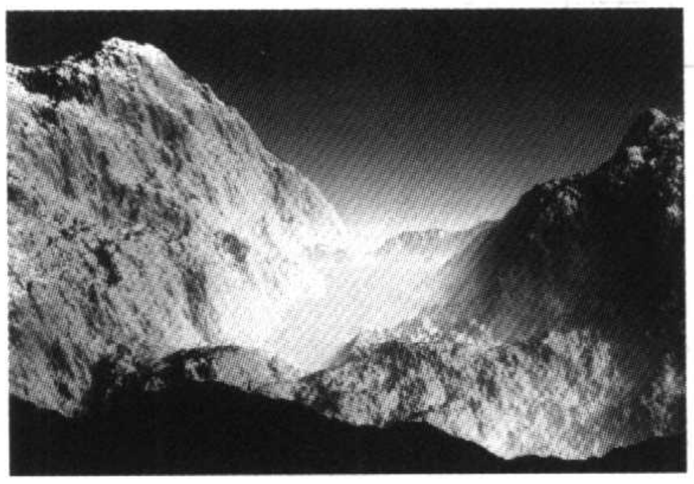
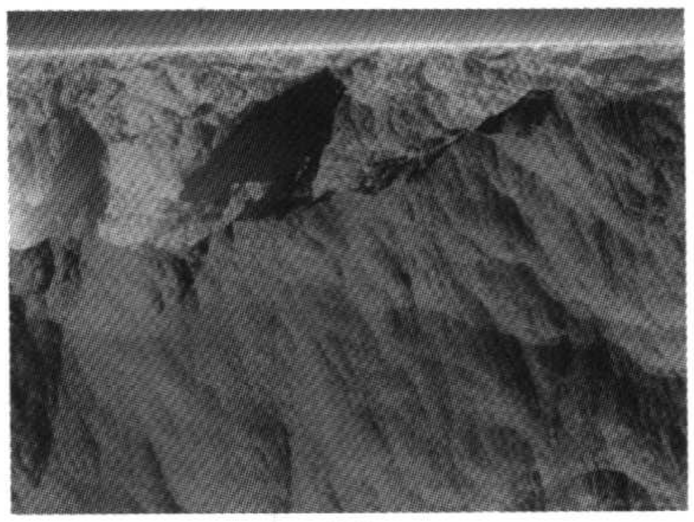


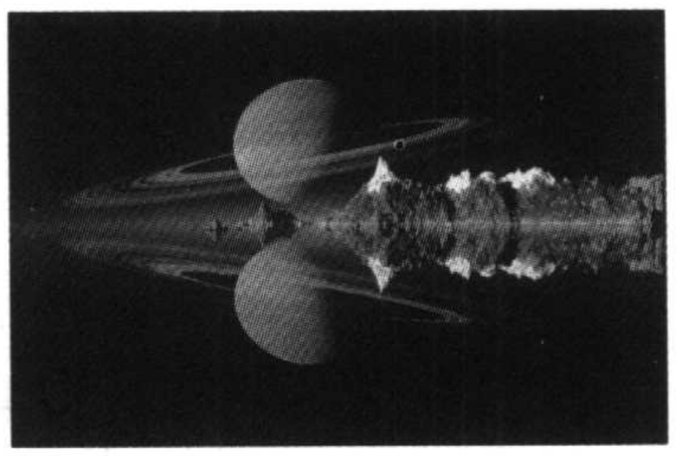
图 8.86　从分形布朗运动行星上观察布朗运动行星，背景中增加了陨石坑（R.V.Voss 和 B.B.Mandelbrot 提供）



(a)



(b)



(c)

图 8.87　使用分形布朗运动模拟的地面特征变化（(a)由 R.V.Voss 和 B.B.Mandelbrot 提供；(b)和(c)由耶鲁大学数学和计算机科学系的 Ken Musgrave 和 Benoit B. Mandelbrot 提供）

图 8.88　分形云和分形山模拟的场景（由耶鲁大学数学和计算机科学系的Ken Musgrave和Benoit B.Mandelbrot提供）

8.23.7　随机中点位移方法

分形布朗运动的计算是费时的，因为地面上的高度坐标根据傅里叶级数进行计算，该值是余弦和正弦项之和。通常使用快速傅里叶变换（FFT）方法，但对于生成分形山景物，这仍是一个缓慢的过程。因此，开发了快速**随机中点位移方法**（random midpoint-displacement method），类似于几何构造中所使用的随机位移方法，用于逼近地面和其他自然现象的分形布朗运动表示。这些方法原先用于生成科幻电影中的动画画面，包括异常的地面和星体特征。中点位移方法已经普遍用于很多应用，包括电视广告的动画。

尽管随机中点位移方法的计算速度快于分形布朗运动，但是减少了地面特征的真实感。图8.89说明了 xy 平面上生成随机走动路径的中点位移方法。从一条直线段开始，通过计算端点 y 值的平均值加上随机偏移量，可以获得线段中点位移 y 值：

$$y_{\text{mid}} = \frac{1}{2}[y(a) + y(b)] + r \tag{8.109}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

为了逼近分形布朗运动，从0到正比于$|(b-a)|^{2H}$的均方差之间的高斯分布中选择r值，其中$H=2-D$和$D>1$是分形维数。另一个获得随机偏移的方法是令$r=sr_g|b-a|$，这里s是选定的“表面粗糙度”因子，r_g是从0到均方差1之间的高斯随机值，查表即可得高斯值。然后，通过计算细分线段每一半的中点位移y值而重复该过程。连续地细分直到细分线段小于某一指定值。每一步的随机变量r值都减小，因为该值正比于细分线段长度$|b-a|$。图8.90给出了使用此方法得到的分形曲线。

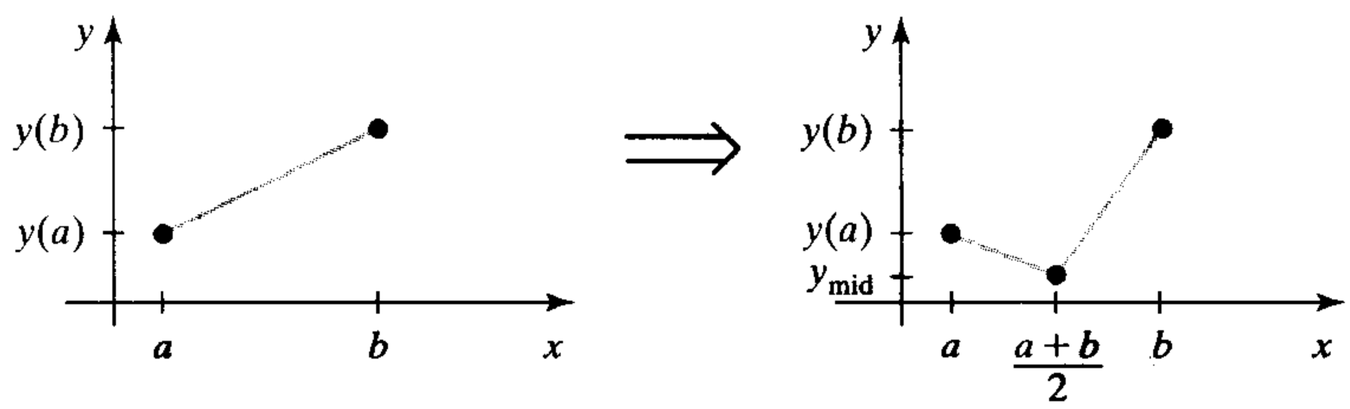


图8.89 直线段的随机中点位移

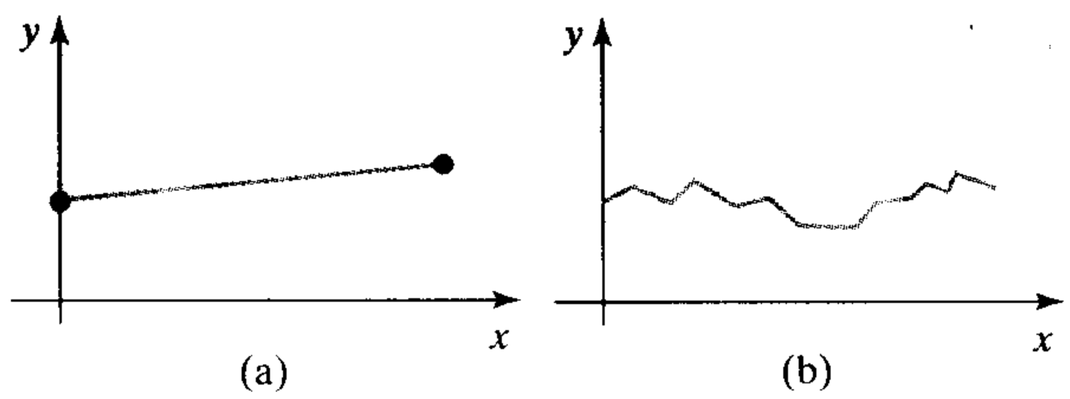


图8.90 一条直线段重复四次随机中点位移过程而生成的随机走动路径

将随机中点位移程序用于矩形地平面（参见图8.91），可以生成地面特征。开始时，对地平面的四个角（图8.91中的**a**、**b**、**c**、**d**）指定一高度z。然后在每边的中点分割地平面，得到5个新网格的点位置：**e**、**f**、**g**、**h**、**m**。地面边上的点**e**、**f**、**g**、**h**上的高度可以按照最近的两个顶点的平均高度加上随机偏移来计算。例如，中点**e**处的高度z_e利用顶点**a**和**b**进行计算，中点**f**处的高度利用顶点**b**和**c**进行计算：

$$z_e=(z_a+z_b)/2+r_e,\qquad z_f=(z_b+z_c)/2+r_f$$

随机值r_e和r_f从0到正比于网格间距$2H$次幂的均方差高斯分布中得到，这里$H=3-D$，$D>2$是表面的分形维数。也可以将随机偏移计算成表面粗糙因子乘以网格间距，再乘以从0到均方差1间的高斯值的查表值。地面中点**m**的高度z_m可以用**e**、**g**或**f**、**h**进行计算。或者，利用四个地面角的指定高度来计算z_m：

$$z_m=(z_a+z_b+z_c+z_d)/4+r_m$$

该过程每次对四个新网格部分重复进行，直到网格间距小于一指定值。

生成高度后就形成了三角形面片。图8.92给出了在第一次分割时形成的8个面片。在每一层递归时，三角形将连续地分成更小的平面片。当分割过程完成后，根据光源位置、其他照明参数值、选定的颜色及表面纹理来绘制面片。

除了地面以外，随机中点位移方法还能应用于景色中的其他部分。例如，可以使用同样的方法来得到波浪的表面特征或地平面上的云彩模型。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

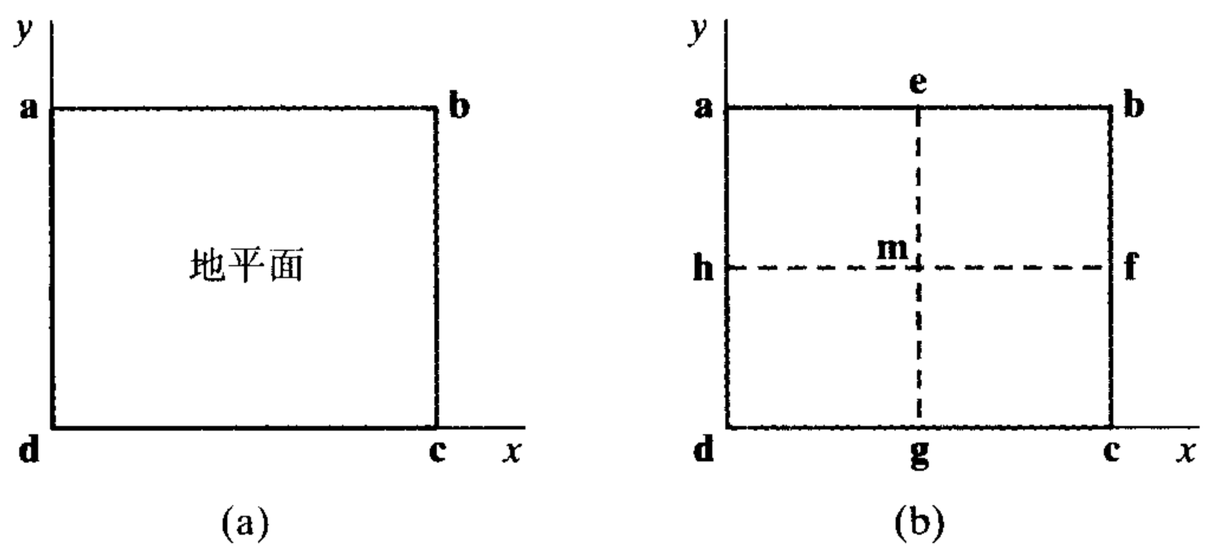


图 8.91　随机中点位移过程用来计算地面高度的第一步，矩形地面(a)分割成四个相等的网格部分(b)

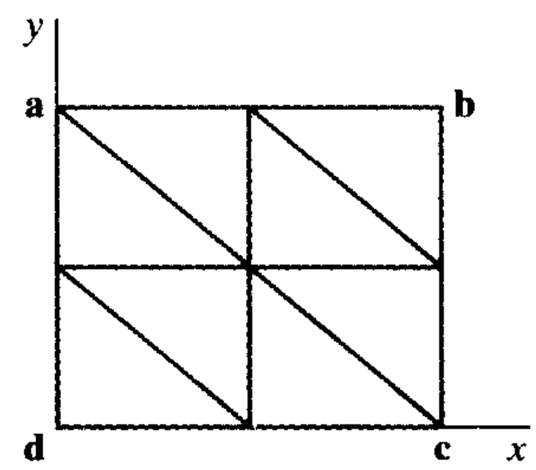


图 8.92　生成地面特征的随机中点位移过程的第一步，形成了地平面上的八个表面片

8.23.8　地面图控制

在使用中点位移方法模拟分形地面场景时，控制山峰和峡谷位移的一个方法，是在地面的不同区域上的某个区间内约束计算的高度。可以通过在地面上建立一组控制面来实现此目的，如图8.93所示。然后，在地面的每一个中点网格位置计算随机高度，该高度依赖于控制高度和该位置处的平均高度之差。此过程在控制地面高度的某一给定区间内约束高度。

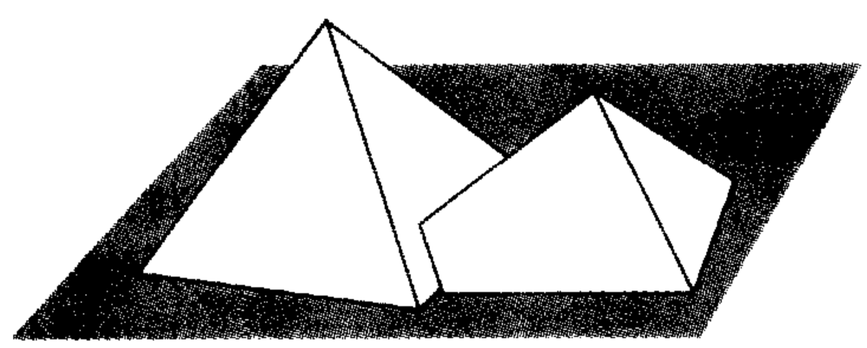


图 8.93　地平面上的控制面

控制面通过使用一特殊区域的轮廓高度来形成平面片，从而模拟落基山脉或其他地区的地面特征。也可以通过建立控制多边形顶点的高度来设计地面特性。控制面可以有任何形状。平面是最容易处理的，但也可以使用球面或其他曲面形状。

我们使用随机中点位移方法来计算网格高度，但现在可以从高斯分布中选取随机值，其中平均值μ和标准方差σ是控制高度的函数。取μ和σ值的一个方法是使它们都正比于一个方差，此方差介于计算出的平均高度和每个网格位置上预定义的控制高度之间。例如，对于图8.91中的网格位置**e**，设平均值和标准方差为

$$\mu_e = zc_e - (z_a + z_b)/2, \qquad \sigma_e = s|\mu_e|$$

其中，zc_e是地面位置**e**处的控制高度，$0 < s < 1$是预定义的缩放因子。较小的s值（$s < 0.1$）生成的地表较平稳，而较大的s值使地面高度有较大起伏。

为了确定一个平面型控制表面上的控制高度值，首先计算平面参数A、B、C和D。对于任意的平面位置(x, y)，包含控制多边形的平面上的高度可以计算为

$$zc = (-Ax - By - D)/C$$

然后，可以使用增量方法来计算地平面网格位置上的控制高度。为了有效地完成这些计算，首先将地平面分割成xy位置的网络，如图8.94所示。然后，将每个多边形控制表面投影到地平面上。可以使用类似于扫描线区域填充的过程，从而确定哪一个网格位置落在控制多边形的投影区内。即对

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

每个穿过多边形边的地平面网格上的y“扫描线”，计算出扫描线交点并确定哪个网格位置在控制多边形的投影区内部。这些方格位置上的控制高度则可以使用增量表示进行计算：

$$zc_{i+1,j} = zc_{i,j} - \Delta x(A/C), \qquad zc_{i,j+1} = zc_{i,j} - \Delta y(B/C) \tag{8.110}$$

其中，Δx和Δy是x和y方向上的网格间距。该过程在使用并行向量方法处理控制表面的网格位置时，计算速度特别快。

图 8.95 显示了使用控制平面构造的地面、水和地平面上云的场景构造。然后使用表面绘制算法来对多边形边进行平滑处理，并且提供恰当的表面颜色。

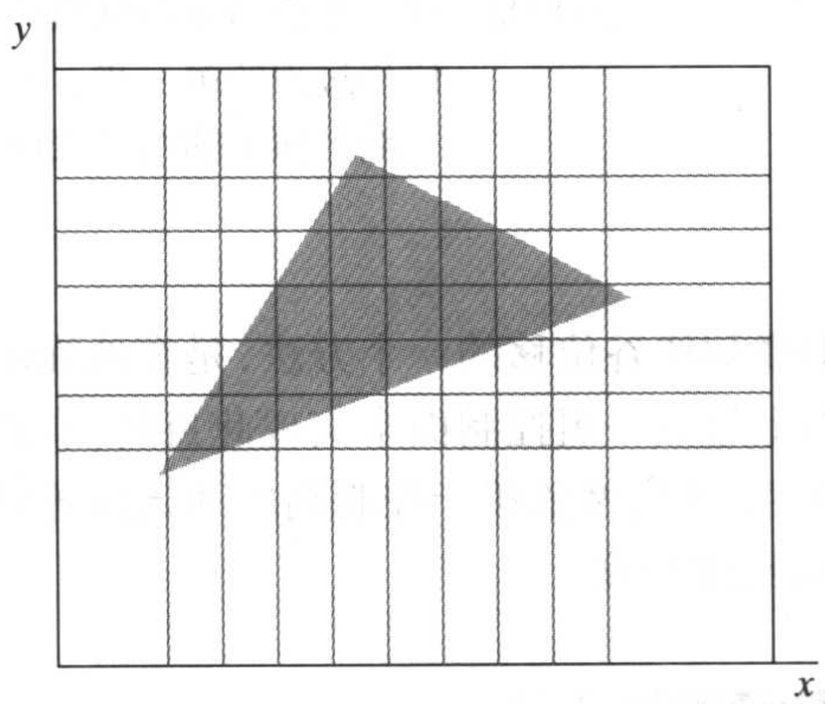


图 8.94 三角控制面在地平面网格上投影

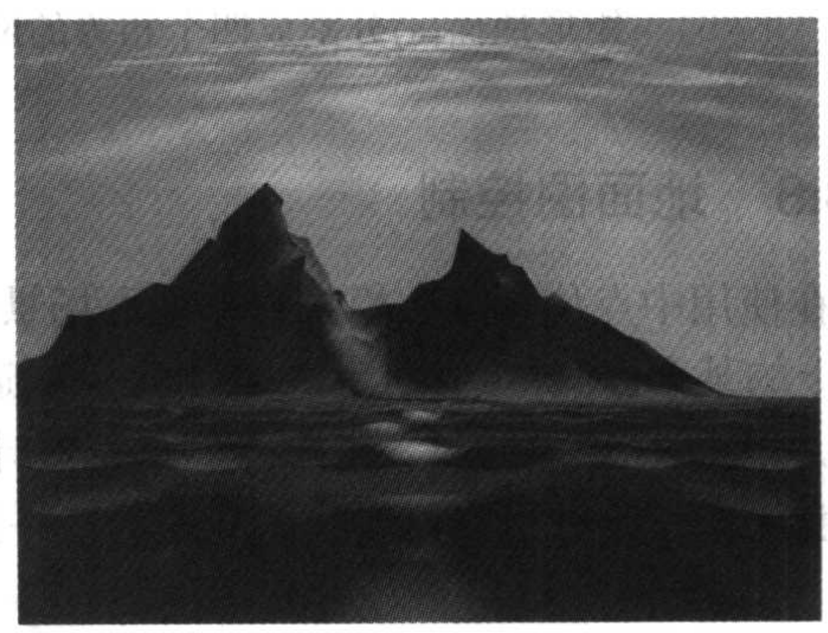


图 8.95 地平面上使用随机中点位移方法和二维控制面模拟的复合场景。分别模拟并绘制地面、水和云的表面特征，然后再组合成复合图形（新加坡信息技术学院的 Eng-Kiat Koh 提供）

8.23.9 自平方分形

生成分形对象的另一种方法是对复数空间上的点重复使用变换函数。在二维平面中，复数可以表示为$z = x + iy$，其中x和y是实数且$i^2 = -1$。在三维和四维空间中，可以使用四元数表示点。复平方函数$f(z)$包括了z^2的计算，可以使用某种自平方函数来生成分形形状。

分形形状取决于迭代的初始点，重复应用自平方函数将产生三种可能的结果之一（参见图 8.96）：

- 变换位置无限发散；
- 变换位置收敛于一极限点，称为吸附点（attractor）；
- 变换位置保持在某对象的边界。

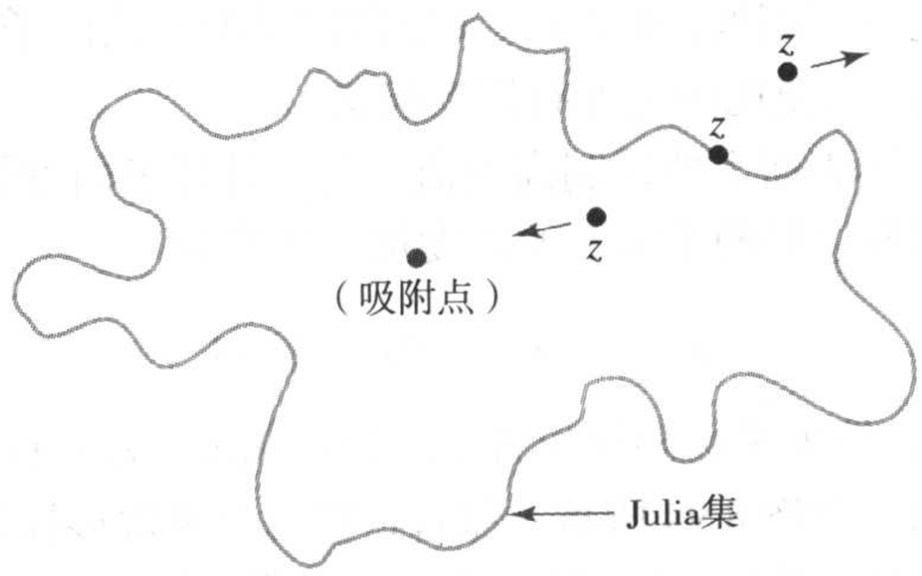


图 8.96 复平面上自平方变换$f(z)$的可能结果，依赖于选定的初始点位置

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

例如，非分形平方运算$f(z)=z^2$根据与单位圆的关系而变换点（参见图 8.97）。其$|z|>1$的点通过一系列点变换后趋于无限。而$|z|<1$的点在变换后将趋于坐标原点。在圆周上的点（$|z|=1$）仍在圆周上。对于某些函数，在趋于无限的点和趋于有限的点之间的边界是分形图形。分形对象的边界称为 Julia 集。

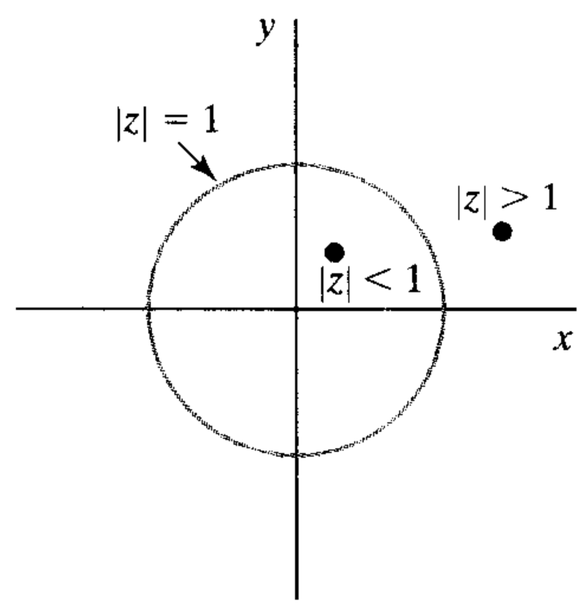


图 8.97　复平面上的单位圆周。非分形复平方函数$f(z)=z^2$将圆内点映射向原点，圆外点远离圆周。圆周上的初始点仍在圆周上

一般情况下，通过测试选定位置的趋向来确定分形边界。若选定的位置无限发散或趋于一吸附点，则可以试一下附近的点。重复这一过程直到最终确定出分形边界上的一个位置。然后，平方变换的迭代操作将生成分形形状。对于复平面上的简单变换，分形曲线上的快速定位方法是使用变换函数的逆函数。然后，曲线内或曲线外的初始点将收敛于分形曲线上的位置（参见图 8.98）。

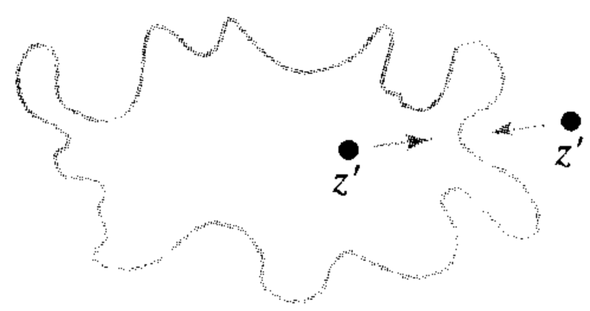


图 8.98　使用自平方函数的逆函数$z=f^{-1}(z')$确定分形边界

分形中用得最多的函数是平方变换，即

$$z' = f(z) = \lambda z(1-z) \tag{8.111}$$

其中，λ是指定的常数复数值。对于该函数，可以使用逆方法来确定分形曲线。首先移项得到二次方程：

$$z^2 - z + z'/\lambda = 0 \tag{8.112}$$

则逆变换是二次式：

$$z = f^{-1}(z') = \frac{1}{2}\left(1 \pm \sqrt{1-(4z')/\lambda}\right) \tag{8.113}$$

使用复数算术操作，求解此方程得到z的实、虚部为

$$x = \mathrm{Re}(z) = \frac{1}{2}\left(1 \pm \sqrt{\frac{|\mathrm{discr}| + \mathrm{Re}(\mathrm{discr})}{2}}\right) \tag{8.114}$$

$$y = \mathrm{Im}(z) = \pm\frac{1}{2}\sqrt{\frac{|\mathrm{discr}| - \mathrm{Re}(\mathrm{discr})}{2}}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

二次式判别式为 discr = 1 − (4z′)/λ。可以计算出 n 个初始 x、y 值（比如说 10 个），并在绘制分形曲线前去掉这些值。由于该函数产生两种可能的变换(x, y)位置，因此只要 Im(discr) ≥ 0，每次可以随机地选择加或者减符号。当 Im(discr) < 0 时，两个可能位置在第二和第四象限。这种情况下，x 和 y 必有相反的符号。下面的程序给出了这个自平方函数的实现，图 8.99 中绘制了两条曲线。

图 8.99 过程 selfSqTransf 使用函数 $f(z) = \lambda z(1 - z)$的逆函数，当(a)$\lambda = 3$ 和 (b) $\lambda = 2 + i$ 时生成两条分形曲线。每个曲线由 10 000 个点绘制

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

/*  Set initial size of display window.  */
GLsizei winWidth = 600, winHeight = 600;

/*  Set coordinate limits in complex plane.  */
GLfloat xComplexMin = -0.25, xComplexMax = 1.25;
GLfloat yComplexMin = -0.75, yComplexMax = 0.75;

struct complexNum
{
   GLfloat x, y;
};

void init (void)
{
   /*  Set color of display window to white.  */
   glClearColor (1.0, 1.0, 1.0, 0.0);
}

void plotPoint (complexNum z)
{
    glBegin (GL_POINTS);
        glVertex2f (z.x, z.y);
    glEnd ( );
}

void solveQuadraticEq (complexNum lambda, complexNum * z)
{
   GLfloat lambdaMagSq, discrMag;
   complexNum discr;
   static complexNum fourOverLambda = { 0.0, 0.0 };
   static GLboolean firstPoint = true;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
   if (firstPoint) {
      /*  Compute the complex number: 4.0 divided by lambda.  */
      lambdaMagSq = lambda.x * lambda.x + lambda.y * lambda.y;
      fourOverLambda.x =  4.0 * lambda.x / lambdaMagSq;
      fourOverLambda.y = -4.0 * lambda.y / lambdaMagSq;
      firstPoint = false;
   }
   discr.x = 1.0 - (z->x * fourOverLambda.x - z->y * fourOverLambda.y);
   discr.y = z->x * fourOverLambda.y + z->y * fourOverLambda.x;
   discrMag = sqrt (discr.x * discr.x + discr.y * discr.y);

   /*  Update z, checking to avoid the sqrt of a negative number.  */
   if (discrMag + discr.x < 0)
      z->x = 0;
   else
      z->x = sqrt ((discrMag + discr.x) / 2.0);

   if (discrMag - discr.x < 0)
      z->y = 0;
   else
      z->y = 0.5 * sqrt ((discrMag - discr.x) / 2.0);

   /*  For half the points, use negative root,
    *  placing point in quadrant 3.
    */
   if (rand ( ) < RAND_MAX / 2) {
      z->x = -z->x;
      z->y = -z->y;
   }

   /*  When imaginary part of discriminant is negative, point
    *  should lie in quadrant 2 or 4, so reverse sign of x.
    */
   if (discr.y < 0)
      z->x = -z->x;

   /* Complete the calculation for the real part of z. */
   z->x = 0.5 * (1 - z->x);
}

void selfSqTransf (complexNum lambda, complexNum z, GLint numPoints)
{
   GLint k;

   /*  Skip the first few points.  */
   for (k = 0;  k < 10;  k++)
      solveQuadraticEq (lambda, &z);

   /*  Plot the specified number of transformation points.  */
   for (k = 0;  k < numPoints;  k++) {
      solveQuadraticEq (lambda, &z);
      plotPoint (z);
   }
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void displayFcn (void)
{
   GLint numPoints = 10000;          // Set number of points to be plotted.
   complexNum lambda = { 3.0, 0.0 }; //  Set complex value for lambda.
   complexNum z0 = { 1.5, 0.4 };     //  Set initial point in complex plane.

   glClear (GL_COLOR_BUFFER_BIT);    //  Clear display window.

   glColor3f (0.0, 0.0, 1.0);        //  Set point color to blue.

   selfSqTransf (lambda, z0, numPoints);
   glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   /*  Maintain an aspect ratio of 1.0, assuming that
    *  width of complex window = height of complex window.
    */
   glViewport (0, 0, newHeight, newHeight);

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );

   gluOrtho2D (xComplexMin, xComplexMax, yComplexMin, yComplexMax);

   glClear (GL_COLOR_BUFFER_BIT);
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (50, 50);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Self-Squaring Fractal");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);

   glutMainLoop ( );
}
```

自平方函数$f(z) = \lambda z(1 - z)$中的变量x、y和λ绘制的三维图形（这里$|\lambda| = 1$）由图8.100给出。此图例的每个剖面片是复平面上的分形曲线。

另一个平方操作产生了分形形状的变形，即稍微修改一下z^2变换。这是在下列平方变换下不发散的复数值z的集合，平方变换为

$$\begin{aligned} z_0 &= z \\ z_k &= z_{k-1}^2 + z_0 \qquad k = 1, 2, 3, \ldots \end{aligned} \tag{8.115}$$

即首先在复平面上选一点z，然后计算变换位置$z^2 + z$。下一步，求变换位置的平方并加上原z值。重复此步骤直到可以确定变换是否发散。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图 8.100　函数 $f(z) = \lambda z(1 - z)$在三维中的绘图，其规范化的 λ 值沿垂直轴变化（IBM 研究所的 Alan Norton 提供）

数学家早已弄清了这种平方函数的奇异特性。但没有计算机的帮助，这些函数很难模拟。数字计算机诞生以后，在行式打印机上绘制了变换（8.115）的收敛边界。随着数字计算机性能的不断提高，有可能进一步研究这个函数的图形特性。随后，Benoit Mandelbrot 使用更复杂的图形技术广泛地研究了这个函数，而在变换（8.115）中不发散的点集成为著名的 Mandelbrot 集。

为了实现变换（8.115），首先在复平面上选一个矩形区域。然后，将该矩形区域内的位置映射到选定的视频监视器显示窗口中颜色编码的像素位置（参见图8.101）。变换（8.115）根据复平面上相应点的偏差率来选择像素颜色。如果复数大于2，则在自平方算法下的结果将很快地发散。因此，建立一条路径来重复平方算法，一直到复数的模大于2或者达到一预定的迭代数。迭代的最大次数依赖于我们要显示的细节数量和要绘制的点数。通常取 100 和 1000 之间的某个值，尽管较低的值可以加快计算速度。但降低迭代次数，的确会失去收敛区域边界（Julia 集）的某些细节。在循环的末尾，根据循环执行的迭代次数来选择颜色值。例如，若迭代计数达到最大值时（不发散点）像素为黑色，当迭代计算接近0时像素为红色。其他颜色值可以根据从0到最大数之间的迭代值来选择。通过选择不同的颜色映射和复平面的不同部分，可以生成许多围住非发散点的分形边界附近的生动显示。对围绕 Mandelbrot 集的区域中的像素位置进行颜色编码的一个选择如图 8.102(a)所示。

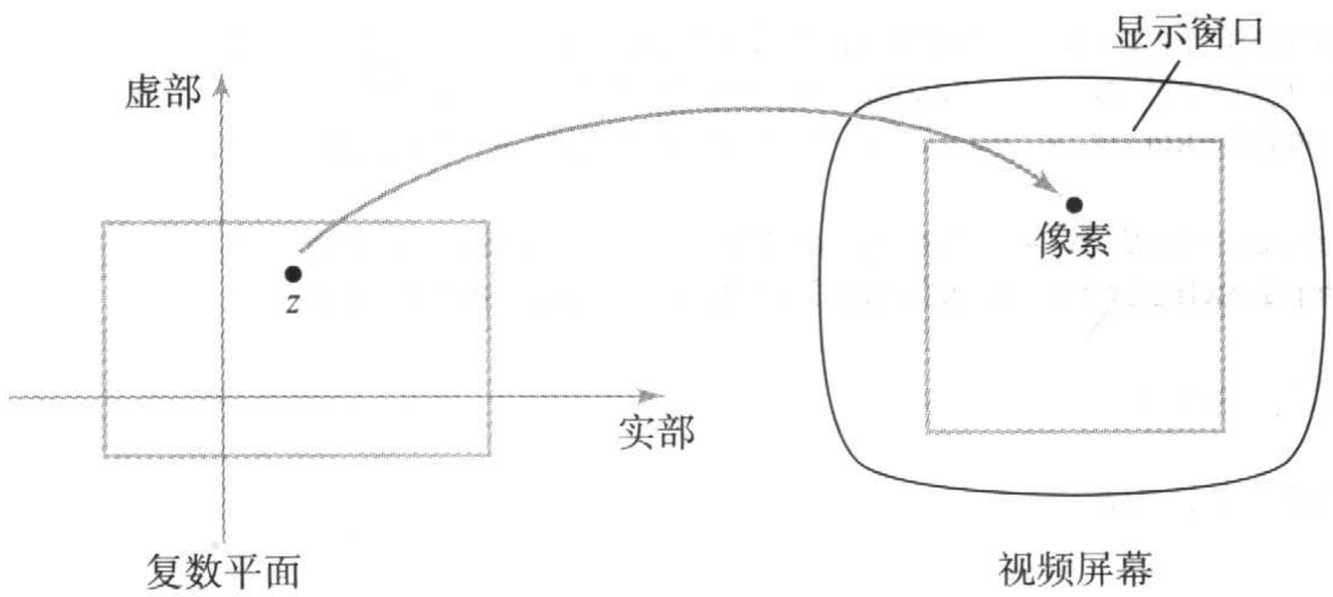


图 8.101　将复平面里矩形区域的位置映射到视频监视器上彩色编码的像素位置

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

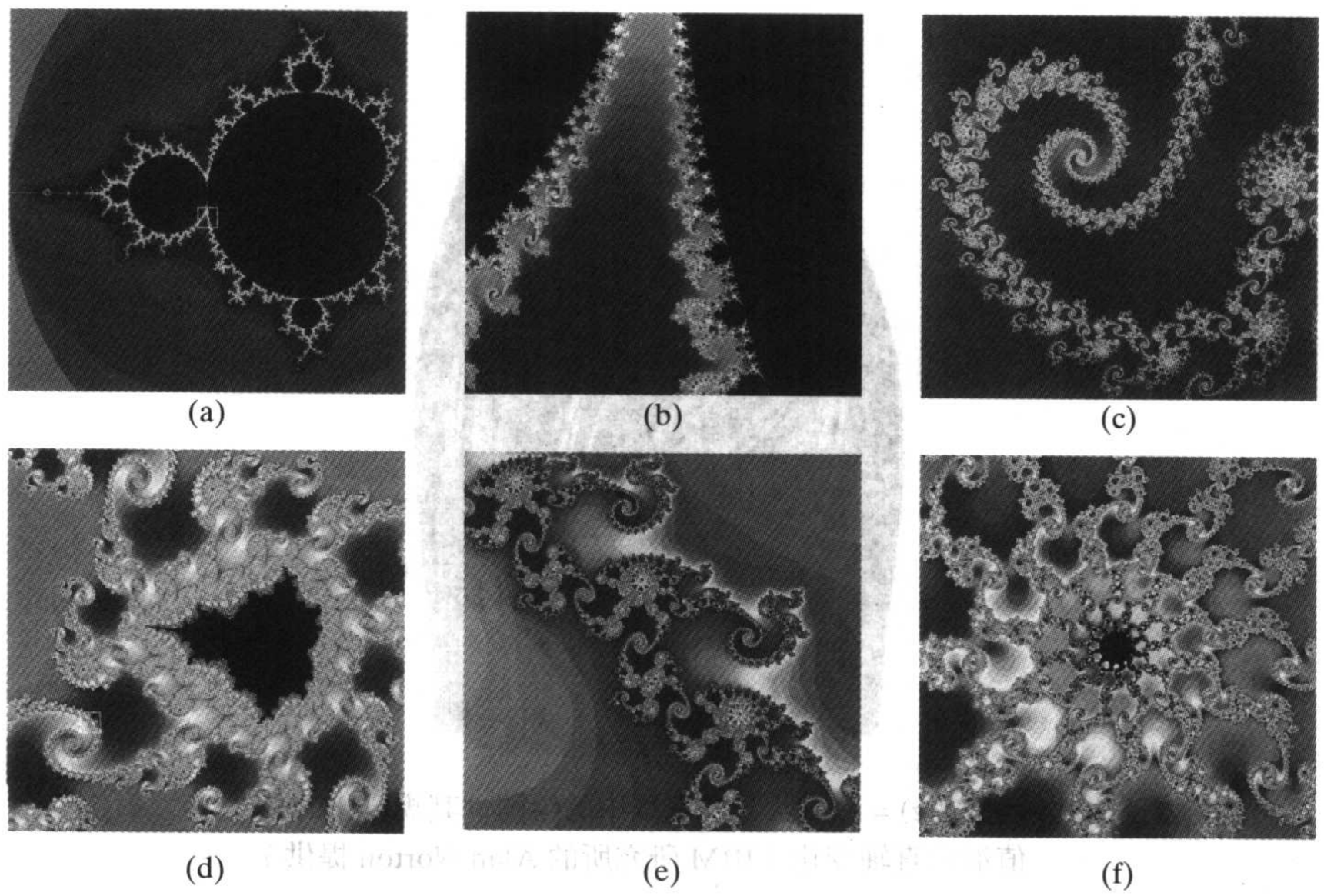


图 8.102　变换（8.115）分形边界的放大显示。从 Mandelbrot 集的一个显示开始，(a)中的黑色区域及其周围区域，放大选定边界区域(b)到(f)。白色方框表示选定的下一个要放大的窗口（由 Vanderbilt 大学的 Brian Evans 提供）

下列程序给出了显示该收敛点集及其边界的变换（8.115）的实现。收敛点集的主要部分包括在复平面的下列区域中。

$$-2.00 \leqslant \mathrm{Re}(z) \leqslant 0.50$$

$$-1.20 \leqslant \mathrm{Im}(z) \leqslant 1.20$$

通过在复平面上选择更小的矩形区域并放大所选区域的图示，可以发现沿该Mandelbrot集合边界上的细节。图8.102给出围绕收敛集区域的颜色编码显示和展示该平方变换的某些著名特征的一组放大显示。

```
#include <GL/glut.h>

/*  Set initial size of the display window.  */
GLsizei winWidth = 500, winHeight = 500;

/*  Set limits for the rectangular area in complex plane.  */
GLfloat xComplexMin = -2.00, xComplexMax = 0.50;
GLfloat yComplexMin = -1.25, yComplexMax = 1.25;

GLfloat complexWidth = xComplexMax - xComplexMin;
GLfloat complexHeight = yComplexMax - yComplexMin;

class complexNum {
   public:
      GLfloat x, y;
};

struct color { GLfloat r, g, b; };

void init (void)
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
{
   /*  Set display-window color to white.  */
   glClearColor (1.0, 1.0, 1.0, 0.0);
}

void plotPoint (complexNum z)
{
    glBegin (GL_POINTS);
        glVertex2f (z.x, z.y);
    glEnd ( );
}

/*  Calculate the square of a complex number.  */
complexNum complexSquare (complexNum z)
{
   complexNum zSquare;

   zSquare.x = z.x * z.x - z.y * z.y;
   zSquare.y = 2 * z.x * z.y;
   return zSquare;
}

GLint mandelSqTransf (complexNum z0, GLint maxIter)
{
   complexNum z = z0;
   GLint count = 0;

   /*  Quit when z * z > 4  */
   while ((z.x * z.x + z.y * z.y <= 4.0) && (count < maxIter)) {
      z = complexSquare (z);
      z.x += z0.x;
      z.y += z0.y;
      count++;
   }
   return count;
}

void mandelbrot (GLint nx, GLint ny, GLint maxIter)
{
   complexNum z, zIncr;
   color ptColor;

   GLint iterCount;

   zIncr.x =  complexWidth / GLfloat (nx);
   zIncr.y =  complexHeight / GLfloat (ny);

   for (z.x = xComplexMin;  z.x < xComplexMax;  z.x += zIncr.x)
      for (z.y = yComplexMin;  z.y < yComplexMax;  z.y += zIncr.y) {
         iterCount = mandelSqTransf (z, maxIter);
         if (iterCount >= maxIter)
            /*  Set point color to black.  */
            ptColor.r = ptColor.g = ptColor.b = 0.0;
            else if (iterCount > (maxIter / 8)) {
                    /*  Set point color to orange.  */
                    ptColor.r = 1.0;
                    ptColor.g = 0.5;
                    ptColor.b = 0.0;
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
                }
                else if (iterCount > (maxIter / 10)) {
                        /*  Set point color to red.  */
                        ptColor.r = 1.0;
                        ptColor.g = ptColor.b = 0.0;
                    }
                    else if (iterCount > (maxIter /20)) {
                            /*  Set point color to dark blue.  */
                            ptColor.b = 0.5;
                            ptColor.r = ptColor.g = 0.0;
                        }
                        else if (iterCount > (maxIter / 40)) {
                                /*  Set point color to yellow.  */
                                ptColor.r = ptColor.g = 1.0;
                                ptColor.b = 0.0;
                            }
                            else if (iterCount > (maxIter / 100)) {
                                /*  Set point color to dark green.  */
                                    ptColor.r = ptColor.b = 0.0;
                                    ptColor.g = 0.3;
                                }

                                else {
                                      /*  Set point color to cyan.  */
                                      ptColor.r = 0.0;
                                      ptColor.g = ptColor.b = 1.0;
                                }
            /*  Plot the color point.  */
                glColor3f (ptColor.r, ptColor.g, ptColor.b);
                plotPoint (z);
            }
        }
        void displayFcn (void)
        {
            /*  Set number of x and y subdivisions and the max iterations.  */
            GLint nx = 1000, ny = 1000, maxIter = 1000;

            glClear (GL_COLOR_BUFFER_BIT);   //  Clear display window.

            mandelbrot (nx, ny, maxIter);
            glFlush ( );
        }

        void winReshapeFcn (GLint newWidth, GLint newHeight)
        {
            /*  Maintain an aspect ratio of 1.0, assuming that
             *  complexWidth = complexHeight.
             */
            glViewport (0, 0, newHeight, newHeight);

            glMatrixMode (GL_PROJECTION);
            glLoadIdentity ( );

            gluOrtho2D (xComplexMin, xComplexMax, yComplexMin, yComplexMax);

            glClear (GL_COLOR_BUFFER_BIT);
        }
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (50, 50);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Mandelbrot Set");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);

   glutMainLoop ( );
}
```

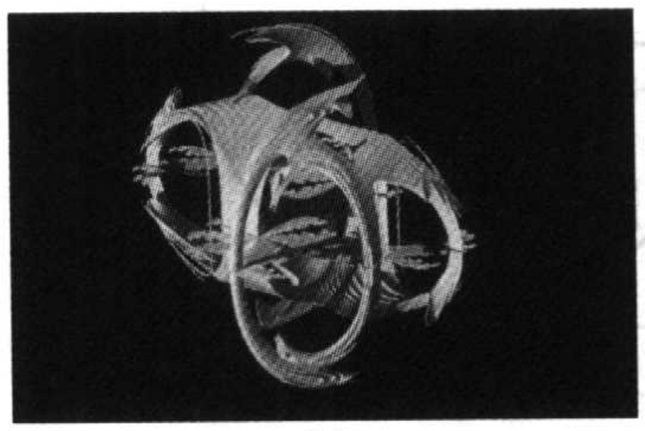
可以扩充如等式（8.111）所示的复函数变换，从而产生分形曲面和分形实体。生成这些对象的方法是使用四元数（quaternion）表示（参见附录 A）来变换三维和四维空间中的点。四元数有 4 个元素，1 个实部和 3 个虚部，可以表示为复平面上复数概念的扩充：

$$q = s + ia + jb + kc \tag{8.116}$$

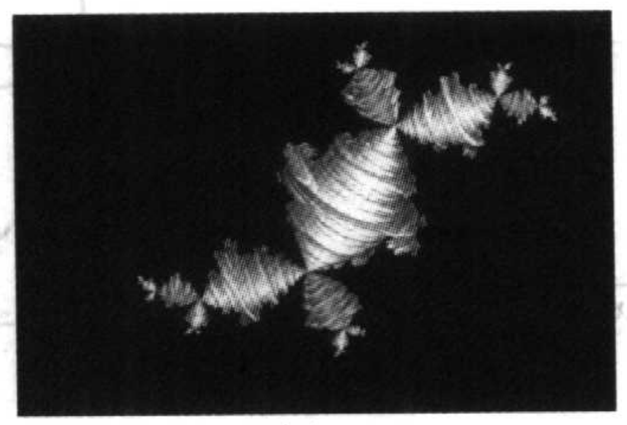
其中，$i^2 = j^2 = k^2 = -1$。实部 s 也称为四元数的标量部分（scalar part），虚部称为四元数的向量部分（vector part），$\mathbf{v} = (a, b, c)$。

使用附录 A 中讨论的四元素乘法和加法规则，可以使用自平方函数和其他迭代方法来生成分形对象的曲面。一个基本过程是从复数空间内部的一点开始，生成后继点直到确认发散和不发散位置的边界。例如，如果先定位一个不发散（内部）位置，然后检测该位置邻域中的点，直到发现发散（外部）点。先前的内部点保留下来作为边界曲面位置。然后检测这些曲面点的邻域以确定它们是否为内部（收敛）或外部（发散）。任何连接外部点的内部点是曲面点。在这种情况下，该过程在沿分形边界离曲面不远的地方继续进行。在生成四维分形后，三维面片投影到监视器的二维平面中。

四维空间中生成自平方分形的过程，需要大量时间来求迭代函数值和测试收敛或发散的位置。曲面上的每一点可以表示为小立方体，给出曲面的内部和外部范围。分形的三维投影程序的输出一般包含了超过一百万个的曲面立方体顶点。分形对象的显示则通过确定每个曲面立方体的光线和颜色的照明方法来完成。然后使用隐面消除方法从而仅显示对象的可见面。图 8.103 和图 8.104 给出了投影到三维空间的自平方四维分形。



(a)



(b)

图 8.103　四维分形的三维投影，由自平方四元数函数 $f(q) = \lambda q(1 - q)$ 生成，使用(a)$\lambda = 1.475 + 0.9061i$，(b)$\lambda = -0.57 + i$（IBM 研究所的 Alan Norton 提供）

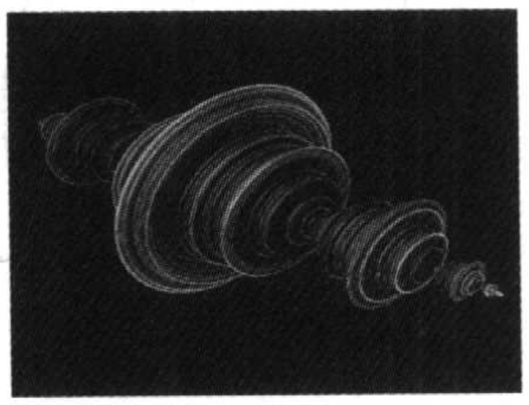


图 8.104　四维物体的三维表面投影，由自平方四元数函数 $f(q) = q^2 - 1$ 生成（IBM 研究所的 Alan Norton 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.23.10 自逆分形

可以使用多种几何逆变换生成分形形状。我们还是从初始点集开始，重复使用非线性逆算法将初始点变换到分形点。

作为一个例子，考察相对于圆的二维逆变换，圆半径为 r，中心在 $\mathbf{P}_c = (x_c, y_c)$ 位置。圆外任意一点 $\mathbf{P}$ 到圆内一点 $\mathbf{P}'$（参见图 8.105）的变换为

$$(\overline{\mathbf{P}_c\mathbf{P}})(\overline{\mathbf{P}_c\mathbf{P}'}) = r^2 \tag{8.117}$$

这里 $\mathbf{P}$ 和 $\mathbf{P}'$ 位于经过圆心 $\mathbf{P}_c$ 的直线上。也可以使用等式（8.117）对圆内的位置进行变换。某些内部位置变换到外部位置，而另一些内部位置仍变换到内部位置。

若两点的坐标是 $\mathbf{P} = (x, y)$、$\mathbf{P}' = (x', y')$，可以把等式（8.117）写为

$$[(x - x_c)^2 + (y - y_c)^2]^{1/2}[(x' - x_c)^2 + (y' - y_c)^2]^{1/2} = r^2$$

由于两点连线通过圆心，我们有 $(y - y_c) / (x - x_c) = (y' - y_c) / (x' - x_c)$。因此，位置 $\mathbf{P}'$ 的变换坐标值是

$$x' = x_c + \frac{r^2(x - x_c)}{(x - x_c)^2 + (y - y_c)^2}, \qquad y' = y_c + \frac{r^2(y - y_c)}{(x - x_c)^2 + (y - y_c)^2} \tag{8.118}$$

因此，圆外的点映射到圆周位置，远距离的点（$\pm\infty$）映射到圆心。反过来，圆心附近的点映射到圆外远处的点。从圆心往外移动时，点映射到圆外靠近圆周的位置。而圆内靠近圆周的点变换到圆心附近。例如，对以 $(x_c, y_c) = (0, 0)$ 为圆心的圆来说，在 r 到 $+\infty$ 范围内的外部 x 值映射到 $r/2$ 到 0 范围中的 x' 值。对一个以原点为圆心的圆来说，在 $r/2$ 到 0 范围内的 x 映射到 r 到 $+\infty$ 范围内的 x' 值，而在 $r/2$ 到 r 范围内的点变换到 r 到 $r/2$ 范围中的值。对 x 的负值可得类似的结果。

我们可将此变换应用于各种对象，如直线段、圆或椭圆。通过圆心的直线段在此逆变换下是不变的；它映射到自身。但不通过圆心的直线段逆转到以中心点 $\mathbf{P}_c$ 作为圆上一点的一个圆上。而任一通过参考圆中心的圆逆转成一条不经过该圆心的直线段。如果该圆与参考圆不相交，则它逆转到另一个圆，如图 8.106 所示。另一个不变逆转是与参考圆正交的圆的变换，即在交点处两圆切线互相垂直。

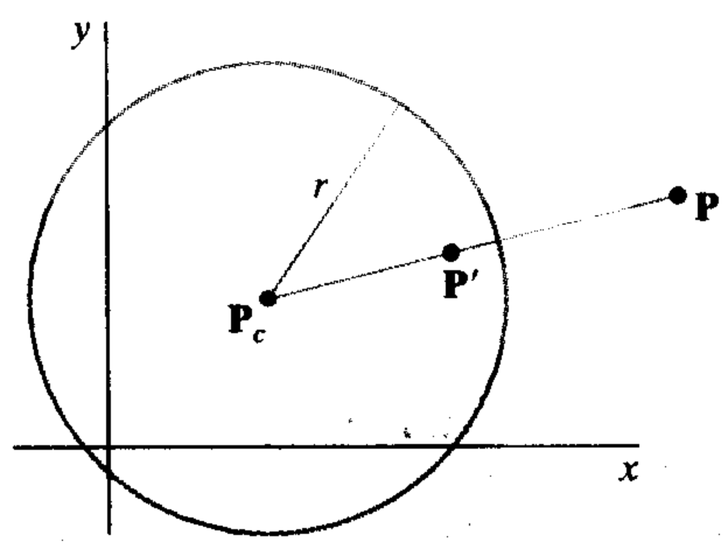


图 8.105 将 $\mathbf{P}$ 变到圆内点 $\mathbf{P}'$，其圆半径为 r

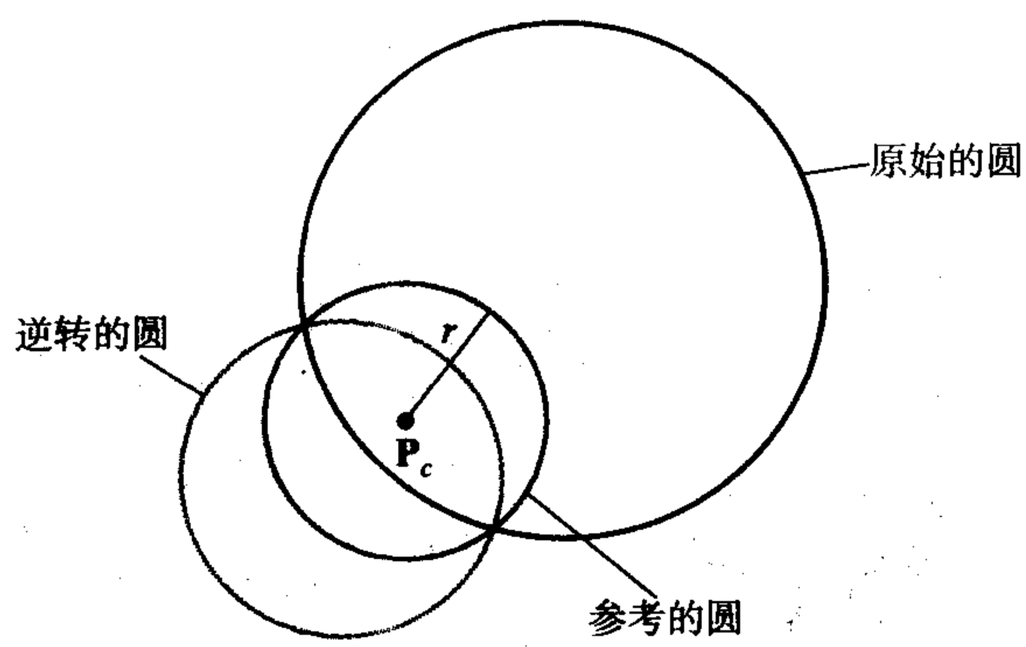


图 8.106 不通过参考圆原点的圆的逆变换

在逆变换下，从一组圆开始，重复使用不同参考圆的变换，可以生成各种分形形状。同样，可以对一组直线使用圆周逆变换。对其他对象也可以开发类似的逆方法。我们可以将该过程推广到其他二维形状。且可生成球面、平面或三维对象。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.24　形状语法和其他过程方法

已经开发了很多其他的过程性方法来生成对象形态或曲面细节层次。**形状语法**（shape grammar）是一组产生式规则，可以应用到初始对象，从而增加与原始形状相协调的细节层次。使用变换可以改变对象的几何结构（形状），使用变换规则可以增加表面颜色或表面纹理细节。

给定一组产生式规则，形状设计者可以在给定初始对象到最终对象结构的每一次变换中应用不同的规则。图 8.107 表示了变换三角形状的四条几何替代规则。这些规则的几何变换可以由系统根据算法写出，而这些输入图形则由产生式规则编辑器画出。即每个规则可以通过显示初始和最终形状而图形化地描述，然后使用数学方法或其他具有绘图功能的编程语言来建立实现过程。

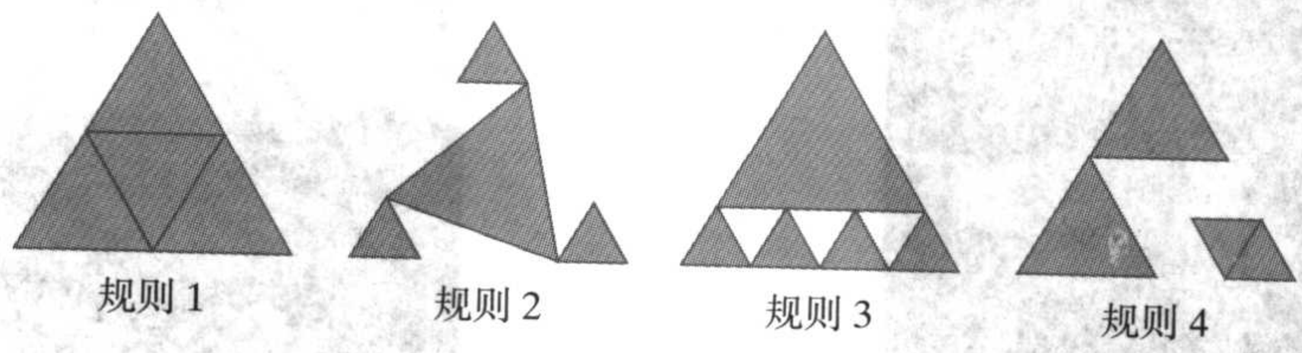


图 8.107　细分和变换等边三角形形状的四条几何替换规则

图 8.107 中的几何替换规则的另一个应用由图 8.108 给出，其中图 8.108(d)通过对图 8.108(a)中的初始三角形使用后面的四条规则而获得。图8.109表示了利用三角形替换规则生成的另一个形状。

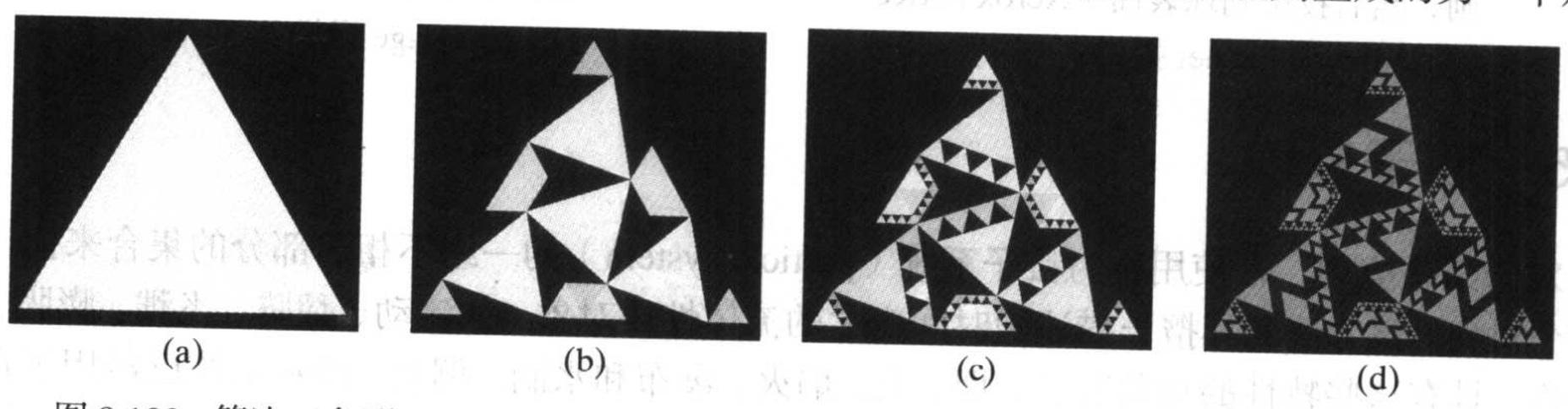


图 8.108　等边三角形(a)用图 8.107 中的替换规则 1 和 2 变到形状(b)。然后使用规则 3 将(b)变到(c)，再用规则 4 变到(d)（Xerox PARC 的 Andrew Glassner 提供）

三维形状和表面特性使用类似的算法进行变换。图8.110给出了应用于多面体的几何替代规则。图 8.111 中显示对象的初始形状是一个 20 面体，即有 20 个侧面的多面体。几何替换应用于 20 面体的侧平面上，生成的多边形顶点投影到一封闭球面的表面上。

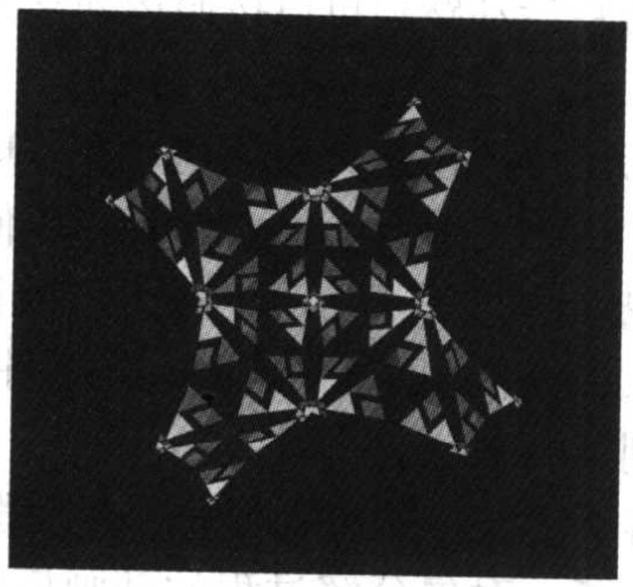


图 8.109　由变换三角形形状的几何替换规则生成的一个设计（Xerox PARC 的 Andrew Glassner 提供）

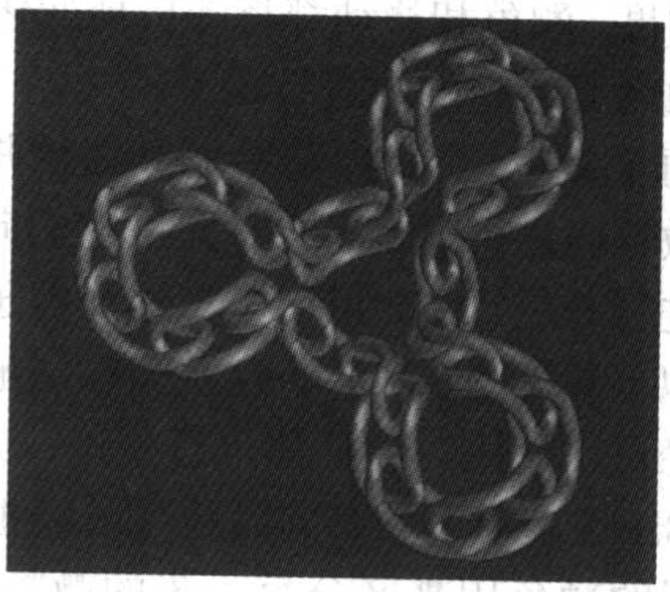


图 8.110　由变换菱形形状的几何替换规则生成的一个设计。该设计的初始形状是 Rubik's Snake 的表示结果（Xerox PARC 的 Andrew Glassner 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

利用产生式规则来描述对象形状的另一个例子是L语法或嫁接（graftals）。这些规则提供了显示植物的一种方法。例如，树的拓扑结构可以描述成一个树干，其上附有枝和叶。然后，给出独立分支上枝和叶的特殊连接规则来模拟树。该过程将对象结构放置在特殊坐标位置而给出几何描述。

图8.112表示了由商用植物生成器软件包构造的含有各种树和植物的场景，构造植物的软件程序基于植物学法则。

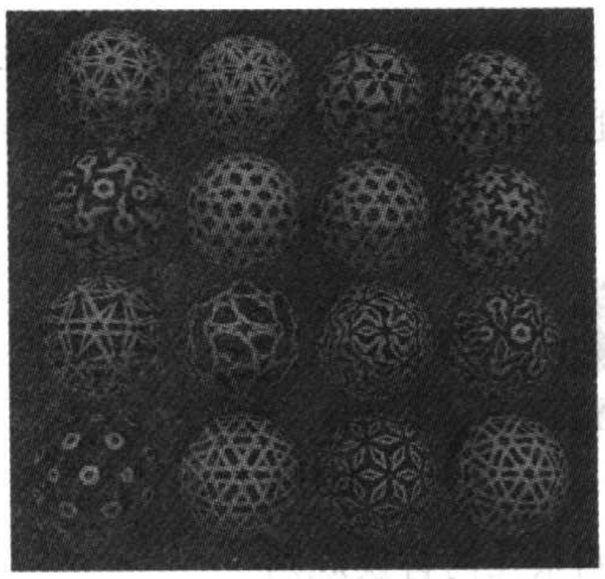


图 8.111　在球表面上生成的设计，使用了用于二十面体中侧平面上的三角形替换规则，然后投影到球表面（Xerox PARC 的 Andrew Glassner 提供）

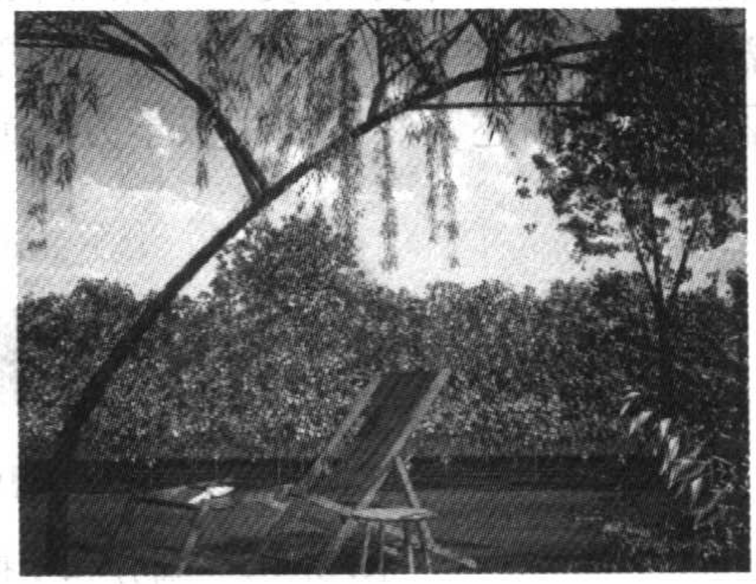


图 8.112　使用 TDI-AMAP 软件包生成的真实场景，由超过 100 种的各种树和植物组成，使用了基于植物学法则的程序（Thomson Digital Image 提供）

8.25　粒子系统

对于某些应用，常使用称为**粒子系统**（particle system）的一组不相连部分的集合来描述一个或多个对象。该方法尤其擅长描述随时间变化的流体性质对象，如流动、翻腾、飞溅、膨胀、内聚或爆炸。具有这些特性的对象有云、烟、火、焰火、瀑布和水滴。例如，微粒系统已经用于在动画 *Star Trek Ⅱ: The Wrath of Khan* 中模拟引起“起源爆炸”的星体爆炸和火焰喷发。粒子系统方法已用于对其他一些对象建模，包括草丛。

在一般应用中，粒子系统在某个空间区域定义，然后应用随机过程随时间而改变系统参数。这些系统参数包括单个粒子的运动路径及它们的颜色和形状。在某个随机时间，将删除每一个对象。

微粒形状可以是小球、椭球、立方体或其他形状，微粒的大小和形状随时间变化。其他性质如微粒透明度、颜色和移动都将随机地变化。在某些应用中，微粒运动路径可以按照运动学方式描述或由重力场等给定的力进行定义。

在每个微粒运动时，将绘制其路径并以特殊颜色进行显示。例如，焰火可以通过在一球域内随机生成微粒而进行显示，并且允许它们向外快速移动，如图 8.113 中所示。微粒路径可以使用红色到黄色进行着色，例如可以模拟爆炸粒子的温度。同样，草丛的真实感显示可以利用“轨道”微粒进行模拟（参见图 8.114），这些微粒从地面上射出，并在重力作用下回落到地面。在这种情况下，微粒路径可以使用一卷柱体进行模拟，并且可以使用从绿到黄的颜色。

图 8.115 表示了瀑布的微粒系统模拟。水滴从一高度落下，被一障碍物偏移，然后散开到地面。不同的颜色用来区分每一步的微粒路径。图 8.116 表示了一对象分解过程的动画模拟。左边的对象分解成右边的微粒状况。图 8.117 给出了由多种表示形成的复合场景，利用微粒系统表示的草丛、分形山、纹理映射和其他表面绘制程序来模拟此场景。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

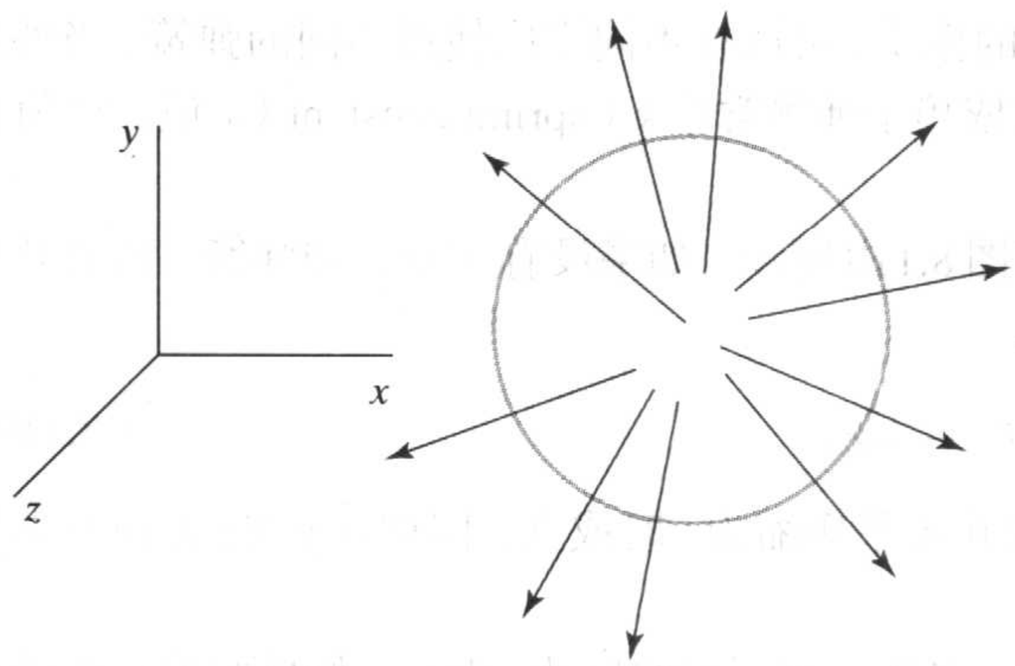


图 8.113　利用微粒系统模拟焰火，微粒迅速地由球内向外发射

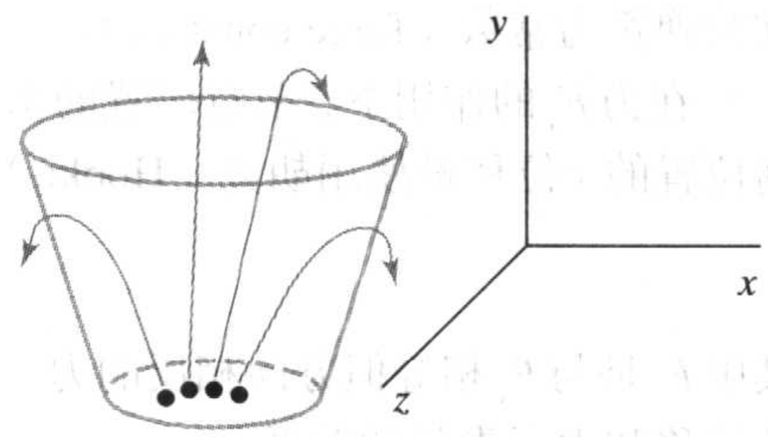


图 8.114　发射向上微粒，用锥形圆柱体来模拟草丛。由于向下重力，微粒路径是抛物线

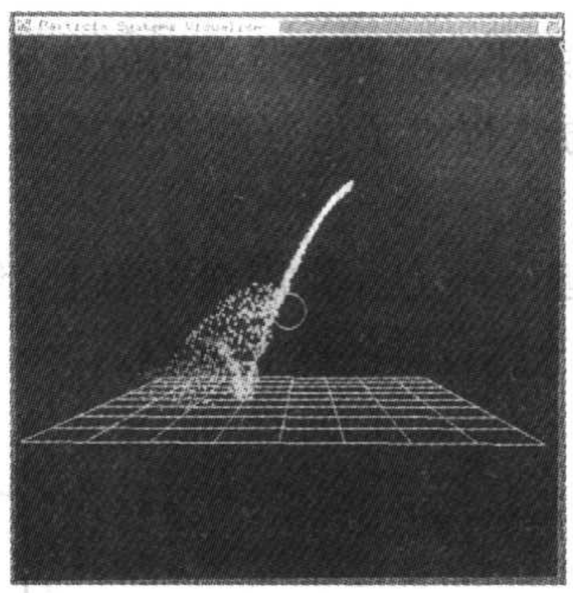


图 8.115　瀑布撞击一石头（圆形）的效果模拟。水被石头阻挡，然后散落到地面（曼彻斯特大学计算机科学系的 M.Brooks 和 T.L.J.Howard 提供）

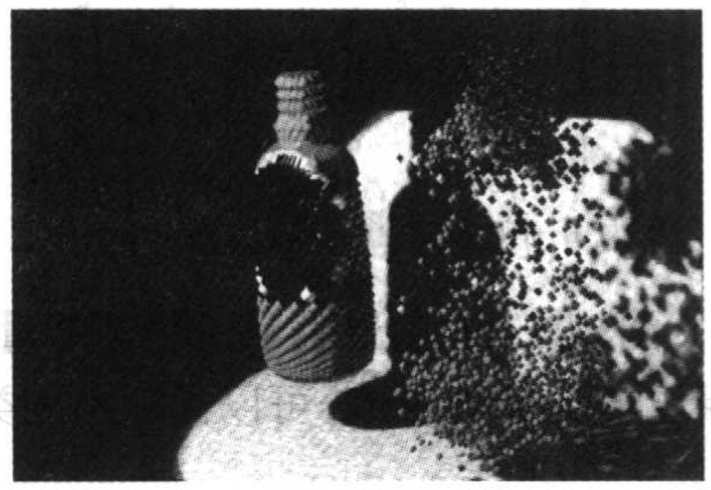


图 8.116　一物体分解成一团微粒（Autodesk 公司提供）

图 8.117　一场景，标题为 Road to Point Reyes，表示了微粒系统草、分形山和纹理映射表面（Pixar 提供）

8.26　基于物理的方法

非刚性对象，如绳、布或软橡皮球可以使用**基于物理的建模**（physically based modeling）方法进行表示。此方法描述了物体在内外力相互作用下的行为。将毛巾织物围在椅子背后的形状的精确描述，则通过考虑织物围绕椅子的效果以及织物和线的相互作用而获得。

模拟非刚性对象的一个普遍方法，是用一组网络节点来逼近对象，而且节点间带有柔性连接。一个简单的连接类型是弹簧。图 8.118 表示了二维弹簧网络，用来逼近一块布或一块橡皮的行为。在三维空间中可建立类似的弹簧网络来模拟一橡皮球或胶状块。对于均质物质，可以使用相同的弹

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

簧贯穿网络。如果需要对象在不同方向上有不同的性质，可以在不同方向使用不同的弹簧。当外力作用于弹簧网络时，单个弹簧延伸或压缩的程度依赖于弹簧常数 k（spring constant k）值，该值也称为弹簧力常数（force constant）。

在力 F_x 的作用下，节点位置的水平位移 x 如图 8.119 所示。如果没有过分拉伸弹簧，则离开平衡位置的 x 位移量使用胡克（Hooke）定律计算：

$$F_s = -F_x = -kx \tag{8.119}$$

其中 F_s 是与 F_x 相等但方向相反的力。这一关系也在水平压缩为 x 时成立，同样在 y 和 z 方向有类似的位移和力元素间的关系。

如果对象完全是柔性的，那么当外力消失时，将恢复到原来的形状。但是，如果要模拟粘性材料或其他可变形对象，则需要修改弹簧特性，从而使弹簧在外力消失时不回到原来位置。然后作用力可以在其他途径上改变对象形状。

不使用弹簧，也可以使用弹性材料来模拟两节点间的连接。然后在外力影响下，利用最小化张量能量函数来决定对象形状。这一方法提供了更好的织物模型，而且设计了各种能量函数来描述不同布料的效果。

为了模拟一非刚性对象，首先建立外力并作用在对象上。然后，考虑贯穿对象网络的作用力传递。因此必须建立确定贯穿网络的节点位移的联立方程。

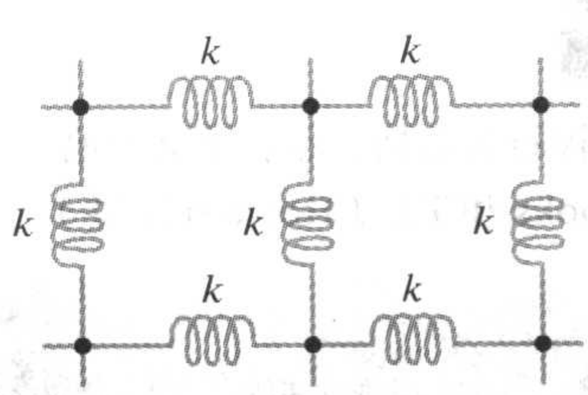


图 8.118 二维弹簧网络，由相同的常数 k 构造

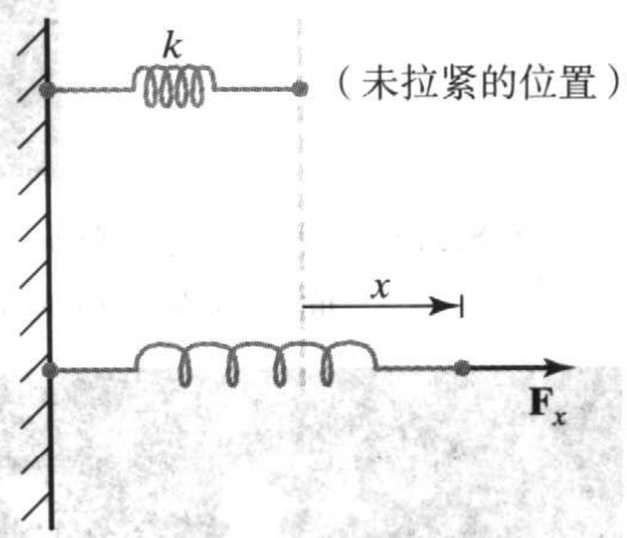


图 8.119 外力 F_x 拉住弹簧一端，另一端固定

图 8.120 表示了利用弹簧网络模拟的香蕉剥皮过程。图 8.121 的场景表示了利用能量函数模拟布的例子，每一种布上使用一种纹理映射模式。调整能量函数计算的参数，可以模拟不同类型的布。图 8.122 模拟了披在桌上的棉质、毛质和塑料材料的布。

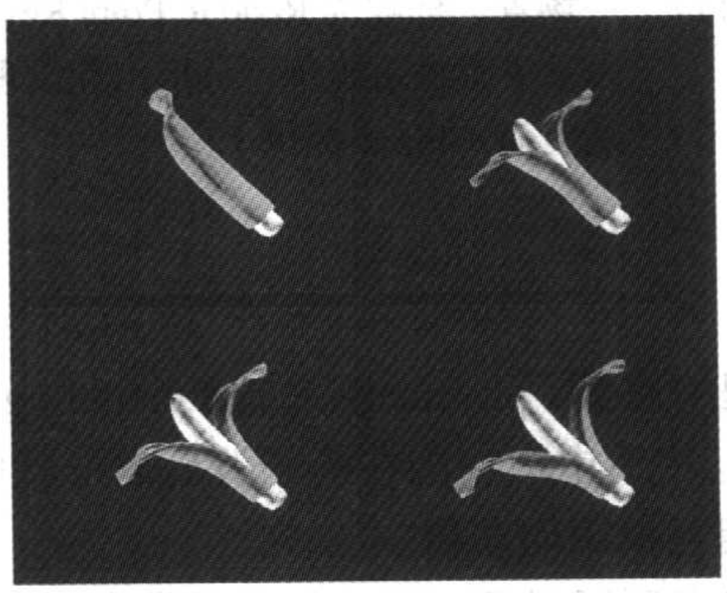


图 8.120 使用弹簧网络模拟香蕉剥皮的柔性效果（加州技术学院计算机图形实验室的 David Laidlaw、John Snyder、Adam Woodbury 和 Alan Barr 提供）

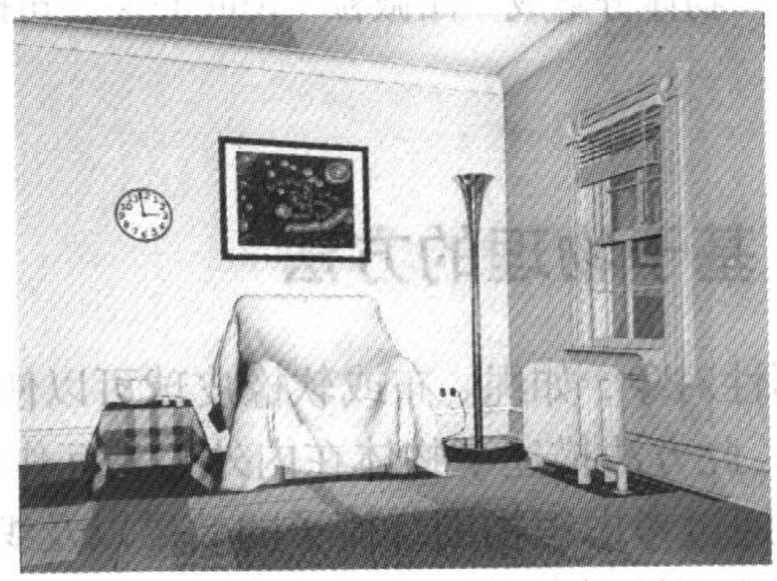
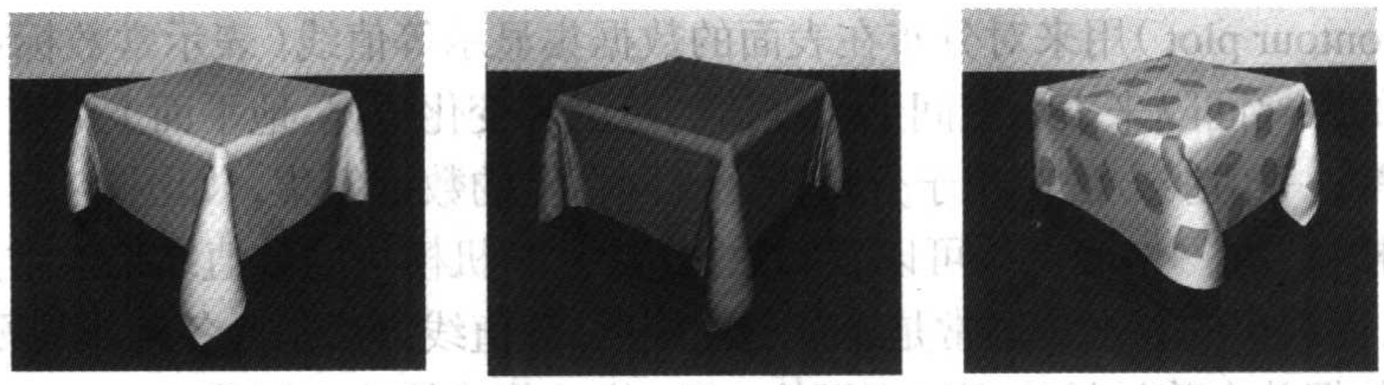


图 8.121 利用能量函数最小化来模拟披在家具上的布的柔性效果（Rensselaer 工业学院设计研究中心的 Gene Greger 和 David E.Breen 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com



(a)　(b)　(c)

图 8.122　使用能量函数最小化来模拟(a)棉、(b)毛、(c)塑料的特征(Rensselaer 工业学院设计研究中心的 David E.Breen 和 Donald H.House 提供)

基于物理的模拟方法也可以用于在动画中更精确地描述运动路径。以前的动画常限于利用样条路径和运动学，其中运动参数也仅基于位置和速度。基于物理的模拟方法使用力学方程描述运动，包括力和加速度。基于力学方程的动画描述比基于运动学方程的描述产生的运动更真实。

8.27　数据集可视化

将计算机图形方法作为工具应用于科学和工程分析通常称为**科学计算可视化**(scientific visualization)。这包括数据集的可视化以及不使用图形方法是难以或不可能分析的应用。例如，可视化技术用来处理大容量数据输出，如超级计算机、卫星和宇宙飞船扫描器、无线电射电望远镜、医用扫描器。计算机模拟的数字化结果及观察设备通常生成数以百万计的数据点，因此简单地扫描原始数据很难确定其趋向和关系。类似地，可视化技术也可以用于长时间的过程分析，或不能直接观察的现象，如对象接近光速运行时的量子力学现象和狭义相对效果。科学计算可视化利用计算机图形学、图像处理、计算机视觉和其他领域的方法进行可视化显示、增强效果和操纵信息，使得更好地理解数据。用于商业、工业和其他非科学领域中的类似方法有时则称为**商用可视化**（business visualization）。

数据集按其空间分布及其数据的类型进行分类。二维数据集有分布在曲面上的值，三维数据集有分布在立方体、球体或其他空间区域内部的值。数据类型包括标量、向量、张量和多变量数据。

8.27.1　标量场的可视化表示

一个标量只有一个值。标量数据集包括了以时间分布同时也以空间位置分布的值。数据值也许是其他标量参数的函数。能量、密度、质量、温度、压力、电荷、电阻、反射率、频率和含水量等都是标量的一些实际例子。

可视化一个标量数据集的一般方法是，把数据值分布看成诸如以位置和时间为参数的函数而形成的图形或图表。如果数据分布在曲面上，可以在表面上绘制一竖棒来标识数据值，或插入数据值以显示一平滑曲面。**伪彩色方法**（pseudo-color method）也用在标量数据集中以区分不同的值。颜色编码技术可以与图形和图表方式合并使用。对于颜色编码标量集，选择颜色值域并把数据值域映射到颜色值域。例如，蓝色对应到最小的标量值，红色对应到最高值。图 8.123 给出了颜色编码表面绘图的例子。对数据集进行颜色编码可能是复杂的，因为某些颜色组合会导致数据的曲解。

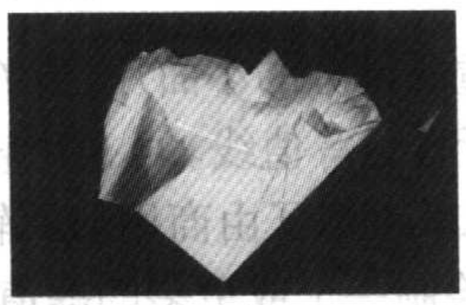


图 8.123　表示1987年10月股市崩溃期间股票变化态势的金融表面绘图。红色表示高回涨，图中显示了崩溃期间低增长股票表现良好（新加坡信息技术学院的Eng-Kiat Koh提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

等值线图（contour plot）用来对分布在表面的数据集显示等值线（表示常数标量值的直线）。等值线以某一合适的距离间隔来表示空间上数据值的范围和变化情况。地平面上高度的等值线图是一个典型应用。通常，等值线方法应用于分布在规则网格上的数据值集，如图 8.124 所示。规则网格有相等间距的网格线，在网格交点处可以获知数据值。计算机模拟的数值结果通常用来生成规则网格上的数据分布，而观察的数据集通常是不规则间距的。等值线方法是为各种非规则网格而设计的。但通常非规则的数据分布将被转换到规则网格上。二维等值线算法在网格内一个单元一个单元地描绘等值线，需要检查网格单元的四个角以确定特殊等值线穿过哪一个单元边。等值线通常在每个单元中以直线段进行绘制，如图 8.125 所示。某些等值线使用样条曲线进行绘制，但样条拟合会导致数据集的不一致性和曲解。例如，两根样条等值线可能重叠，或者等值线路径也许不是数据趋向的真正指示者，因为数据值只是在单元角上是已知的。等值线软件包允许研究者交互调整等值线以修正其不一致性。图 8.126 表示了 xy 平面上三种交叠的彩色编码等值线图，图 8.127 表示了非规则形状空间的等值线和颜色编码。

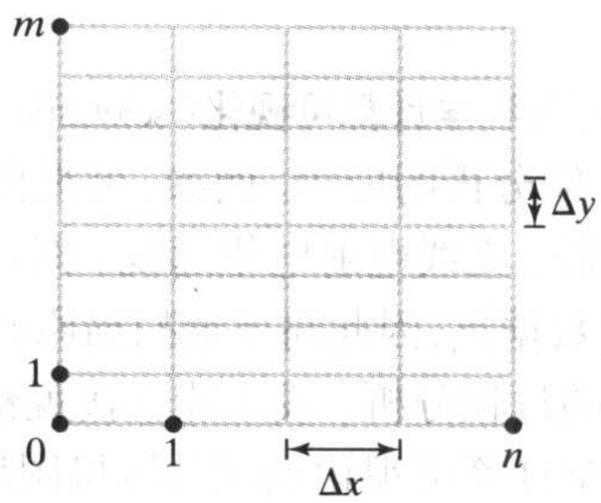


图 8.124　规则二维网格在网格线交点处有数据值。x 方向的网格线有常量 Δx 间距，y 方向的网格线有常量 Δy 间距，这里 x 和 y 方向的间距也许不一样

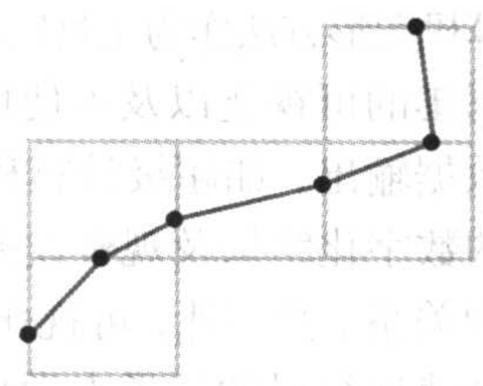


图 8.125　等值线路径通过了 5 个网格单元

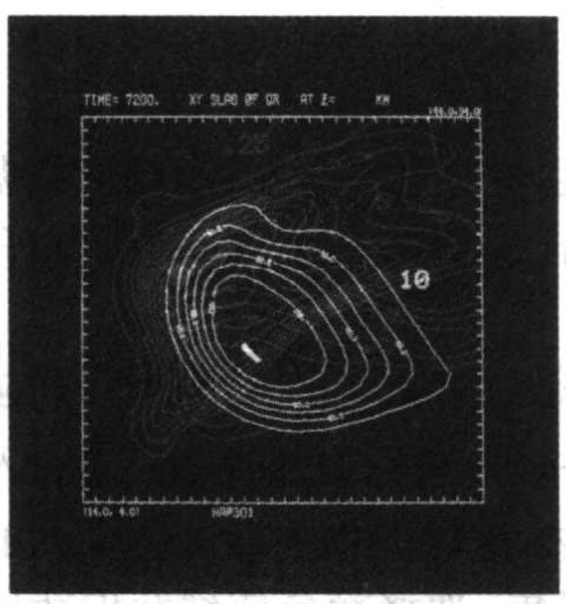


图 8.126　在同一 xy 平面区域内的三个数据集的颜色编码等值线图（伊利诺伊大学提供）

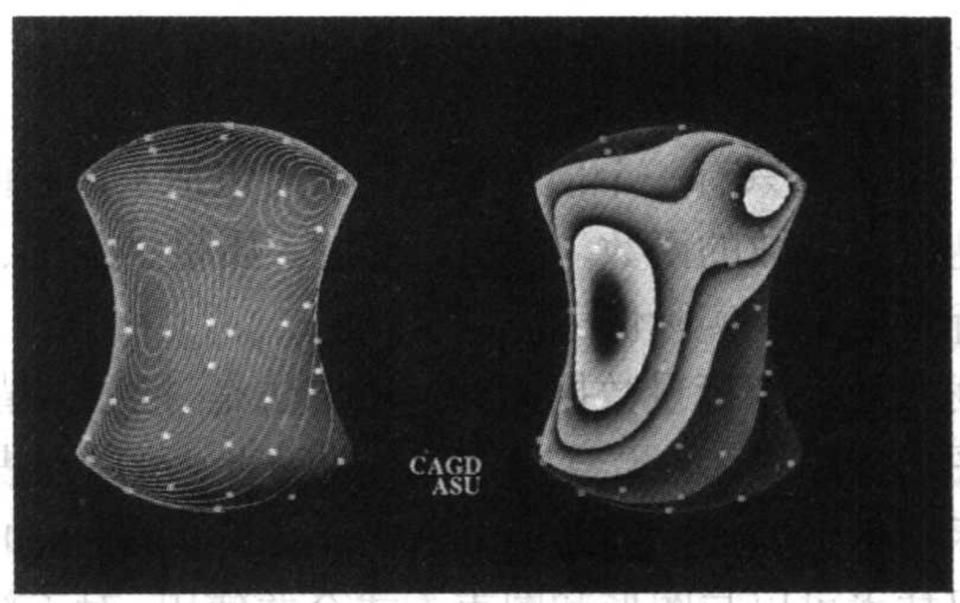


图 8.127　在苹果核形状的空间区域表面上的颜色编码等值线图（亚利桑娜州立大学计算机科学和工程系的 Greg Nielson 提供）

对于三维标量数据场，可以取一重叠部分的小面片，并在此面片上显示二维数据分布。或者对小面片上的数据值进行颜色编码，或者显示等值线。可视化软件包一般提供一个面片子程序，从而允许以任一角度取得剖面部分。图 8.128 表示了由商用切片软件包生成的图示。

除了观察二维剖面部分，还可以绘制一个或更多的**等值面**（isosurface），即三维等值面（参见图 8.129）。当显示两个重叠等值面时，使外部的面半透明以观察两个等值面形状。构造等值面类

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

似于绘制等值线，只是现在使用三维网格单元，需要检查一个单元的八个角以确定等值面部分。图 8.130 给出了与网格单元的等值面相交的部分。等值面使用三角形网格进行模拟，然后利用表面绘制算法以显示最后的形状。

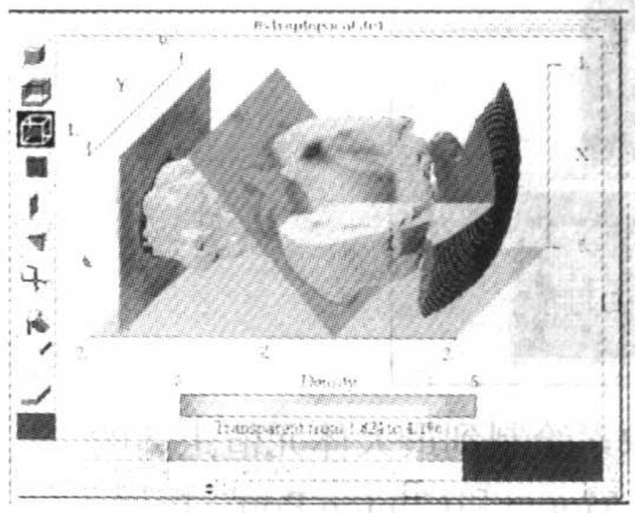


图 8.128　三维数据集的剖面部分（Spyglass 公司提供）

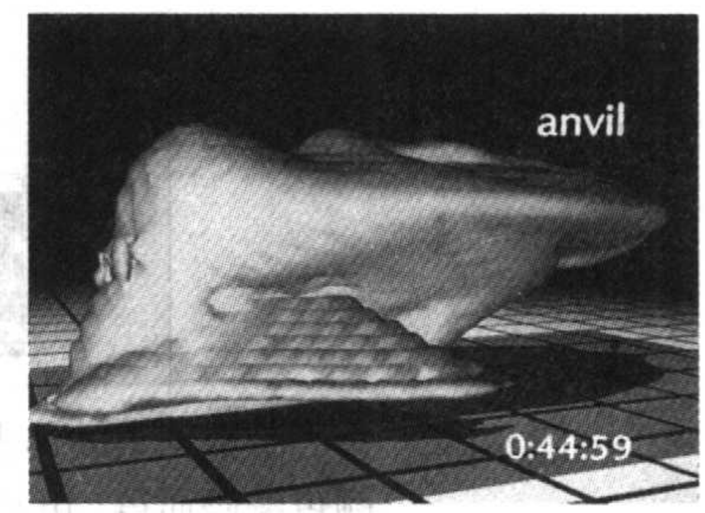


图 8.129　由台风的数值模拟而得到的水量值生成的等值面（伊利诺伊大学大气科学系和 UIUC 巨型机应用国家中心的 Bob Wilhelmson 提供）

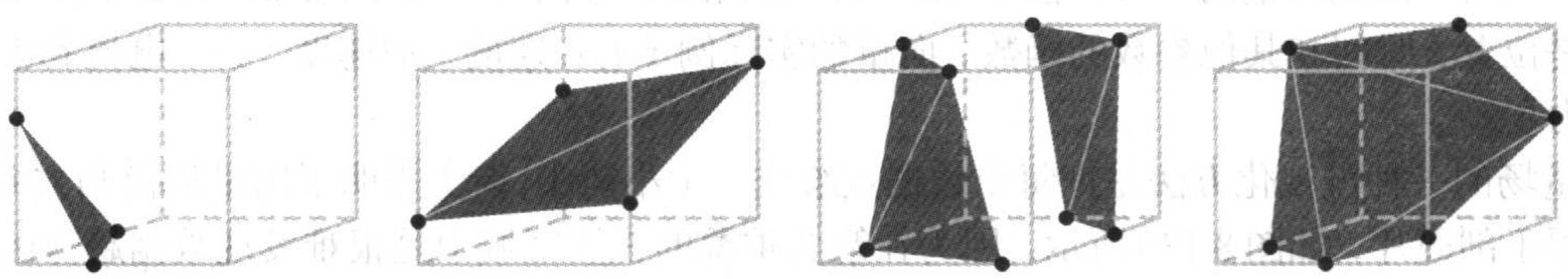


图 8.130　网格单元中的等值面重叠，以三角面片模拟

体绘制（volume rendering）有点类似 X 光片，是另一种可视化三维数据集的方法。有关数据集的内部信息利用8.20节引入的光线投射方法而投影到显示屏上。沿着来自每个屏幕像素的射线路径（参见图 8.131），检测并编码内部数据值。通常网格位置上的数据值取平均，因此每个数据空间体元只要存储一个值。数据显示时如何编码则依赖于应用。例如，检测地震数据通常寻找沿每一射线的最大与最小值。然后，对这些值进行颜色编码以给出有关区间的宽度和最小值。在医学应用中，数据值是组织和骨骼层的从 0 到 1 的阻光度因子。骨骼层完全阻光，而组织是有点透明的（低阻光度）。沿着每一条射线，累计阻光度因子直到其总体值大于 1 或等于 1 为止，或直到射线在三维数据网格后消失。累计的阻光度值作为像素密度层显示成黑白的或彩色的。图 8.132 是描述狗心脏结构的医用数据集的体视图。对于该体视图，显示沿每一像素射线到最大体元值距离的颜色编码绘图。

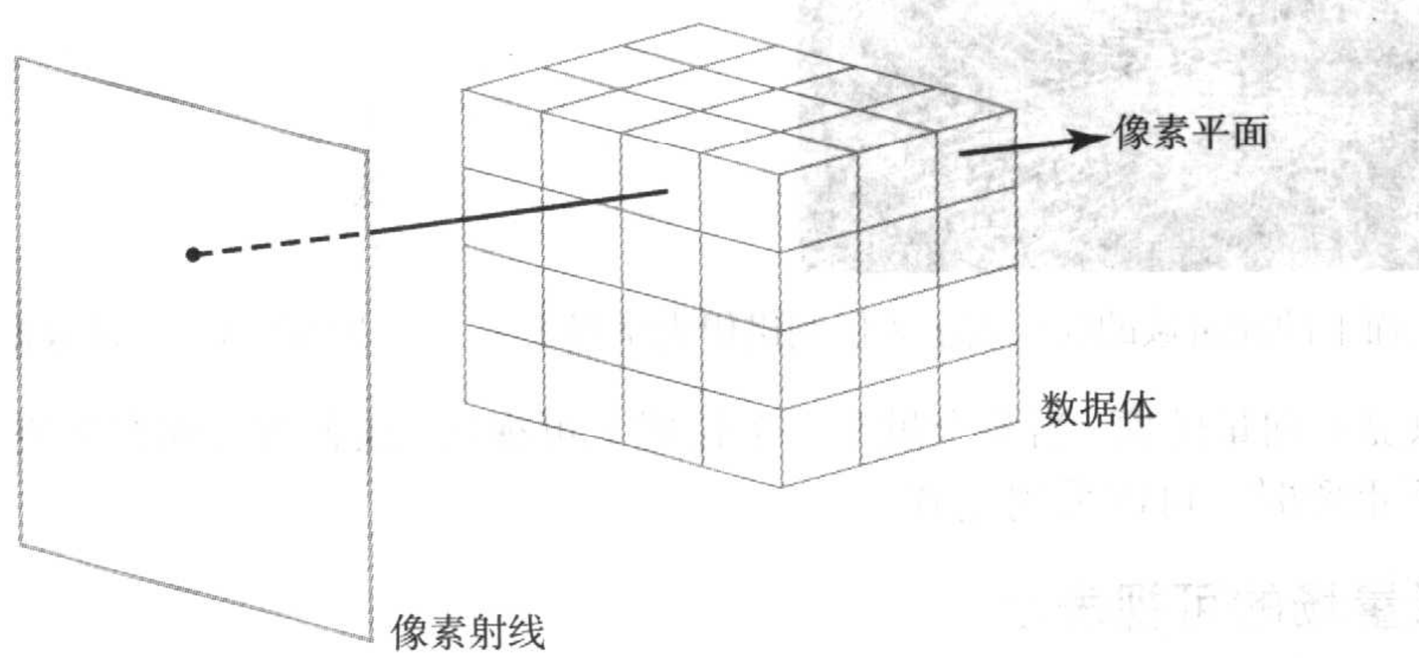


图 8.131　一个规则的笛卡儿数据网格的体可视化，使用了光线投射方法来测量内部数据值

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

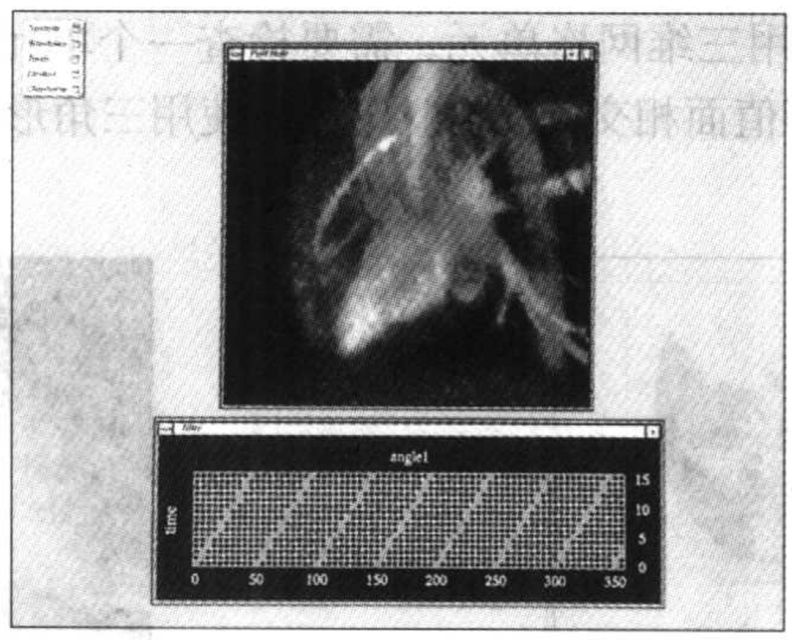


图 8.132　狗心脏数据集的体视图，通过对每个像素绘制到最大体元值距离的颜色编码而得（伊利诺伊大学的Patrick Moran和Clinton Potter提供）

8.27.2　向量场的可视表示

三维空间中的向量 **V** 有三个标量值(V_x, V_y, V_z)，每一标量对应于一个坐标方向。二维向量有两个分量(V_x, V_y)。描述向量的一种方法是给定其模 |**V**| 和其方向上的单位向量 **u**。类似于标量，向量也可以是位置、时间和其他参数的函数。向量的实际例子有：速度、加速度、力、电流和电场、磁场和重力场等。

向量场的一种可视化方法是把每个数据点绘成一个小箭头，箭头表明了向量的模和方向。该方法最常用于剖面片，如图 8.133 所示，因为在箭头重叠的三维区域中是很难发现数据趋势的。变化箭头长度可以表示向量模值，也可以让所有箭头具有相同的大小，但一般根据向量模值选择的颜色编码来使箭头具有不同的颜色。

我们也可以通过绘制场线（field line）或流线（stream-line）来表示向量值。场线通常用于电、磁和重力场。向量模值由两场线的间隔指定，方向与场线相切，如图 8.134 所示。向量场的流线绘制如图 8.135 所示，流线以宽箭头表示，特别是旋涡、旋风效果可以这样表示。这类例子如图 8.136 所示，该图表示了雷暴雨内部旋涡的气流模型。有关液体流动的动画，沿流动方向跟踪微粒可以使向量场可视化。运用流线和微粒向量域的可视化例子如图 8.137 所示。

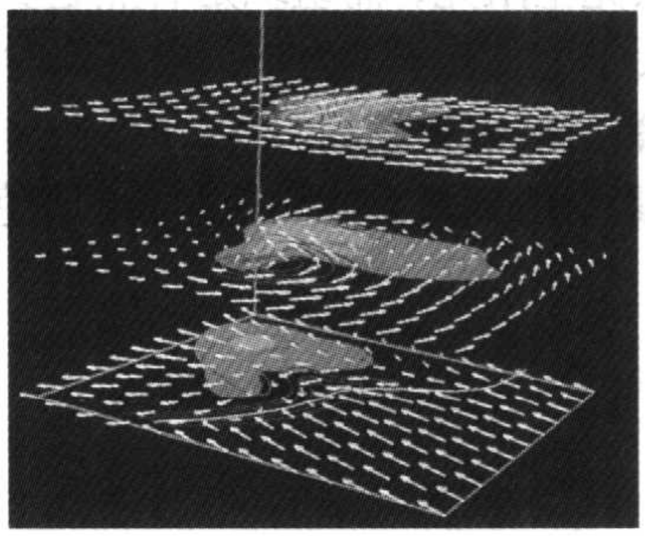


图 8.133　在剖面上的向量域的箭头表示（伊利诺伊大学提供）

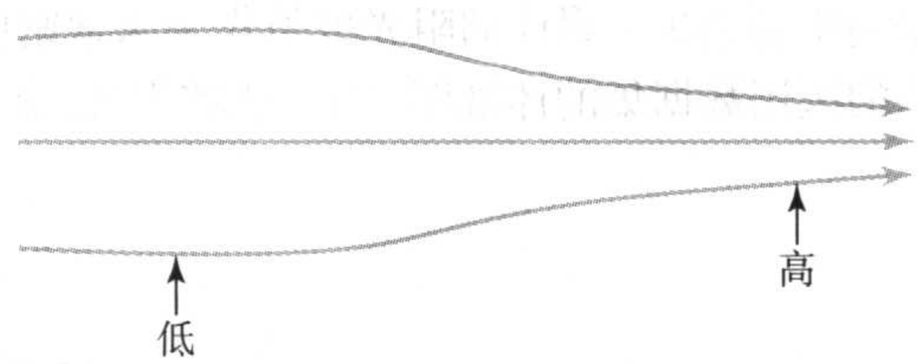


图 8.134　向量数据集的场线表示

有时，只显示向量的模。当多个量在一个位置上可视化，或者当方向在某空间区域内变化不大或向量方向不重要时，可以采用此方法。

8.27.3　张量场的可视表示

三维空间的张量有9个分量，可以使用3 × 3矩阵表示。实际上，这种表示用于二阶张量（second-order tensor），而高阶张量也会出现在某些应用中，特别是在广义相对论中。实际的二阶张量例子

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

有：材料受到外来的压力和拉力，电子传导器的传导率（电阻），给出特殊坐标空间性质的张量。例如，笛卡儿坐标中的应力张量可以表示为

$$\begin{bmatrix} \sigma_x & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_y & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z \end{bmatrix} \tag{8.120}$$

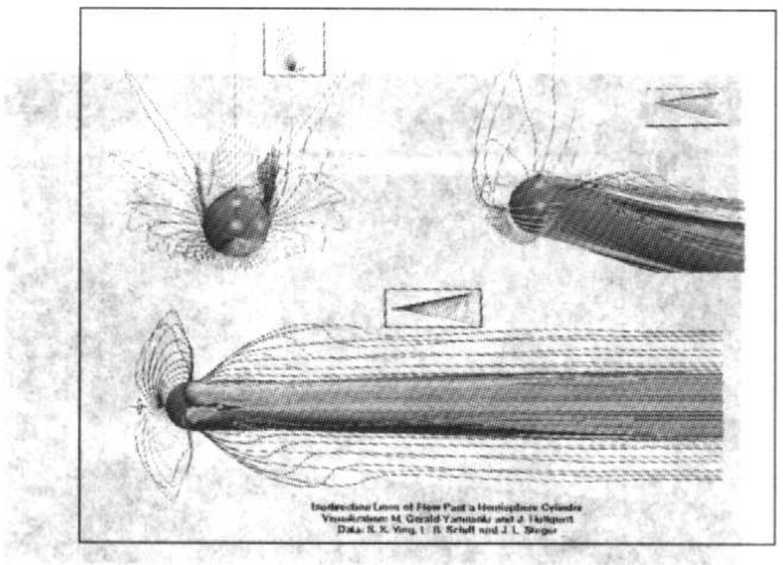


图 8.135　柱体端头是半球帽状的气流图视，柱体与气流进入的方向有一点倾斜（NASA Ames 研究中心的 M.Gerald-Yamasaki、J.Huiltquist 和 Sam Uselton 提供）

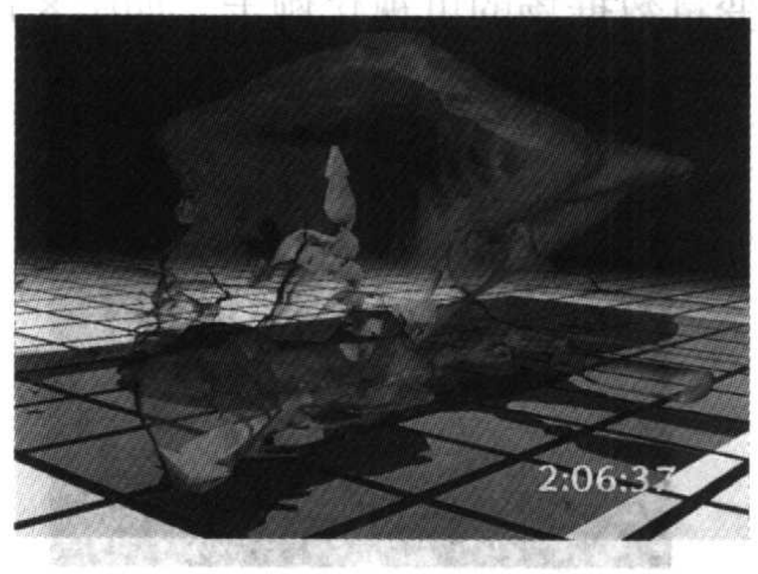


图 8.136　旋转气流模式，在风景的透明等值面绘图中的宽流线图视（伊利诺伊大学的 Bob Wilhelmson 提供）

通常在各向异性的物质中出现张量，即在不同方向有不同的性质。例如，传导率的 x、xy 和 xz 元素描述了 x、y、z 方向到 x 方向电流的电场分量的属性。通常，实际张量是对称的，因此张量只有 6 个不同值。例如，应力张量的 xy 和 yx 分量是相同的。

表示二阶张量 6 个分量的可视化方法基于有 6 个参数的设计形状。图 8.138 给出了张量的图形化表示。张量的三个对角元素用来构造向量的模和方向，非对角元素用来建立椭圆盘的形状和颜色。

我们可以将张量变回到向量或标量从而不必显示6个分量。运用向量表示，可以简单地将张量的对角元素改为向量表示。利用张量缩减（tensor-contraction）算法，可以得到一个标量表示。例如，应力和应变张量可以缩减，从而生成标量的张量能量密度，此密度可以根据受到外力作用的物质点进行绘制（参见图 8.139）。

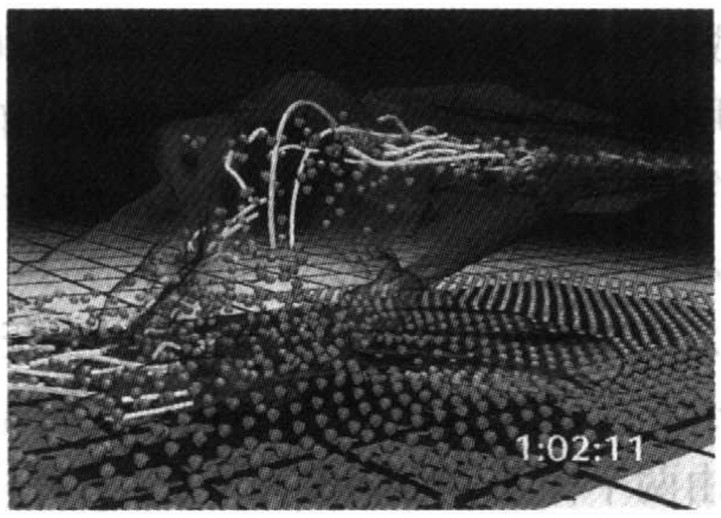


图 8.137　气流模式，由风景透明等值面绘图中的流线和微粒运动来表示。上升的微粒为橘红色，下落的为蓝色（伊利诺伊大学的 Bob Wilhelmson 提供）

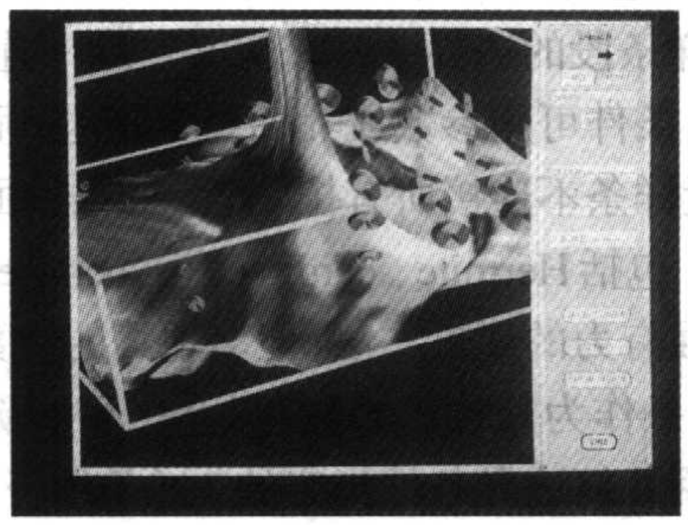


图 8.138　利用椭圆表示应力和应变张量，以及在受力物质表面上的椭圆盘（伊利诺伊大学的 Bob Haber 提供）

8.27.4　多变量数据场的可视表示

某些应用中，在一些空间区域的每个网格点位置上可以有多个数据值，它们可以是标量、向量

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

甚至是张量值的混合表示。以液体流动问题为例，在每个三维位置，可以有液体速度、温度和密度值。这样，在每个位置可以显示 5 个标量值，这种情况类似于显示一张量场。

显示多变量数据场的方法是构造有多个部分的图形对象，有时称其为**点符**（glyph）。点符的每一部分表示一个实际量。每部分的大小和颜色可以用来显示有关标量的信息。对于向量场而言，为了给出方向信息，可以使用楔形、锥形或其他形状来表示向量。在选定的网格位置运用点符结构的多变量数据场的可视化例子，如图 8.140 所示。

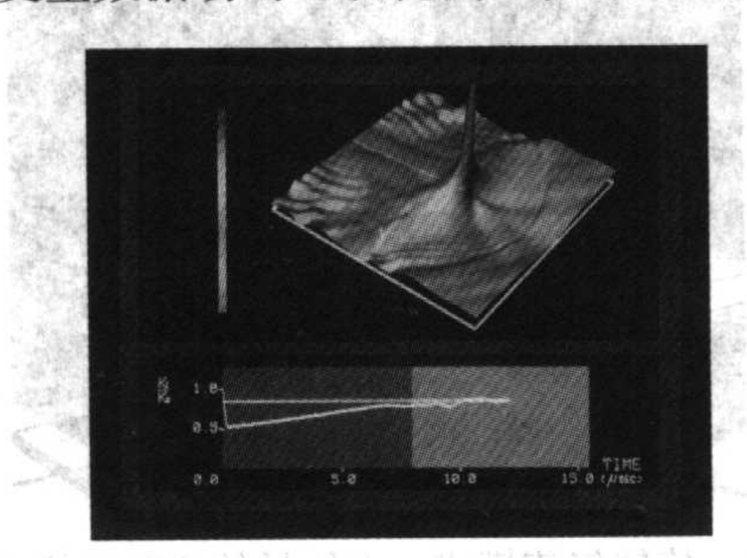


图 8.139　在图视受力物质表面的裂纹传播中，使用应变能密度表示的应力和应变张量绘图（伊利诺伊大学的 Bob Haber 提供）

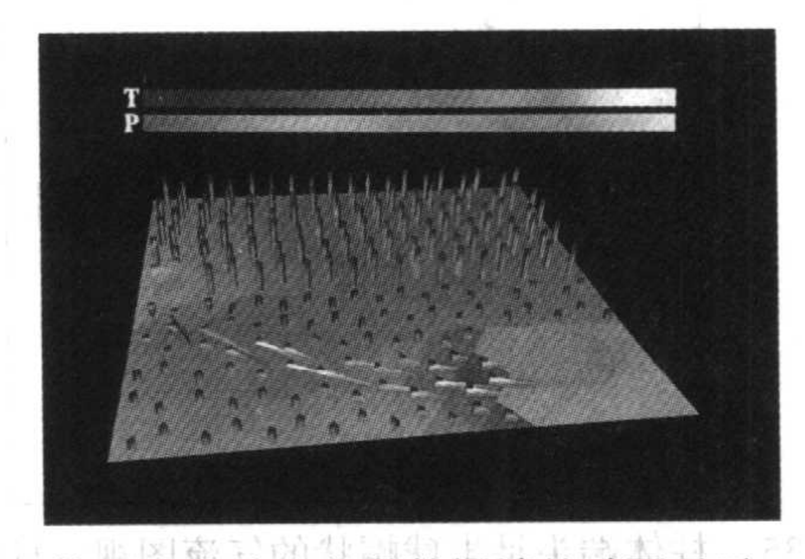


图 8.140　使用点符的多变量数据域的动画显示一结构。点符的锲形部分指示了向量在每一点的方向（伊利诺伊大学 UIUC 巨型机应用国家中心提供）

8.28　小结

已经开发出多种表示法用于模拟在计算机图形场景中显示的各种各样的对象。多数情况下，三维对象由一个软件包将其按标准图形对象（standard graphics object）来绘制，其表面用多边形网格显示。

图形软件包均提供显示某些普通二次曲面，如球面和椭球面的函数。二次曲面的扩充，称为超二次曲面，提供了另外的参数用来创建众多的对象形状。对于非刚体、柔性曲面，可使用半流质对象按高斯碰撞建立形态。

分段连续多项式曲线样条表示是最广泛地应用于CAD中的方法。样条曲线或曲面由一组控制点和关于样条段的边界条件来定义。顺序连接控制点的线形成控制图，而所有控制点位于样条的凸包内。边界条件可用参数或几何导数指定，而多数样条表示使用参数边界条件。插值样条连接所有控制点，逼近样条不连接所有控制点。样条曲面可用两个多项式的笛卡儿积描述。三次多项式一般用于插值表示，包括 Hermite、cardinal 和 Kochanek-Bartels 样条。Bézier 样条为描述曲线和曲面提供了一种简单的且强有力的逼近方法，但多项式的次数由控制点数量确定且对曲线形状的局部控制很难获得。以 Bézier样条作为其特例的B样条是一种有更多用途的逼近表示方法，但是它们要求指定节点向量。beta 样条是用几何边界条件指定的B样条的一般化。而有理样条由两个样条表示相除而形成。有理样条可用来描述二次曲面，且它们在透视观察变换中是不变的。具有非均匀节点向量的有理B样条通常称为一个 NURB。可使用向前差分计算或细分方法获得沿样条曲线或曲面确定坐标位置。

其他设计技术有扫描表示、结构实体几何法、八叉树和 BSP 树。扫描表示法由一个二维形状在一个区域或空间中平移或旋转来形成。结构实体几何法将两个或多个三维形体通过集合操作：并、差和交来组合。八叉树和 BSP 树使用空间细分方法。

分形几何表示为描述自然景象提供了高效率的方法。可使用这些方法建立地面、树、灌木、水和云的模型，以及生成奇异的图案。分形对象使用一个构造过程和一个分形维数来描述。分形构造

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

过程包括几何构造、中点-移位方法、复数空间自平方操作及逆变换。使用变换规则的其他构造对象表示的过程方法有形状语法和嫁接。

表现为流质性的对象如云、烟、火、水和爆炸或内聚，可以用粒子系统建模。使用这一表示框架，可用一组粒子和主导粒子运动的规则描述对象。

基于物理的建模方法可用来描述柔性对象的特性，如绳子、橡皮或布料。这种方法利用类似弹簧的剖面网格来表示一种材料，使用加于对象的力来计算变形。

可视化技术使用计算机图形方法分析数据集，包括标量、向量和各种混合的张量值。数据表示可用颜色编码或各种对象形状的显示来实现。

在 OpenGL 中使用多边形、三角形或四边形图元函数可指定标准图形对象的多边形面片。GLUT 子程序可用来显示五种规则多面体。球面、锥面和其他二次曲面可用 GLUT 和 GLU 函数显示，而且有一个 GLUT 函数用来生成三次曲面 Utah 茶壶。OpenGL 核心库包含生成 Bézier 样条的函数。而 GLU 函数库中有指定 B 样条和样条曲面修剪曲线的函数。表 8.1 到表 8.4 总结了本章讨论的 OpenGL 多面体、二次曲面、三次曲面和样条函数。

表 8.1　OpenGL 多面体函数小结

函数	描述
glutWireTetrahedron	显示一个线框图的三棱锥（四面体）
glutSolidTetrahedron	显示一个表面着色的四面体
glutWireCube	显示一个线框图的立方体
glutSolidCube	显示一个表面着色的立方体
glutWireOctahedron	显示一个线框图的八面体
glutSolidOctahedron	显示一个表面着色的八面体
glutWireDodecahedron	显示一个线框图的十二面体
glutSolidDodecahedron	显示一个表面着色的十二面体
glutWireIcosahedron	显示一个线框图的二十面体
glutSolidIcosahedron	显示一个表面着色的二十面体

表 8.2　OpenGL 二次曲面和三次曲面函数小结

函数	描述
glutWireSphere	显示一个线框图的 GLUT 球面
glutSolidSphere	显示一个表面着色的 GLUT 球面
glutWireCone	显示一个线框图的 GLUT 锥面
glutSolidCone	显示一个表面着色的 GLUT 锥面
glutWireTorus	显示一个线框图的 GLUT 剖面为圆的环形曲面
glutSolidTorus	显示一个表面着色的 GLUT 剖面为圆的环形曲面
glutWireTeapot	显示一个线框图的 GLUT 茶壶
glutSolidTeapot	显示一个表面着色的 GLUT 茶壶
gluNewQuadric	一个用声明：GLUquadricObj *nameOfObject，定义的对象名激活该 GLU 二次曲面绘制器
gluQuadricDrawStyle	为一个预定义的 GLU 对象名选择显示模式
gluSphere	显示一个 GLU 球面
gluCylinder	显示一个 GLU 锥面、柱面或锥形柱面
gluDisk	显示一个 GLU 平圆环或实心圆盘
gluPartialDisk	显示一个 GLU 平圆环或实心圆盘的一部分
gluDeleteQuadric	删除一个二次曲面对象
gluQuadricOrientation	定义一个 GLU 二次曲面的内外方向
gluQuadricNormals	指定如何为一个 GLU 二次曲面对象生成表面法线
gluQuadricCallback	为一个 GLU 二次曲面对象指定一个出错回调函数

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

表 8.3 OpenGL 的 Bézier 函数小结

函数	描述
glMap1	为 Bézier 曲线显示、颜色值等指定参数，并用 glEnable 激活这些函数
glEvalCoord1	为 Bézier 曲线计算坐标位置
glMapGrid1	指定两个 Bézier 曲线参数之间均匀分段的数目
glEvalMesh1	为 Bézier 曲线显示指定显示模式和整型范围
glMap2	为 Bézier 曲面显示、颜色值等指定参数，并用 glEnable 激活这些函数
glEvalCoord2	为 Bézier 曲面计算坐标位置
glMapGrid2	指定两个 Bézier 曲面参数之间均匀分段的数目
glEvalMesh2	为 Bézier 曲面显示指定显示模式和整型范围

表 8.3 OpenGL 的 B 样条函数小结

函数	描述
gluNewNurbsRenderer	为一个用声明：GLUnurbsObj *bsplineName 定义的对象名激活该 GLU B 样条绘制器
gluBeginCurve	为指定的一段或多段 B 样条曲线赋参数值
gluEndCurve	指示 B 样条曲线参数描述的结尾
gluNurbsCurve	为一个命名的 B 样条曲线段指定参数值
gluDeleteNurbsRenderer	删除一条指定的 B 样条
gluNurbsProperty	为指定的 B 样条指出绘制选择
gluGetNurbsProperty	为一特定 B 样条确定指定特性的当前值
gluBeginSurface	为一指定的有一个或多个部分的 B 样条曲面开始赋参数值
gluEndSurface	指示 B 样条曲面参数描述的结尾
gluNurbsSurface	为一个命名的 B 样条曲面部分指定参数值
gluLoadSamplingMatrices	指定用于为 B 样条曲面取样和剔除子程序的观察和几何参数
gluNurbsCallback	为一个设定的 B 样条和辅助事件指定一个回调函数
gluNurbsCallbackData	指定要传递给事件回调函数的数据值
gluBeginTrim	为一个 B 样条曲面开始赋修剪曲线参数值
gluEndTrim	指示修剪曲线描述的结尾
gluPwlCurve	为一个 B 样条曲面指定修剪曲线参数值

参考文献

超二次曲面的详细讨论请参见 Barr(1981)。柔性对象模型的更多信息，请参见 Blinn(1982)。元球（meta-ball）模型在 Nishimura(1985)中讨论。软对象在 Wyville, Wyville, and McPheeters(1987)中讨论。

有关参数曲线和曲面表达式的信息来源，包括 Bézier(1972)、Barsky and Beatty(1983)、Barsky (1984)、Kochanek and Bartels(1984)、Huitric and Nahas(1985)、Mortenson(1985)、Farin(1988)和Rogers and Adams(1990)和 Piegl and Tiller(1997)。

八叉树和四叉树讨论请参见 Doctor and Torberg(1981)、Yamaguchi, Kunii, and Fujimura(1984) 和 Brunet and Navazo(1990)。Gordon and Chen(1991)给出了BSP树的方法。而Requicha and Rossignac (1992)讨论了实体建模方法。

分形表示的更详细信息请参见 Mandelbrot(1977,1982)、Fournier, Fussel, and Carpenter(1982)、Norton(1982)、Peitgen and Richter(1986)、Peitgen and Saupe(1988)、Hart, Sandin, and Kauffman(1989)、Koh and Hearn(1992)和 Barnsley(1993)。用于各种自然景物的建模方法在 Fournier and Reeves(1986) 和 Fowler, Meinhardt, and Prusinkiewicz(1992)中给出。形状语法在 Glassner(1992)中讨论，微粒系统在 Reeves(1983)中讨论。基于物理的模拟讨论由 Barzel(1992)给出。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

可视化方法的一般介绍由 Hearn 和 Baker(1991)给出。特定可视化方法的其他信息可以参见：Sabin(1985)、Lorensen and Cline(1987)、Drebin, Carpenter, and Hanrahan(1988)、Sabella(1988)、Upson and Keeler(1988)、Frenkel(1989)、Nielson, Shriver, and Rosenblum(1990)和 Nielson(1993)。信息的可视化显示准则由 Tufte(1990, 1997, and 2001)给出。

各种表示方法的编程技术请参见 Glassner(1990)、Arvo(1991)、Kirk(1992)、Heckbert(1994)和 Paeth(1995)。OpenGL Bézier 样条、B 样条和修剪曲线函数请参见 Woo, Neider, Davis, and Shreiner (1990)。Kilgard(1996)讨论了显示多面体、二次曲面和 Utah 茶壶的 GLUT 函数。OpenGL 核心库和 GLU 库函数的完整集在 Shreiner(2000)中给出。

练习题

8.1　建立一个将给定球面转换为多边形网格表示的算法。

8.2　建立一个将给定椭球面转换为多边形网格表示的算法。

8.3　建立一个将给定柱面转换为多边形网格表示的算法。

8.4　建立一个将给定超椭球面转换为多边形网格表示的算法。

8.5　建立一个将给定元球对象转换为多边形网格表示的算法。

8.6　给定在 xy 平面上的一组控制点输入，编写一个显示 cardinal 样条曲线的子程序。

8.7　给定在 xy 平面上的一组控制点输入，编写一个显示二维 Kochanek-Bartels 曲线的子程序。

8.8　在 xy 平面上指定的三个控制点的 Bézier 曲线混合函数是什么？绘出每个函数并标出最小和最大函数值。

8.9　在 xy 平面上指定的五个控制点的 Bézier 曲线混合函数是什么？绘出每个函数并标出最小和最大函数值。

8.10　修改 8.10 节中的示例程序，使其显示任一使用 xy 平面上的四个输入控制点的三次 Bézier 曲线。

8.11　修改 8.10 节中的示例程序，使其显示任一使用 xy 平面上的 n 个输入控制点的 $n-1$ 次 Bézier 曲线。

8.12　完善 8.18 节中的 OpenGL 示例程序，使其显示任一使用 xy 平面上的四个输入控制点的三次 Bézier 曲线。

8.13　完善 8.18 节中的 OpenGL 示例程序，使其显示任一使用 xyz 空间上的四个输入控制点的三次 Bézier 曲线。使用正交投影显示该曲线，观察参数作为输入来指定。

8.14　写出设计二维的、具有一阶分段连续性的二维 Bézier 曲线形状的子程序。曲线每一部分的控制点数量和位置作为输入来指定。

8.15　写出设计二维的、具有二阶分段连续性的二维 Bézier 曲线形状的子程序。曲线每一部分的控制点数量和位置作为输入来指定。

8.16　修改 8.10 节中的示例程序，使其用细分方法计算曲线点来显示任一使用 xy 平面上的四个输入控制点的三次 Bézier 曲线。

8.17　修改 8.10 节中的示例程序，使其用向前差分方法计算曲线点来显示任一使用 xy 平面上的四个输入控制点的三次 Bézier 曲线。

8.18　$d = 5$ 时二维均匀周期性 B 样条曲线的混合函数是什么？

8.19　$d = 6$ 时二维均匀周期性 B 样条曲线的混合函数是什么？

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

8.20 修改8.10节中的示例程序，使其用向前差分方法计算曲线点来显示任一使用一组输入的控制点的二维均匀周期性B样条曲线。

8.21 修改上题的程序，使其用OpenGL函数显示B样条曲线。

8.22 使用有理Bézier样条表示，写出显示xy平面上任一二次曲线的子程序。

8.23 使用有理B样条表示，写出显示xy平面上任一二次曲线的子程序。

8.24 给出一种计算Bézier曲面上一给定点$\mathbf{P}(u, v)$处法向量的算法。

8.25 为一个给定的二次曲面推导计算向前差分的表达式。

8.26 为一个给定的三次曲面推导计算向前差分的表达式。

8.27 建立由输入参数生成三维对象描述的程序，参数以平移扫描定义物体。

8.28 建立由输入参数生成三维对象描述的程序，参数以旋转扫描定义物体。

8.29 利用构造实体几何结构的方法，设计一个生成实体的算法，实体由三维基本形状组合而成。

8.30 修改上题中的算法使图元形状由八叉树结构定义。

8.31 考虑将二维景物编码成四叉树表示的算法。

8.32 建立把四叉树表示装入显示景物的帧缓存中的算法。

8.33 写出将三维对象的多边形定义转变到八叉树表示的子程序。

8.34 使用随机中点位移方法，写出起始于xy平面的水平线上生成山的轮廓的子程序。

8.35 为地面给定一组拐角高度，使用随机中点位移方法，写出计算地平面高度的子程序。

8.36 对于任意指定的迭代次数，写出生成分形雪花（Koch曲线）的程序。

8.37 对于任意指定的迭代次数，用图8.73或图8.74中的一个生成元，写出生成分形曲线的程序。该曲线的分形维数是多少？

8.38 利用自平方函数$f(z) = z^2 + \lambda$，其中λ是作为输入来指定的复常量，写出生成分形曲线的子程序。

8.39 利用自平方函数$f(z) = i(z^2 + 1)$，其中$i = \sqrt{-1}$，写出生成分形曲线的子程序。

8.40 修改8.23节的例子程序，使用附加的颜色层次显示沿Mandelbrot集的边界区域。

8.41 修改上题中的程序，使颜色和颜色层次数作为输入值给出。

8.42 修改上题中的程序，沿Mandelbrot集选择和显示矩形边界区域。

8.43 为一个指定的圆和一组点位置写出一个实现点的逆变换等式（8.118）的子程序。

8.44 设计一组几何替换规则来改变等边三角形的形状。

8.45 为上题编写一个显示三角形转换过程的程序。

8.46 利用微粒系统，写出在xy平面上对一个爆炸的球建模和显示的程序。

8.47 修改上题中的程序，使一个爆竹（圆柱形）爆炸。

8.48 开发一个子程序，用同一个弹簧组成的网格对一块小矩形布料建模。

8.49 使用伪颜色方法，写出可视化一个二维标量数据集的子程序。

8.50 使用等值线方法，写出可视化二维标量数据集的子程序。

8.51 使用向量值的箭头表示，写出可视化二维向量数据集的子程序。使每个箭头具有相同长度，但使用不同颜色来显示箭头以表示不同的向量模。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第9章　可见面判别算法

在生成真实感图形时，考虑最多的是如何判别出从某一选定观察位置所能看到的场景中的内容。目前，已经有多种解决该问题的方法，并针对不同的应用开发出了许多有效地判别可见对象的算法。其中，有些算法的内存开销较大，有些算法的处理时间较长，而其他一些算法则仅针对某些特定的对象类型。我们为某个特定的应用背景选择算法时，应考虑场景复杂度、待显示对象的类型、显示设备及最终画面是静态还是动态等因素。尽管判别可见面与隐藏面之间有细微的差别，不过这些算法通常称为**可见面判别**（visible-surface detection）算法，有时也称为**隐藏面消除**（hidden-surface elimination）算法。例如，在显示线框方式的图形时，我们可能仅仅希望使用虚线轮廓或其他方式来显示隐藏面而不必消除它们，从而保留对象的外形特征。

9.1　可见面判别算法的分类

可见面判别算法的分类，通常是根据其处理场景时是直接处理对象定义还是处理它们的投影图像。这两种类型分别称为**物空间**（object-space）算法和**像空间**（image-space）算法。物空间算法将场景中的各对象和对象的各个组成部件相互进行比较，从而最终判别出哪些面是可见的；而像空间算法则在投影平面上逐点判断各像素所对应的可见面。尽管物空间可见面判别算法在许多场合十分高效，但人们大多使用像空间的可见面判别算法。例如，线显示算法大多使用物空间算法来确定线框图中的可见线，但是许多像空间的可见面判别算法也很容易移植过来，用于判别出可见线段。

虽然各种可见面判别算法的基本思想各不相同，但它们大多利用排序和连贯性方法以提高效率。将场景中的对象表面根据它们与观察平面的距离进行排序，可以加速深度比较，而连贯性方法则充分利用场景的规则性特征。一条扫描线可能包含相同强度的像素区段，并且相邻扫描线之间的图案变化很小。动画中的各帧之间仅在运动对象的相邻区域内有差异，并且通常可以建立起场景中的对象与场景表面之间的稳定联系。

9.2　后向面判别

快速简便地判别多面体**后向面**（back face）的物空间算法建立在3.15节讨论的前-后测试的基础上。设点(x, y, z)满足：

$$Ax + By + Cz + D < 0 \tag{9.1}$$

则该点在多边形面的后面，这里A、B、C、D是多边形面的平面参数。如果该点位于视点到该多边形面的直线上，则我们正在看该多边形的后向面。因此，我们可以使用观察点来测试后向面。

可以通过考查某多边形面的法向量$\mathbf{N}$的方向来简化后向面测试。如果$\mathbf{V}_{view}$为如图9.1所示的由相机位置出发的观察向量，则当

$$\mathbf{V}_{view} \cdot \mathbf{N} > 0 \tag{9.2}$$

时，该多边形为后向面。另外，若将对象描述转换至投影坐标系后，观察方向平行于观察坐标系中的z_v轴，则我们仅需考虑法向量$\mathbf{N}$的z分量。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在一个沿着 z_v 轴反向观察的右手观察系统中（参见图9.2），若法向量 **N** 的 z 分量 C 满足 $C<0$，则该多边形为一后向面。同时，我们无法观察到法向量的 z 分量 $C=0$ 的所有多边形面，因为观察方向与该面相切。这样，一旦某多边形面的法向量的 z 分量值：

$$C \leqslant 0 \tag{9.3}$$

即可判定其为一后向面。

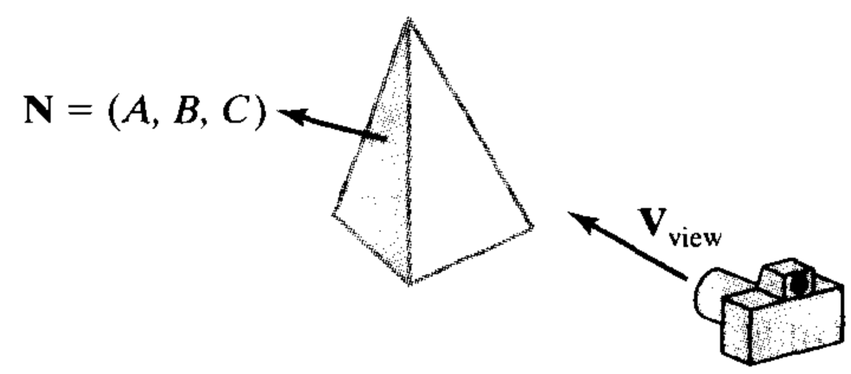


图9.1 一个面的法向量 **N** 与观察方向向量 $\mathbf{V}_{view}$

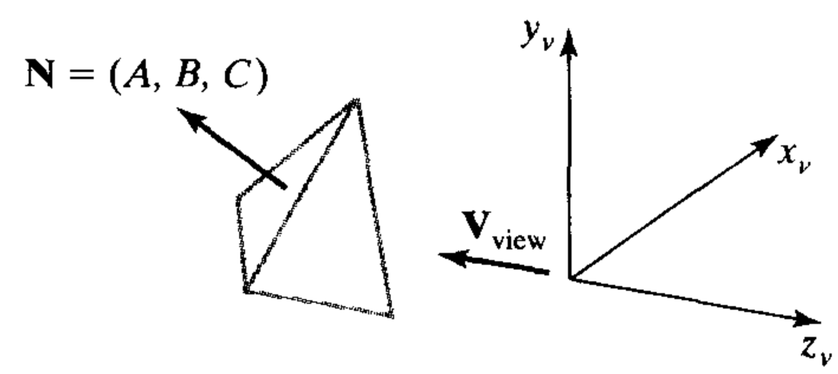


图9.2 在一个沿 z_v 负轴方向观察的右手观察系统中，平面参数 $C<0$ 的多边形面必为后向面

在采用左手观察系统的软件包中，以上方法也同样适用。平面参数 A、B、C、D 可由顺时针方向标识的多边形顶点坐标计算出来（右手系统中为逆时针方向）。不等式（9.1）仍适用于测试多边形后面的点。同样，如果观察方向与 z_v 轴正向一致，则后向面的法向量为远离视点的方向，可由 $C \geqslant 0$ 加以判别。

对于一个对象的所有表面，只须检查其所在平面的平面参数 C，即可迅速判别出所有的后向面。对于单个凸多面体，例如图9.2中的锥体，该测试方法可以判别出对象的所有隐藏面，因为其每个面或者是完全可见，或者是完全不可见。另外，如果场景中只包含一些互相不覆盖的凸多面体，则也可以使用后向面判别算法找出所有的隐藏面。

对于其他对象，例如图9.3中的凹多面体，则还需进行更多的测试，以检查是否仍存在被其他面完全或部分遮挡的对象表面。另外，要检查场景中是否存在沿视线方向覆盖的对象。然后，需要进一步确定是否全部或部分地覆盖了被遮挡对象。通常，后向面消隐处理可消除场景中一半左右的隐藏面。

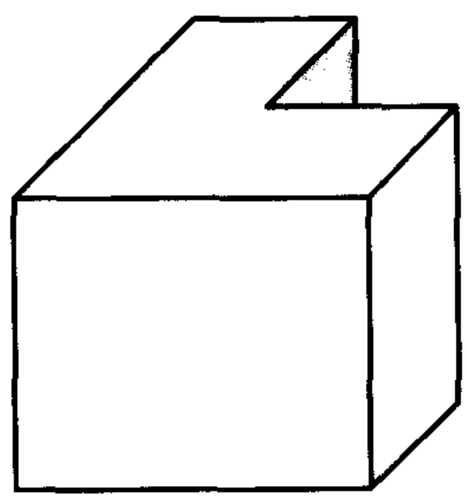


图9.3 一个凹多面体的视图，其中的一个面被其他面遮挡

9.3 深度缓存算法

深度缓存算法（depth-buffer method）是一个比较常用的判定对象表面可见性的像空间算法，它在投影面上的每一像素位置比较场景中所有面的深度。该算法对场景中的各个对象表面单独进行处理，且在表面上逐点进行。该算法通常应用于只包含多边形面的场景，因为这些场景适合于很快地计算出深度值且算法易于实现。当然，该算法也可以应用于非平面的对象表面。由于通常沿着观察系统的 z 轴来计算各对象距观察平面的深度，因此上述算法也称为 z 缓存（z-buffer）算法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图9.4显示了从某观察平面上的点(x, y)出发沿正交投影方向的不同距离的三个表面。这些面可按任意次序处理。在处理每一面时，将其到观察平面的深度与前面已处理表面进行比较。如果一个表面比任一已处理表面都近，则计算其表面颜色并和其深度一起存储。场景的可见面由一组在所有表面处理完后存储的表面来表示。深度缓存算法通常在规范化坐标系统中实现，因此深度值的范围从近裁剪平面的 0 到远裁剪平面的 1.0。

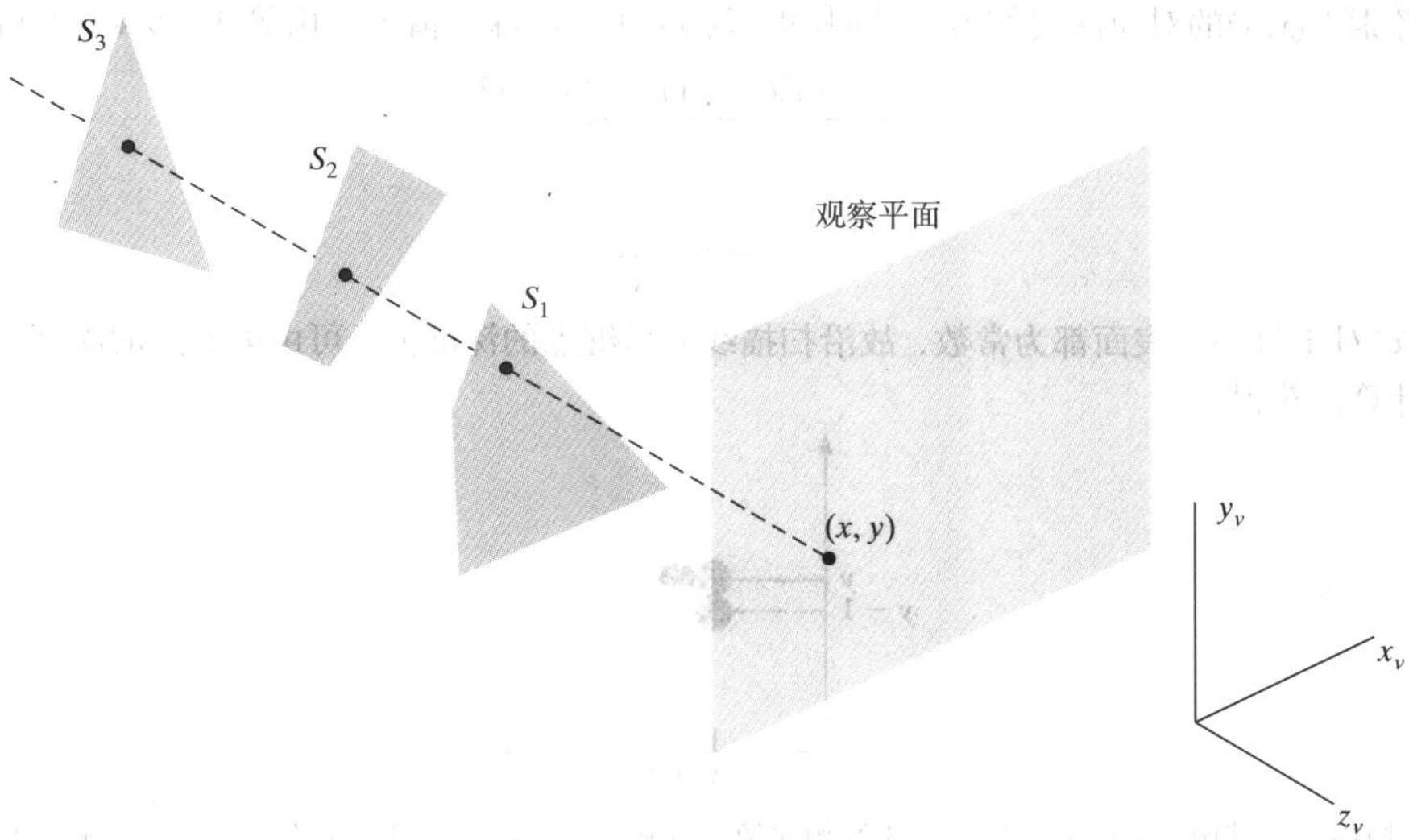


图 9.4　三个表面在观察平面(x, y)位置重叠，可见面 S_1 的深度最小

正如算法名称所示，系统共需两块缓存区域。深度缓存用于保存表面上各像素点(x, y)所对应的深度值，而刷新缓存则保存每一像素位置的表面颜色值。算法执行时，深度缓存中的所有单元均初始化为 1.0（最大深度），而帧缓存中各单元则初始化为背景颜色。然后，逐个处理多边形表中的各个表面，每次扫描一行，计算各像素点(x, y)所对应的深度值。并将计算出的深度与深度缓存中该像素单元所存储的数值进行比较。若计算深度小于存储值，则存储新的深度值。并将该点的表面颜色存入帧缓存的对应单元。

设深度值规范化到0.0 ~ 1.0的范围且观察平面在深度为0处，则下面的算法概述了深度缓存算法的步骤。该算法也可应用于任何其他的深度范围，而有些图形软件允许用户为深度缓存算法指定深度范围。

深度缓存算法

1. 将深度缓存与帧缓存中的所有单元(x, y)初始化，使得

```
depthBuff(x,y) = 1.0,        frameBuff(x,y) = backgndColor
```

2. 处理场景中的每一多边形，每次一个。

- 计算多边形面上各点(x, y)处的深度值 z（如果不是已知）。
- 若 $z <$ depthBuff(x, y)，则计算该位置的表面颜色并设定

```
depthBuff(x,y) = z,          frameBuff(x,y) = surfColor(x,y)
```

当处理完所有多边形面后，深度缓存中保存的是可见面的深度值，而帧缓存保存了这些表面的对应属性值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

给定场景中任一多边形面顶点位置的深度值，可以计算包含该多边形的平面上所有点的深度。在表面点(x, y)处，由平面方程计算深度如下：

$$z = \frac{-Ax - By - D}{C} \tag{9.4}$$

对任一扫描线（参见图9.5），线上相邻点间的x水平位移为±1，相邻扫描线间的y垂直位移也为±1。若已知像素点(x, y)的对应深度值为z，则其相邻点$(x+1, y)$的深度值z'可由等式（9.4）得到：

$$z' = \frac{-A(x+1) - By - D}{C} \tag{9.5}$$

或

$$z' = z - \frac{A}{C} \tag{9.6}$$

比率$-A/C$对于每一个表面都为常数，故沿扫描线的后继点的深度值，可由前面点的深度值使用一次加法计算而获得。

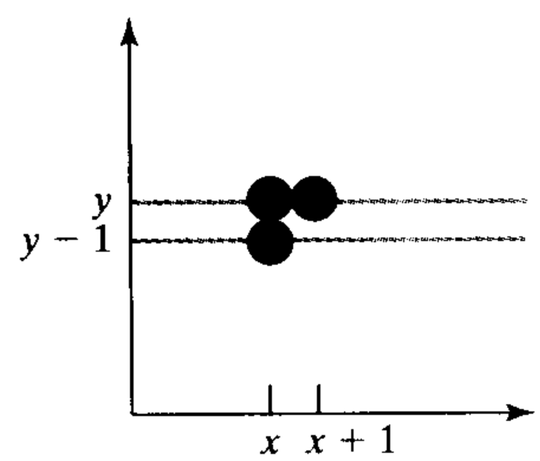


图 9.5 扫描线上某像素点(x, y)，其沿该线的相邻像素点坐标为$(x+1, y)$，其正下方相邻像素点坐标为$(x, y-1)$

对于每条扫描线，首先计算出与其相交的多边形的左边界交点所对应的深度值，如图9.6所示。该扫描线上的所有后继位置的深度可由等式（9.6）计算出来。

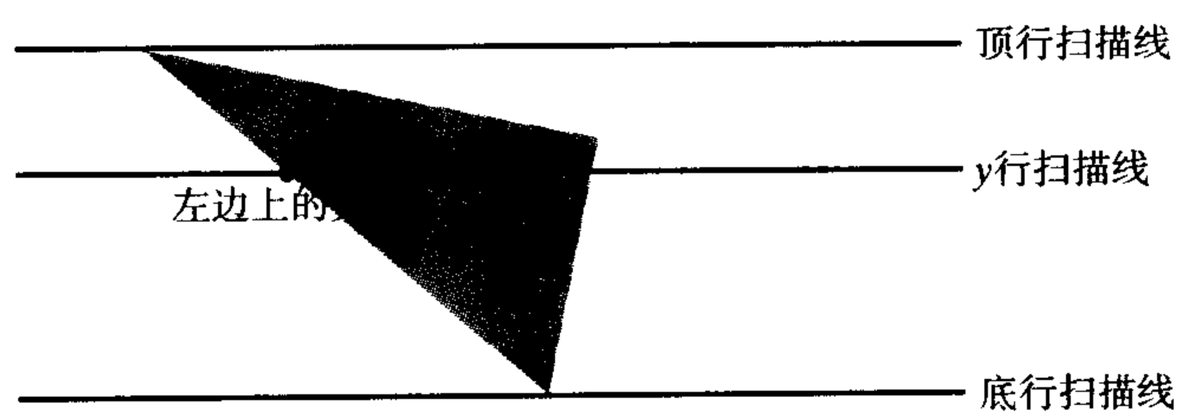


图 9.6 扫描线与多边形面相交

我们从多边形的最高顶点开始来实现深度缓存算法。接着，使用递归方式计算该多边形从该顶点往下在左面一条边上的x坐标。每一扫描行的开始位置x值可从上一扫描线的开始x值计算而得：

$$x' = x - \frac{1}{m}$$

其中m为该边斜率（参见图9.7）。沿该边还可递归计算出深度值：

$$z' = z + \frac{A/m + B}{C} \tag{9.7}$$

若沿一垂直边进行处理，则斜率为无限大，递归计算可简化为

$$z' = z + \frac{B}{C}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

另一种算法是采用中点法或Bresenham类型的算法，为每条扫描线确定最左边的x坐标。该算法还可用于处理曲面，计算各表面投影点的深度和颜色值。

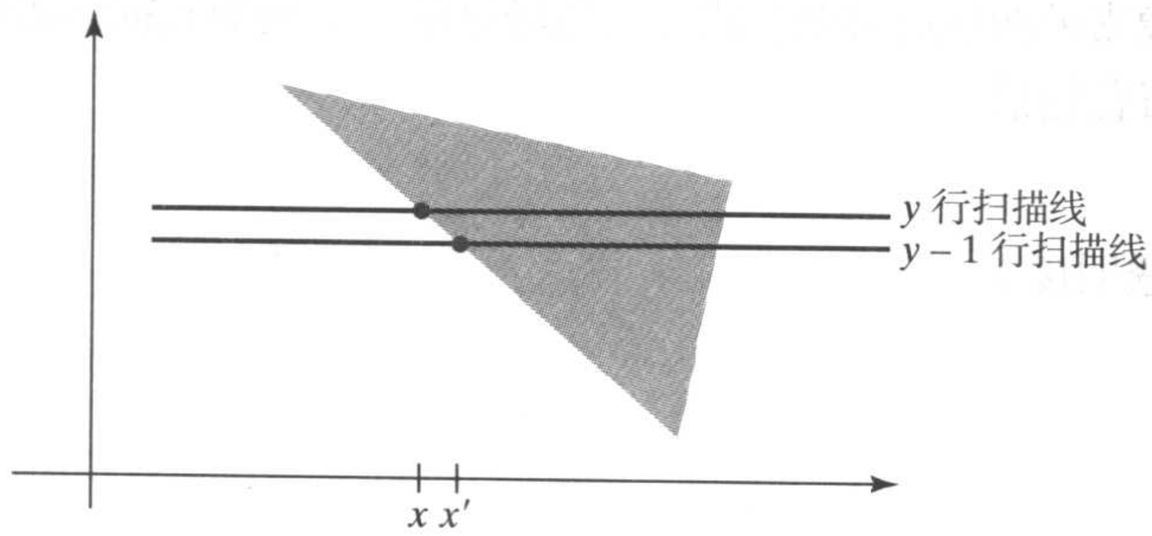


图 9.7　相邻扫描线与多边形左边界的交点

对于多边形面，深度缓存算法易于实现且无需将场景中的表面进行排序，但该算法除了刷新缓存之外还需另一个缓存。对于一个 1280 × 1024分辨率的系统，则需容量超过130万个位置的深度缓存，且每个位置需要包含表示深度值的足够位数。一个减少存储量需求的方案是，每次只对场景的一部分进行处理，这样就只需一个较小的深度缓存。在处理完一部分后，该缓存再用于下一部分的处理。

另外，基本的深度缓存算法经常执行一些不需要的计算。对象按任意次序处理，因此有些表面进行了颜色计算但事后又被更近的表面代替。为了缓减这一问题，有些图形软件提供选项让用户调整表面测试的深度范围。例如，通过深度测试排除较远的对象。使用该选项还可以排除非常靠近投影平面的对象。高档计算机图形系统一般集成了深度缓存算法的硬件实现。

9.4　A 缓存算法

A 缓存（A-buffer）算法是深度缓存算法的延伸（“z-buffer”中的z则表示深度）。这种深度缓存的扩充是一种反走样、区域平均、可见性检测方法，Lucasfilm在曲面绘制系统REYES（“Renders Everything You Ever Saw”）中成功实现了该算法。该过程的缓存区域称为累计缓存（accumulation buffer），因为它除了深度值外还用于存储各种表面数据。

深度缓存算法的一个缺点是：它在每个像素点只能找到一个可见面。即它只能处理非透明表面，而无法处理多个表面的累计强度值，但对于图 9.8 所示的透明表面，这种处理又是必需的。A 缓存算法对深度缓存进行了扩充，使其每一个位置均对应于一个表面链表。因此，不仅可以考虑各像素点处多个表面的强度值，还可以对对象的边界进行反走样处理。

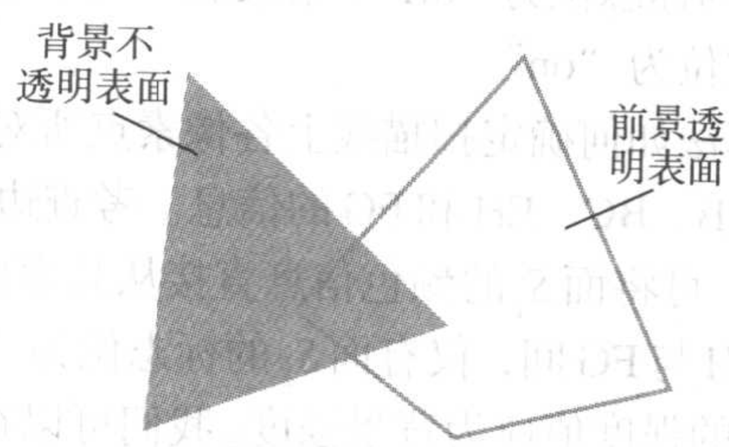


图 9.8　透过透明表面观察不透明表面，需要多个颜色输入并应用颜色混合操作

A 缓存中每个单元均包含两个域：

- 深度域——存储一个正的或负的实数；
- 强度域——存储表面的强度信息或指针值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

若深度域值为正，则该值表示覆盖该像素点的惟一表面的深度。表面数据场中存储各种表面信息，如该点的表面颜色和像素覆盖率，请参见图9.9(a)。如果A缓存某一位置的深度场是负的，则表明有多个表面对该像素的颜色有贡献。此时颜色场存储一个表面数据链表的指针，如图9.9(b)所示。A缓存中的表面信息包括

- RGB强度分量
- 透明性参数（透明度）
- 深度
- 覆盖度
- 表面的标识名
- 其他表面绘制参数

A缓存可以按照类似于深度缓存算法进行创建。即沿每条扫描线确定各像素点所对应的覆盖表面。表面可分割为多边形网格，并利用像素边界对其进行裁剪。采用不透明因子和表面覆盖度，以所有覆盖表面作用的平均值来计算每个像素点处的强度值。

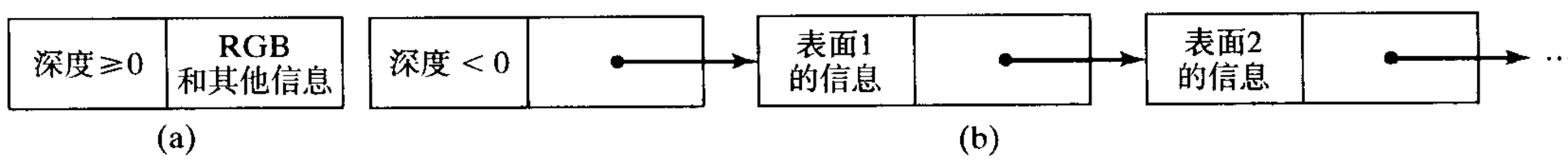


图9.9 A缓存像素单元的创建：(a)只有一个表面覆盖该像素；(b)有多个表面覆盖该像素

9.5 扫描线算法

这个像空间的隐面消除算法沿各扫描线计算并比较场景的深度值。逐条处理各条扫描线时，首先要判别与其相交的所有表面的可见性，然后计算各重叠表面的深度值以找到离观察平面最近的表面。一旦确定了某像素点所对应的可见面，可以得到该点的强度值，并将其置入帧缓存。

存储在多边形表（参见5.15节）中的信息用于处理表面。边表中包含场景中各线段的端点坐标、线段斜率的倒数和指向多边形表中对应多边形的指针；多边形表中则包含各多边形面的平面方程系数、表面材料特性、其他表面数据以及可能有的指向边表的指针。为了加速查找与扫描线相交的表面，可以在处理时建立一张活化边表。该表仅包含与当前扫描线相交的边，并将它们按x升序排列。另外，可为各个多边形面定义一个可设定为“on”或“off”的标志位，以表示扫描线上某像素点位于多边形内或多边形外。扫描线由左向右进行处理，在凸多边形的面投影的左边界处，标志位为“on”（开始），而右边界处的标志位为“off”（结束）。对于凹多边形，扫描线交点从左往右存储，每一对交点中间设定面标志位为“on”。

图9.10举例说明了扫描线算法如何确定扫描线上各像素点所对应的可见面。扫描线1所对应的活化边表中，包含了边表中边AB、BC、EH和FG的信息。考查边AB与BC间沿线的像素点，只有面S_1的标志位为“on”。因此，可将面S_1的颜色信息直接从其表面特性和列出的条件中计算，而无需计算深度值。同样，在边EH与FG间，仅有面S_2的标志位为“on”。而扫描线1的其余部分与所有表面均不相交，这些像素点的强度值应为背景强度。我们可以在初始化工作中将缓存的所有单元均置为背景强度。

对于图9.10中的扫描线2和3，活化边表包含边AD、EH、BC及FG。在扫描线2上边AD与EH间的部分，只有面S_1的标志位为“on”。而在边EH与BC的部分，所有面的标志位均为“on”。因此，在碰到边EH时必须用平面参数来为两个表面计算深度值。在本例中，假设面S_1的深度小于面S_2，因此将面S_1的颜色值沿扫描线置入像素中，直到边界BC。然后将面S_1的标志位置为“off”，

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

再将面 S_2 的颜色值置入刷新缓存，直到边 FG。一旦确定在边 EH 上的深度关系并假定面 S_2 在 S_1 之后，就不再需要另外的深度计算。

逐条处理扫描线时，可利用线段的连贯性。如图 9.10 所示，扫描线 3 与扫描线 2 具有相同的活化边表。由于线段交点没有发生变化，因此无需在边 EH 与 BC 间再进行深度计算。两个表面必须处于相同的方向，类似于扫描线 2 中的情况，因此将面 S_1 置入刷新缓存而无需更多的计算。

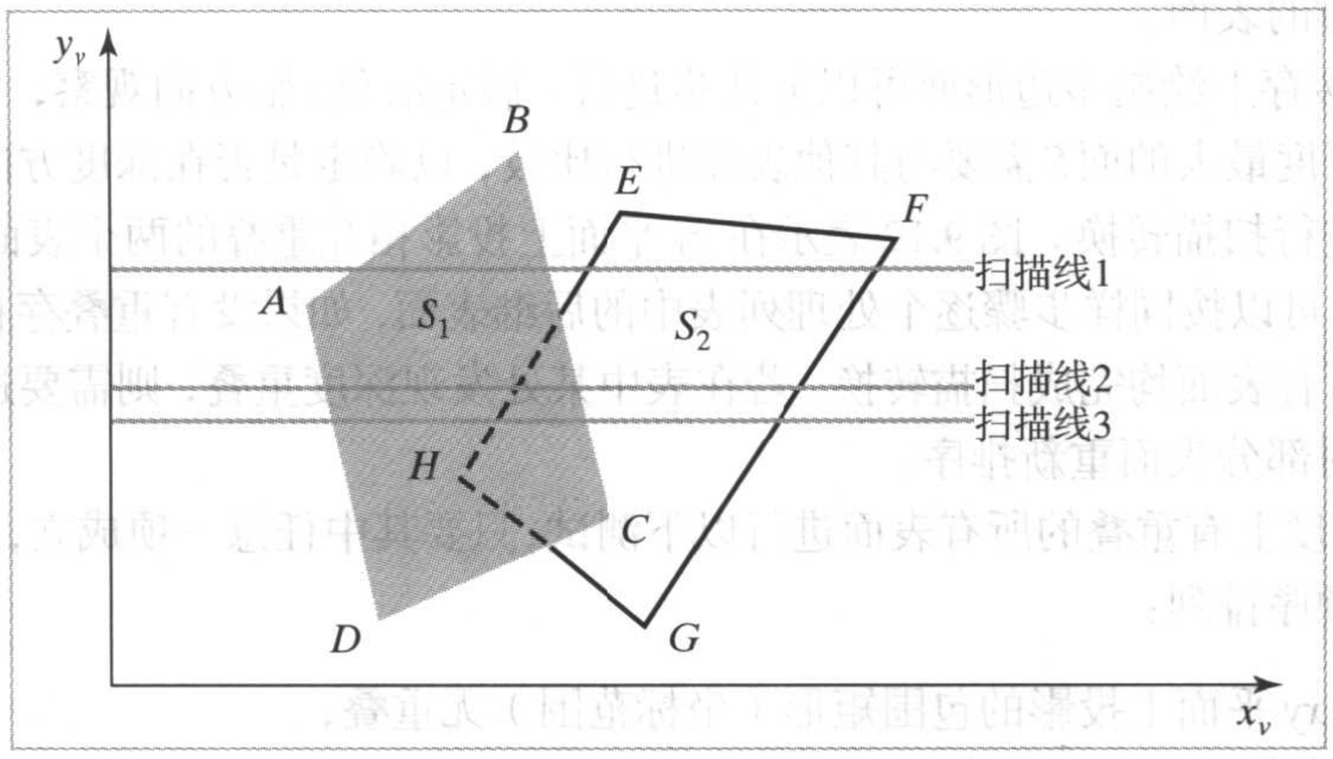


图 9.10　扫描线与面 S_1、S_2 在观察平面上的投影相交，虚线表示隐藏面的边界

扫描线算法可以处理任意数目的相互覆盖的多边形面。设置面标志可以表示某点与平面内外侧的位置关系，当表面间有重叠部分时，需要计算它们的深度值。只有在表面没有相互贯穿或循环遮挡的情况下，上述处理才能获得正确结果（参见图 9.11）。若场景中出现循环遮挡，则需要将表面进行划分以消除循环，图 9.11 中的虚线表示可以在此处将表面分割为两个独立部分，以消除循环遮挡。

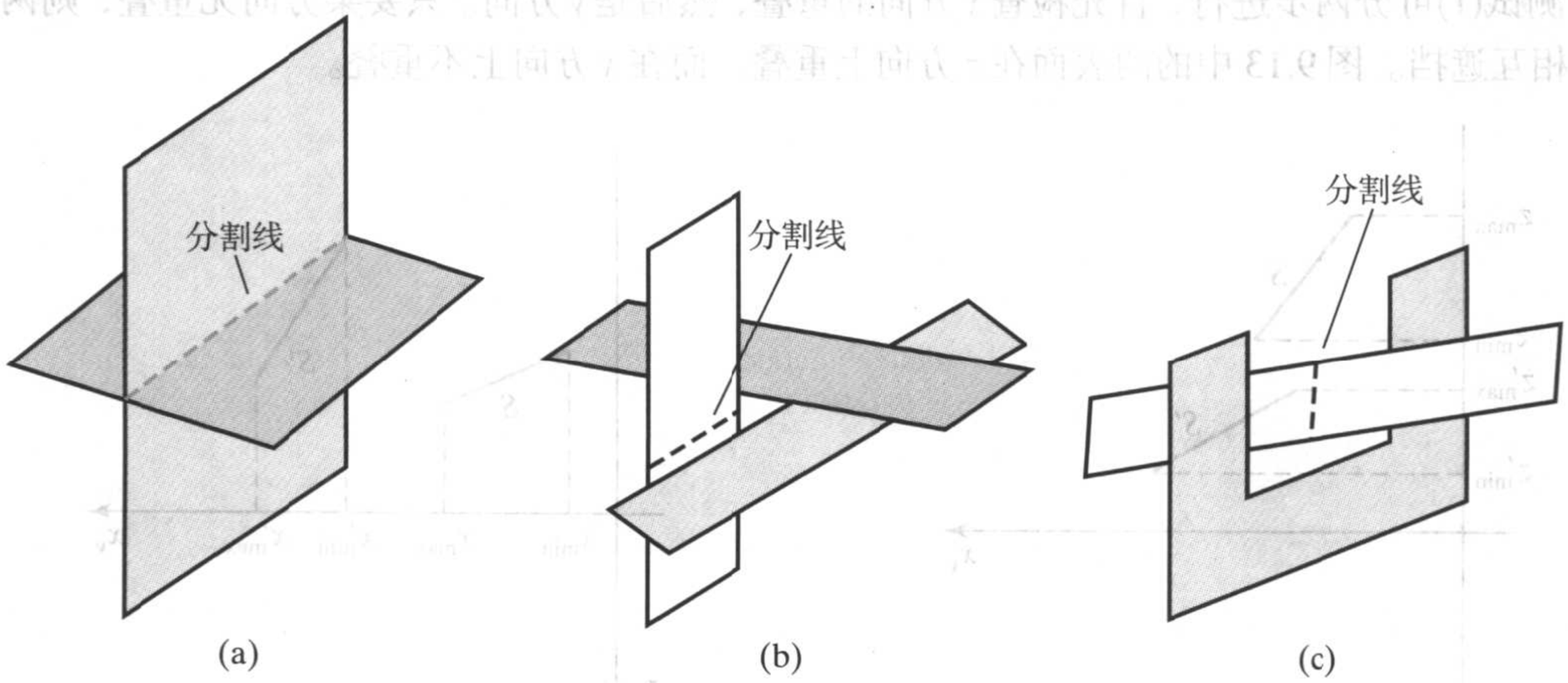


图 9.11　表面相交并循环遮挡的情况

9.6　深度排序算法

深度排序（depth-sorting）算法同时运用物空间与像空间操作，以实现以下基本功能：

(1) 将表面按深度递减方向排序；

(2) 由深度最大的表面开始，逐个对表面进行扫描转换。

排序操作既可在像空间和也可在物空间完成，而多边形面的扫描转换仅在像空间完成。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

这种隐藏面消除算法通常称为**画家算法**（painter' s algorithm）。画家在绘制一幅图画时，总是先在画纸上涂上背景色，然后画上远处的对象，接着是近一些的对象，以此步骤，最后画上最近的对象。这样，最近的对象覆盖了部分背景色和远处的对象，每一层总是在前一层的景物上覆盖。采用同样的技术，我们首先将表面根据它们与观察平面的距离排序，然后在刷新缓存中置入最远处表面的颜色值，接着按深度递减顺序逐个选取后继表面，并将其属性值“涂”在帧缓存上，从而覆盖了部分前面处理过的表面。

按深度在帧缓存上绘制多边形面可以分几步进行。假定沿负 z 轴方向观察，表面按它们 z 坐标的最低值排序，深度最大的面 S 需要与其他表面进行比较，以确定是否在深度方向存在重叠，若没有重叠，则对 S 进行扫描转换。图 9.12 表示在 xy 平面上投影相互重叠的两个表面，但它们在深度方向上没有重叠。可以按同样步骤逐个处理列表中的后继表面，如果没有重叠存在，则按深度顺序处理表面，直至所有表面均完成扫描转换。若在表中某处发现深度重叠，则需要进行一些比较，以确定是否有必要对部分表面重新排序。

可对与S在深度上有重叠的所有表面进行以下测试，只要其中任意一项成立，则无需重新排序。测试按难度递增顺序排列：

(1) 两表面在 xy 平面上投影的包围矩形（坐标范围）无重叠；
(2) 相对于观察位置，面 S 完全位于重叠表面之后；
(3) 相对于观察位置，重叠表面完全位于面 S 之前；
(4) 两表面在观察平面上的投影无重叠。

按以上顺序逐项进行测试，一旦发现某项测试结果为真，则处理下一重叠表面。若所有重叠表面均至少满足一项测试，则它们均不在面 S 之后，因而无需进行重新排序，可直接对 S 进行扫描转换。

测试(1)可分两步进行，首先检查 x 方向的重叠，然后是 y 方向。只要某方向无重叠，则两表面不会相互遮挡。图 9.13 中的两表面在 z 方向上重叠，而在 x 方向上不重叠。

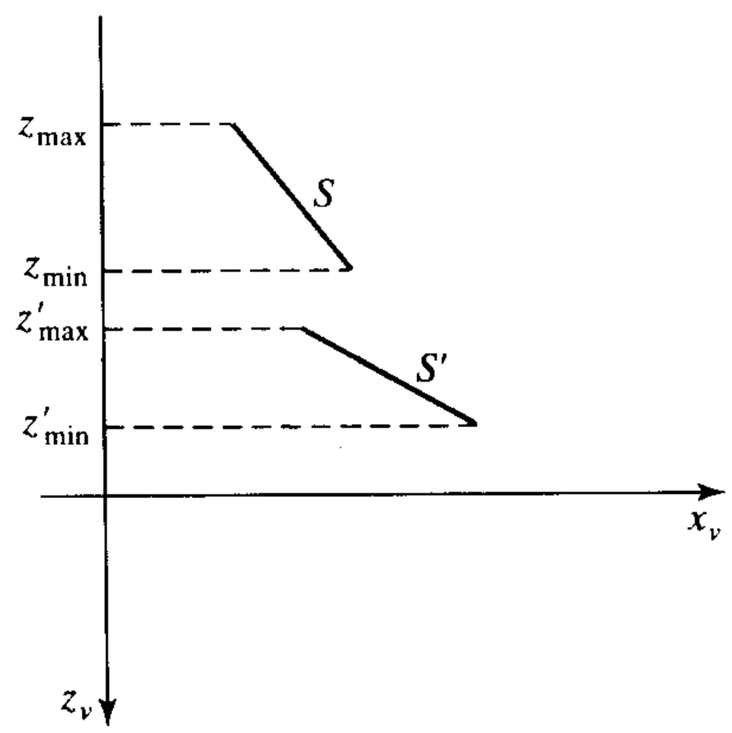


图 9.12 两表面无深度重叠

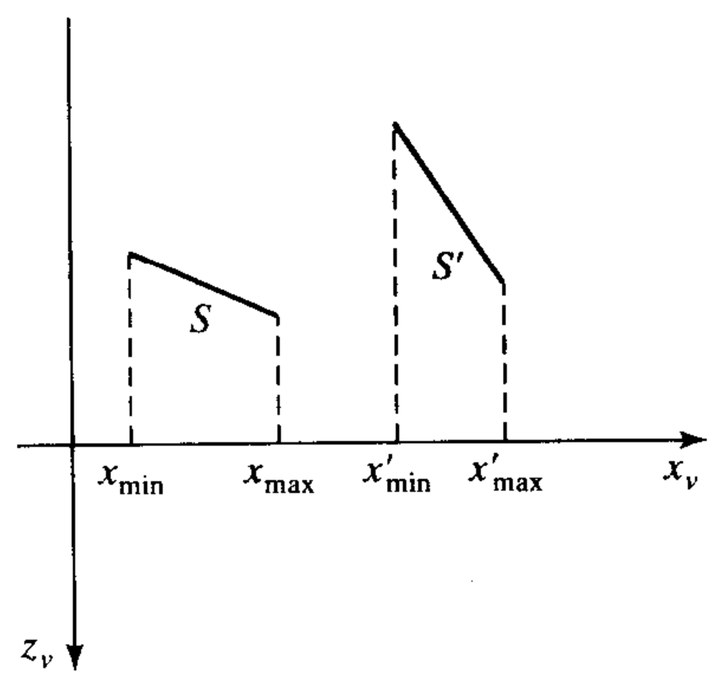


图 9.13 两表面在 x 方向上无重叠，但有深度重叠

我们可以借助后向－前向多边形测试法来实现测试(2)和(3)。即将面 S 各顶点坐标代入重叠表面的平面方程中，以检查结果值的符号。若建立平面方程使表面的外表面正对着视点，则只要 S 的顶点在 S' 的后面，那么面 S 位于 S' 之后（参见图 9.14）；同样，若面 S 的所有顶点均在面 S' 的前面，则 S' 完全位于面 S 之前。图 9.15 表示一个重叠表面 S' 完全位于 S 之前，但 S 并非完全在 S' 的后面（测试(2)为假）。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

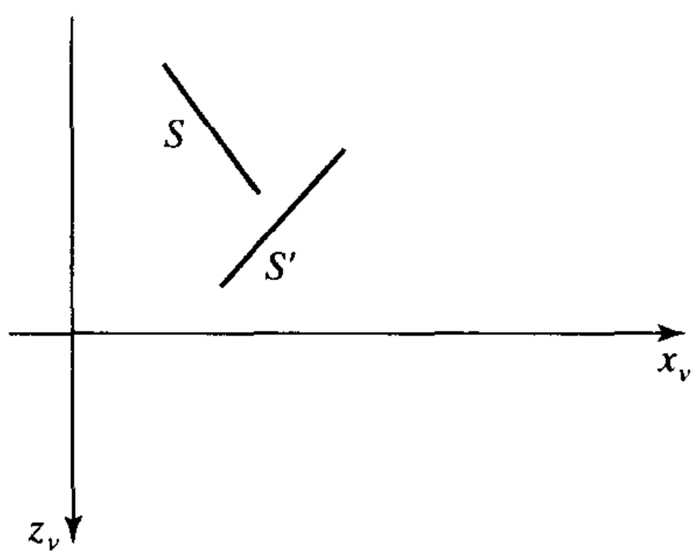


图 9.14　面 S 完全位于重叠面 S' 之后

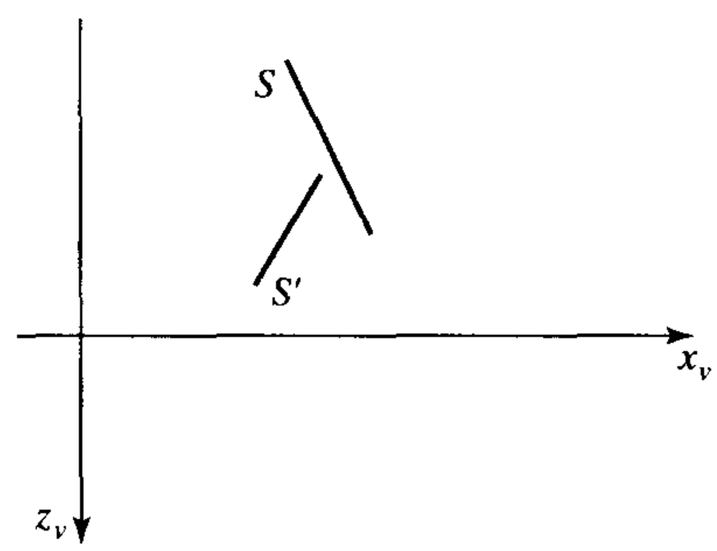


图 9.15　重叠面 S' 完全位于面 S 之前，但 S 并非完全位于 S' 之后

若测试(1)至(3)均为假，则需执行测试(4)来确定两表面投影是否重叠。如图9.16所示，即使两表面的坐标范围均重叠，它们也可能不相交。

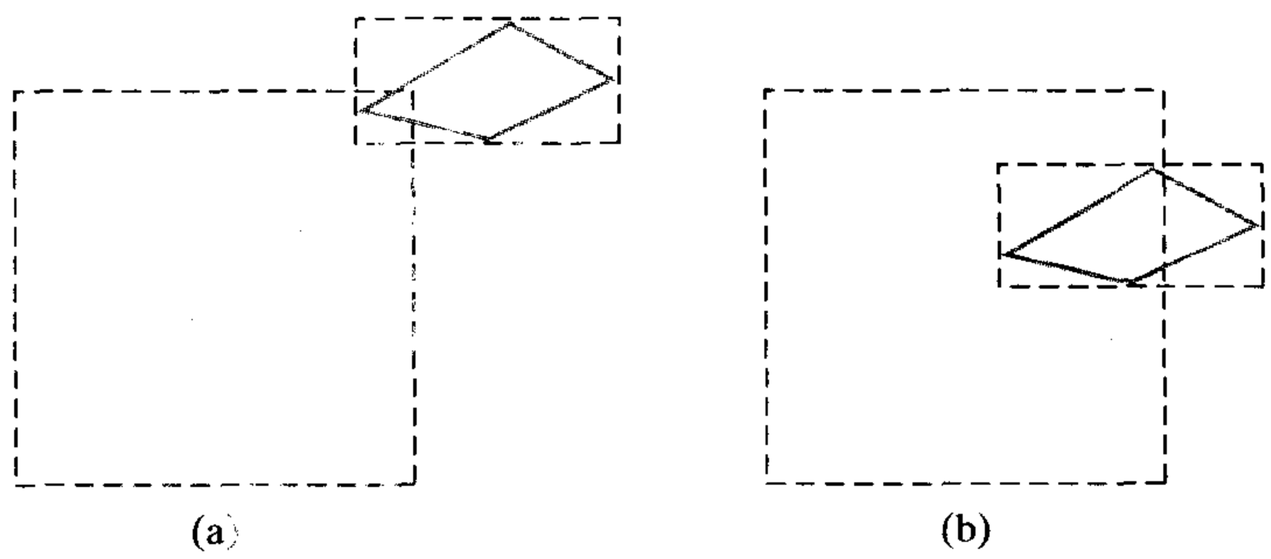


图 9.16　两表面在 xy 平面上投影的包围矩形相互重叠

如果对于某一重叠表面，所有四项测试均不成立，则需要在有序表中调换 S 与 S' 的顺序。图9.17给出了两个表面应重新排序的例子，此时，仍无法确切知道是否已发现离观察平面最远的表面。图 9.18 表示应首先调换 S 与 S''，但由于 S'' 遮挡了 S' 的一部分，因此还需要调整 S'' 与 S'，以得到三个表面的正确深度顺序。因此，我们需要对调换过顺序的表面重复以上四项测试。

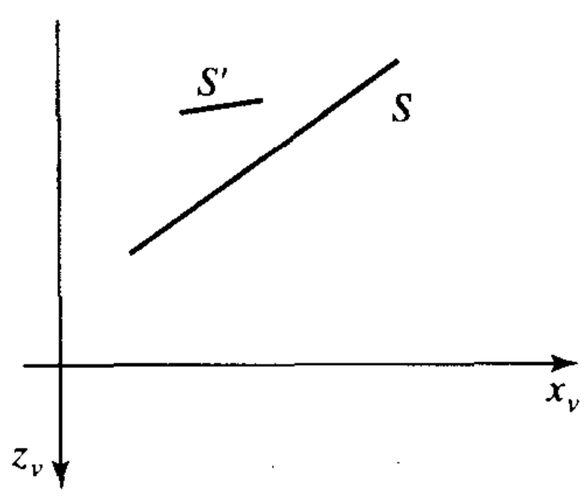


图 9.17　面 S 深度大于面 S'，但遮挡 S'

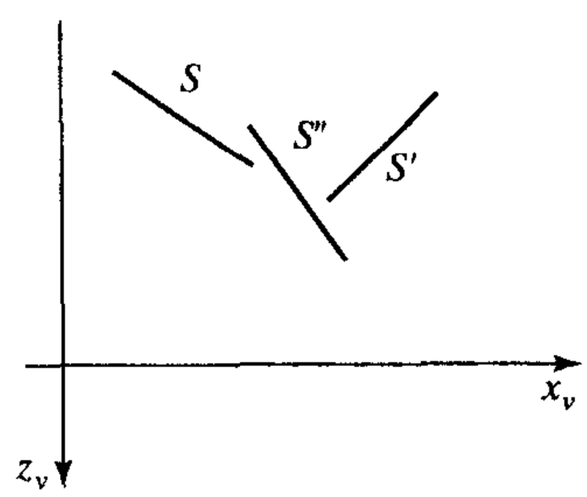


图 9.18　三表面存入有序表面表时的顺序为 S、S'、S''，但需要重新排序为 S'、S''、S

若两个或多个表面循环遮挡（参见图 9.11），则该算法可能导致无限循环。此时，算法反复将重叠表面的位置进行组合。为了避免死循环，可以标识那些在重排序时调至更远位置的表面，使其不再被移动。如果需要将一表面进行第二次调换，则将其分割为两部分以消除循环遮挡。原来的表面被一分为二后，继续执行上述处理。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

9.7 BSP 树算法

BSP（binary space-partitioning）树算法是一种判别对象可见性的有效算法。该算法类似于画家算法，将表面由后往前地在屏幕上绘出。该算法特别适用于场景中对象位置固定不变、仅视点移动的情况。

利用BSP树来判别表面的可见性，其主要操作是在每次分割空间时，判别该表面相对于视点与分割平面的位置关系，即位于其后面还是前面。图9.19表示了该算法的基本思想。首先，平面P_1将空间分割为两部分，相对于观察方向，一组对象位于P_1的后面而另一组则在P_1之前。如果某对象与P_1相交，则立刻将其一分为二并分别标识为A和B。此时，图中A与C位于P_1之前，而B和D在P_1之后。平面P_2对空间进行了二次分割，并生成如图9.19(b)所示的二叉树表示。在这棵树上，对象用叶节点表示，分割平面前方的对象组作为左分支，而后方的对象组为右分支。

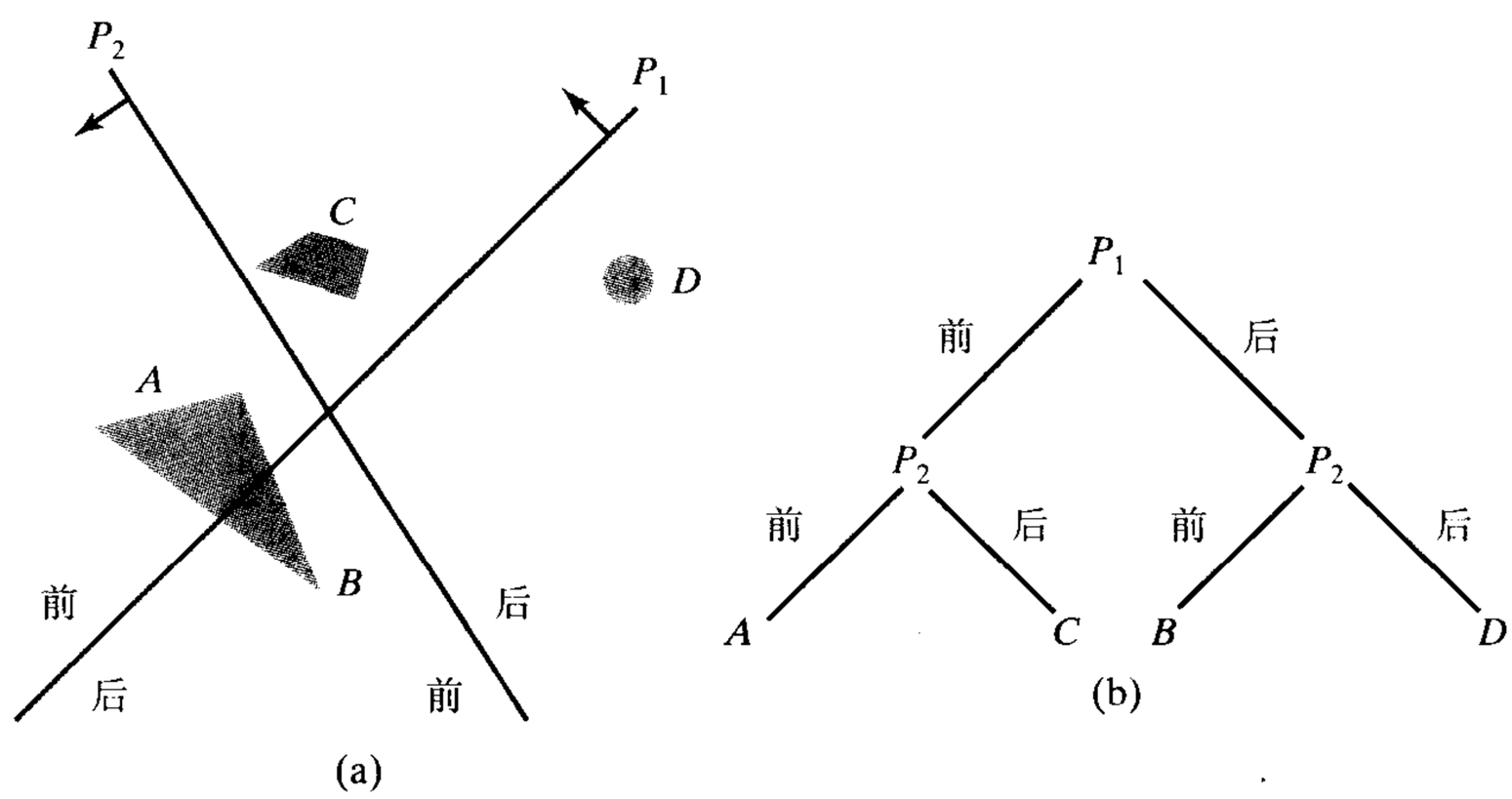


图 9.19 一个空间区域(a)被平面P_1和P_2分割，形成(b)中的BSP树表示

对于由多边形面组成的对象，可以选择与多边形面重合的分割平面，利用平面方程来区分后面的和前面的多边形顶点。随着将每个多边形面作为分割平面，可以生成一棵树。与分割平面相交的每个多边形将被分割为两部分。一旦BSP树创建完毕，即可按先右节点后左节点的次序处理该树。该树上的面按由后往前次序显示，因此前面的对象覆盖后面的对象。目前已有许多系统借助硬件来完成BSP树创建和处理的快速实现。

9.8 区域细分算法

这一算法虽然本质上是一种像空间算法，但它也使用了一些物空间操作来完成表面的深度排序。**区域细分算法**（area-subdivision method）通过对代表单个表面部分的投影区域定位而充分利用了场景中区域的连贯性。应用此方法连续地将整个观察平面区域细分为越来越小的矩形单元，直至每个矩形区域仅包含单个可见面的投影、不含任何表面或该区域只有一个像素大小。

为了实现区域细分算法，必须首先找到一种区域测试手段，可以很快地判别出某一区域仅包含某一表面的一部分，或告诉用户该表面太复杂、难以分析。从整个视图开始，应用该测试来确定是否应将一完整区域分割为一些小矩形单元。若测试表明视图相当复杂，则需将其分割，然后再对各个小区域进行测试。若测试表明还无法确定表面的可见性，则必须再次细分区域，直至最终的区域

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

易于分析，即它属于某一单个表面或到达分辨率的极限。如图9.20所示，一个简单的实现算法是每次将区域分割为四块大小相等的矩形。该方法类似于组织一棵四叉树。这样，即使是对一个1024 × 1024分辨率的视图细分 10 次以后，也能使每个单元覆盖一个像素。

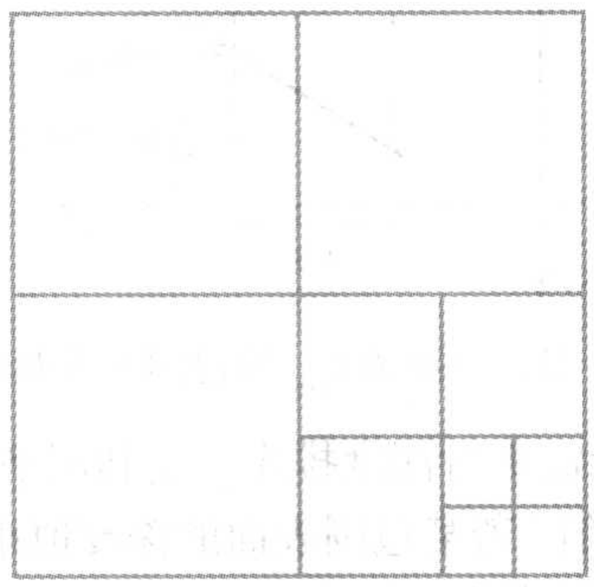


图 9.20　每次将一正方形区域细分为四个大小相同的小单元

一个表面与一个细分后的观察平面区域的关系有四种。可用下列分类来叙述这些相关表面位置（参见图 9.21）：

包围表面（surrounding surface）——完全包含该区域的表面。
重叠表面（overlapping surface）——部分位于该区域内，部分位于该区域外的表面。
内含表面（inside surface）——完全在该区域内的表面。
分离表面（outside surface）——完全在该区域外的表面。

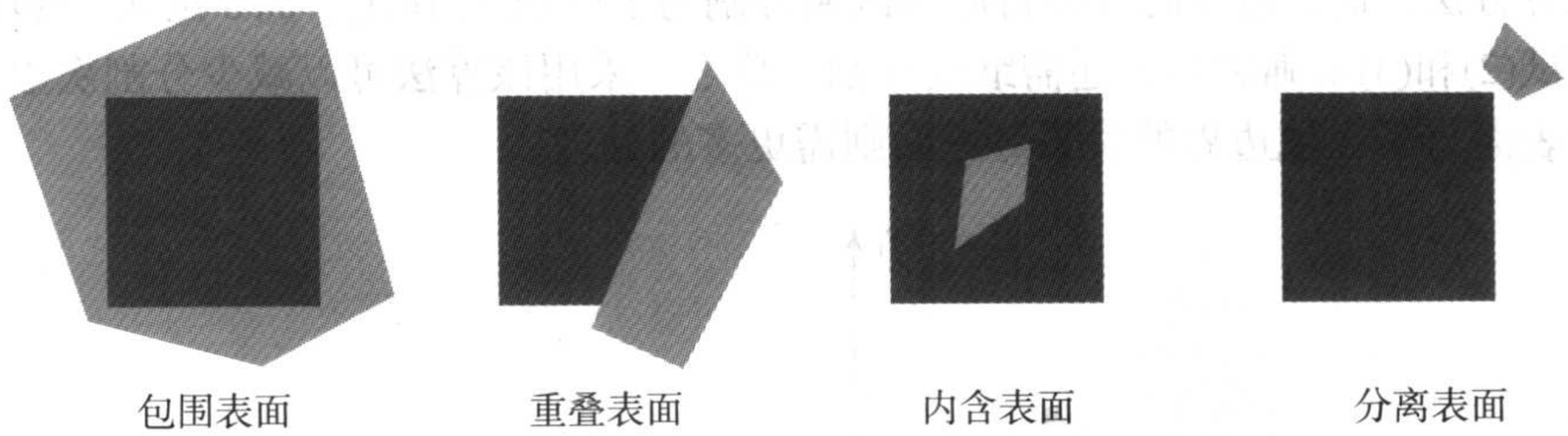


图 9.21　多边形表面与观察平面矩形区域之间可能的关系

可以根据图 9.21 中的四种类别来给出表面的可见性测试。若以下条件之一为真，则无需再对区域进行分割：

条件 1：该区域没有内含、重叠或包围表面（所有表面均为区域的分离表面）。
条件 2：该区域只有一个内含表面、重叠表面或包围表面。
条件 3：该区域有一个在其边界内遮挡了其他所有表面的包围表面。

开始，我们可以检查所有表面的包围矩形与区域边界的关系。这可识别内含和包围表面，但需要求交操作才能识别重叠和分离表面。若包围矩形与区域边界有交点，则还需进行另外的检查来判别表面是否为包围型、重叠型或分离型，一旦判别出某单个表面是内含、重叠或包围型的，就将其像素强度值置入帧缓存的相应位置。

实现测试(3)的方法将表面根据它们离观察平面的最近距离进行排序。对于所有考查区域内的包围表面，计算其最大深度。如果某一包围表面距观察平面的最大深度小于该区域内其他所有表面的最小深度，则满足测试(3)，图 9.22 给出了该情况的一个例子。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

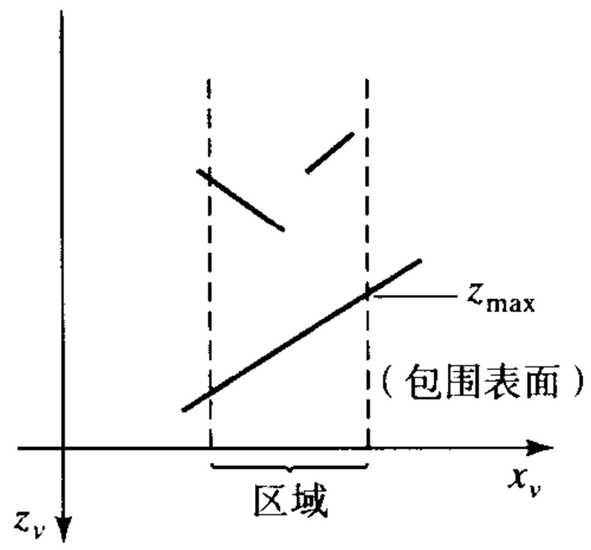


图 9.22　在某特定区域内，一个最大深度为 z_{max} 的包围表面遮挡所有最小深度超过 z_{max} 的表面

另一种实现测试(3)的算法则无需进行深度排序，它利用平面方程计算所有包围表面、重叠表面和内含表面上区域四顶点的深度值。若某包围表面的深度值小于其他所有表面，则测试(3)结果为真，即该区域可以利用包围表面的强度值进行填充。

有些情况下，前面两种测试方法均无法正确判别出遮挡其他所有表面的包围表面，可能还需借助于进一步的测试。此时，采用区域细分的办法比继续进行较复杂的测试要快得多。一旦判别出某区域中的包围表面和分离表面，那么对该区域细分后的子区域仍然保持包围型或分离型的位置关系。随着逐步细分，一些内含表面和重叠表面将被消除，因而区域将越来越易于分析。少数情况下，最终的分割区域为像素大小，那么只需计算当前点的相关表面深度，并将最近的表面强度值置入帧缓存中。

我们可以将基本分割处理进行变形，不再简单地将区域一分为四，而是沿表面边界对区域进行分割。如果表面已完成最小深度排序，则可以利用最小深度的表面对给定区域进行划分。图9.23给出了这种细分方法。面 S 边界的投影将原来区域分割为子区域 A_1 和 A_2，面 S 是 A_1 的包围表面。此时，可由测试(2)和(3)来确定是否还需继续分割。总之，采用该算法可以减少分割次数，但在区域细分及分析表面与子区域边界的关系等方面则需更多的处理。

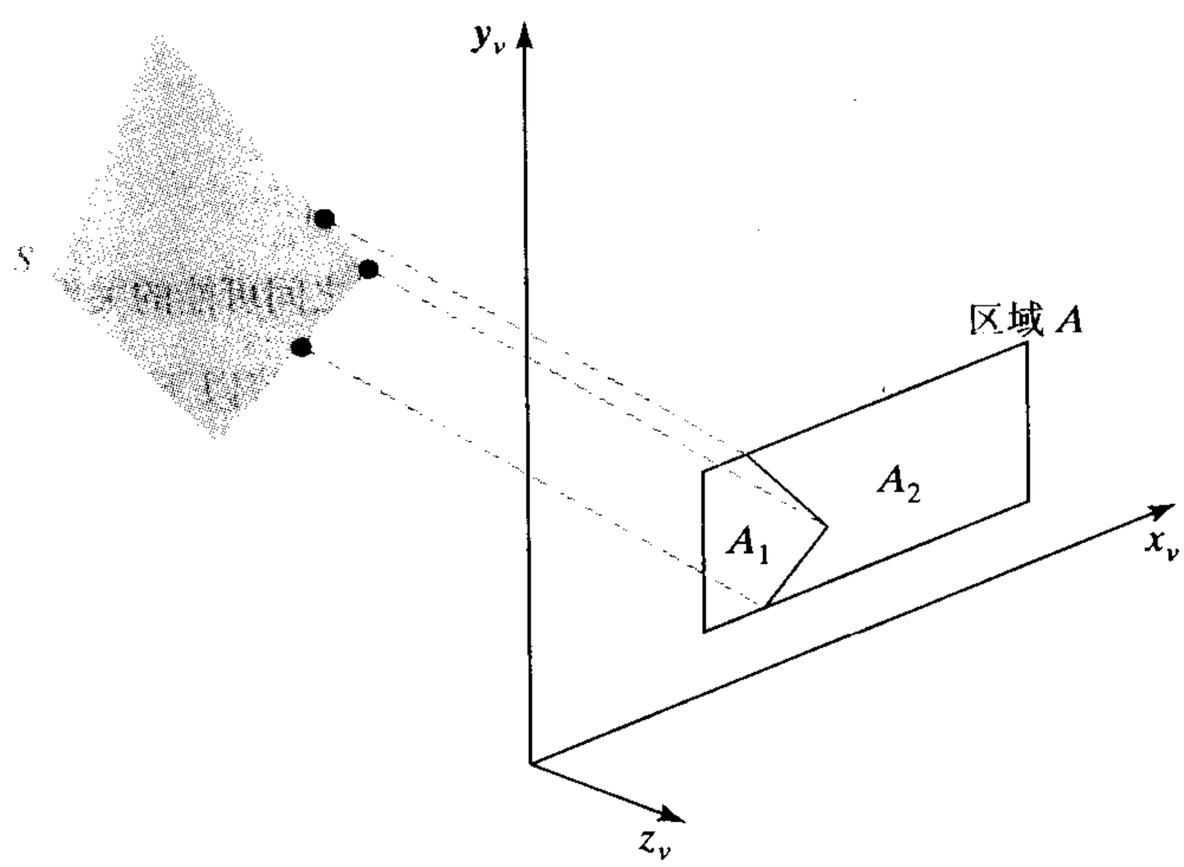


图 9.23　观察面 S 的边界投影将区域 A 细分为 A_1 和 A_2

9.9　八叉树算法

当按照八叉树表示来描述观察体时，通常按由前往后的顺序将八叉树节点映射到观察表面，从而消除隐藏面。在图 9.24 中，空间区域的前部（相对于视点）为体元0、1、2、3。体元的前表面均可见，这些体元尾部的表面和后部体元（4、5、6、7）的表面都可能被前部的表面所遮挡。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于图 9.24 所示的观察方向，按体元顺序 0、1、2、3、4、5、6、7 来处理八叉树节点中的数据可消除隐藏面，这实际上是对八叉树的深度优先遍历。因此，将在节点 4、5、6、7 之前访问代表体元 0、1、2、3 的节点。同样，0 号体元中的前面 4 个子体元将在后面 4 个子体元之前被访问，对于每个八分体元的八叉树遍历将按照这个顺序进行。

如果要在一个八叉树节点中设置颜色值，则仅当帧缓存中对应于该节点的像素位置尚未设置值时才置入。这样，缓存中仅置入前面的颜色值。如果这个位置是无效的，则不置入任何值，任何被完全遮挡的节点都将不再对其处理，也不会再访问它的子树。图 9.25 描述一个空间区域内的八分体元和相应观察平面上对应的四分面元。八分体元 0 和 4 直接影响四分面元 0，四分面元 1 的颜色值从八分体元 1 和 5 获得。另外两个四分面元的值可以分别从与之对应的一对八分体元中得到。

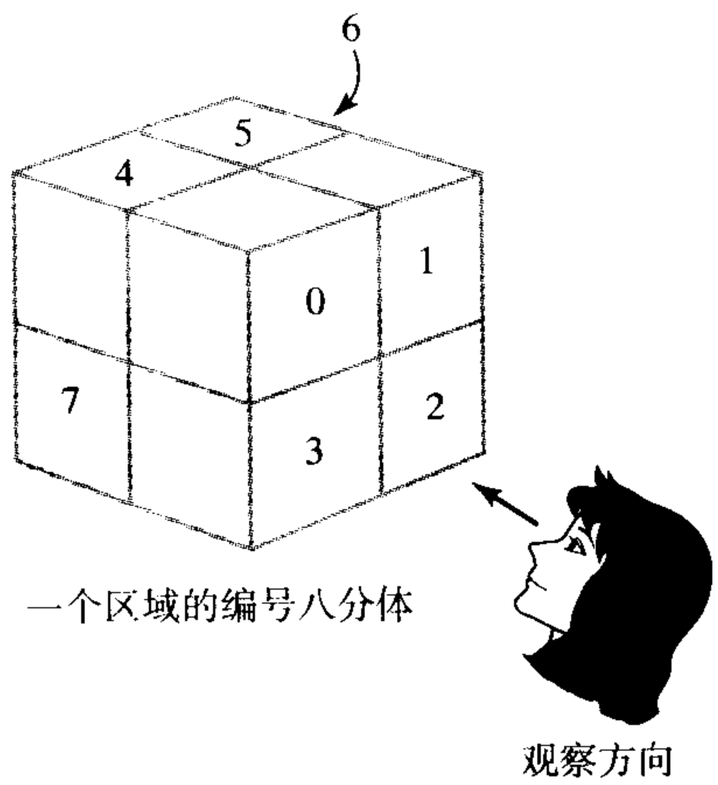


图 9.24　对于当前视线，体元 0、1、2、3 中的对象总是遮挡后面体元（4, 5, 6, 7）中的对象

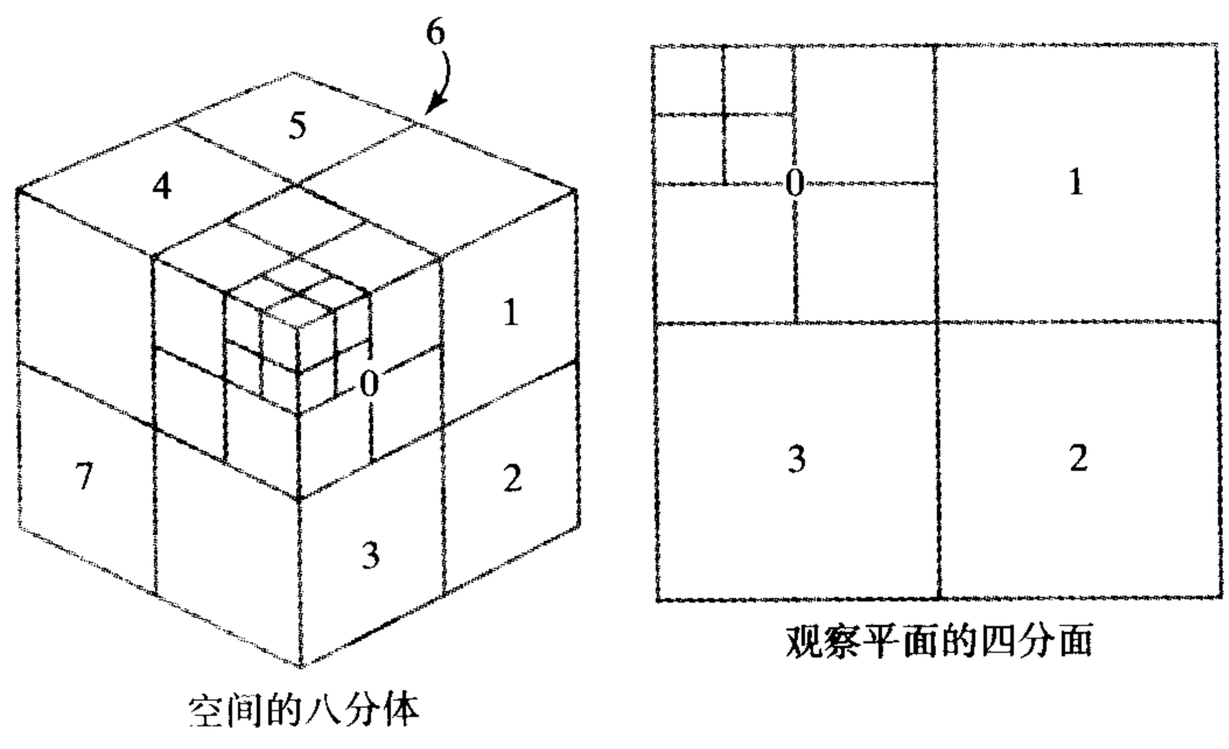


图 9.25　某空间区域的八叉分割及相应的四叉平面

高效的八叉树可见性测试通过递归地处理八叉树节点并生成可见面的四叉树表示来实现。多数情况下，综合考虑前面的和后面的八分体元以确定正确的四分体元颜色。但如果前面的八分体元用某种单一颜色填充，则不必再处理后面的八分体元。对一个单一颜色区域，调用一个递归过程，将单一颜色八分体元的子体元和新建立的四分节点作为新变量传递过去。如果前面是空的，则必须仅处理后面八分体元的子体元。否则，出现两个递归调用，一个为后面的而另一个为前面的八分体元。

通过对八叉树表示的变换按选定视图对对象重定向，可获得用八叉树表示的对象的不同视图。这时八分体元重新编号，以保持八分体元 0、1、2 和 3 始终在前面。

9.10　光线投射算法

如果考查由视点出发穿过观察平面上一个像素而射入场景的一条射线（参见图 9.26），则可以确定出场景中与该射线相交的对象（如果有）。在计算出光线与对象表面的所有交点之后，离像素最近的交点所在的表面即为可见面。这种可见性判别模式应用了 8.20 节中所介绍的光线投射（ray casting）算法。光线投射建立于几何光学的基础之上，它沿光线的路径追踪，是一种有效的可见性判别手段。由于场景中有无限多条光线，而我们仅对穿过像素的光线感兴趣，因此可考虑从像素出发，逆向跟踪射入场景的光线路径。光线投射算法对于包含曲面、特别是球面的场景有很高的效率。

光线投射算法可以看做是深度缓存算法（参见 9.3 节）的一种变形。在深度缓存算法中，每次处理一个表面并对表面上的每个投影点计算深度值。计算出来的值与以前保存的深度值进行比较，

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

从而确定每个像素所对应的可见表面。在光线投射算法中，每次处理一个像素，并沿光线的投射路径计算出该像素所对应的所有表面的深度值。

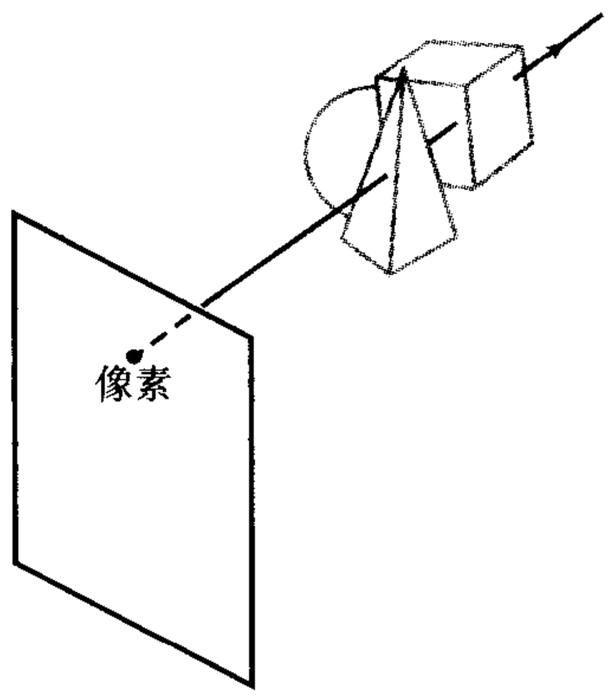


图 9.26 一条由像素点射入场景的视线

光线投射是光线跟踪算法（参见 10.11 节）的特例。光线跟踪技术通过追踪多条光线在场景中的路径，从而得到多个对象表面所产生的反射和折射效果。而在光线投射中，跟踪的光线仅从每个像素到最近的对象为止。人们已经为一些常见的对象，特别是球体，开发出高效的光线与对象表面的求交计算算法，我们将在 10.11 节对此进行详细讨论。

9.11 可见性检测算法的比较

可见面检测方法的效率依赖于具体应用的特点。如果场景中的表面沿视线的分布较广从而在深度上重叠较少，则深度排序或BSP树方法的效率一般较高。在投影面上的表面投影有较少投影时，扫描线或区域细分方法是较快定位可见面的方法。

作为一个通用规则，深度排序算法和 BSP 树方法对有较少面的场景有较高的效率。这是因为这些场景在深度上重叠的面较少。扫描线算法对有较少面的场景也有较高效率。扫描线算法、深度排序算法或BSP树方法对几千个多边形面以下的场景检测可见面的效率较高。对几千个面以上的场景，深度缓存算法或八叉树方法的效果最好。深度缓存算法的处理时间比较稳定，与场景中面的数量无关。这是因为当面的数量增加时，面区域的尺寸减少。因此，深度缓存算法对于简单场景表现为低性能，而对于复杂场景表现为高性能。BSP 树在使用不同观察参考点生成多个视图时较有效。如果场景包含曲面表示，则使用八叉树方法或光线投射法来识别场景中的可见面。

当系统中使用八叉树方法时，可见性检测处理快速而简单。其中仅使用整数加减法，且不需要排序和求交计算。八叉树方法的另一优点是存储了表面几何以外的信息。对象的整个实体区域都可用于显示，从而使八叉树表示可用于获得三维对象的剖切面。

可以将不同的可见面检测方法以多种形式结合并实现。另外，可见性检测算法常用硬件实现，而使用并行处理技术的专用系统可用来提高这些方法的效率。专门的硬件在处理速度是特别重要的考虑因素时应用，如在飞行模拟器动画视图的生成中。

9.12 曲面

对于曲面对象，最有效的可见性判别算法为光线投射算法和八叉树算法。在光线投射算法中，我们首先计算出光线与表面的交点，然后找出沿光线方向离像素最近的交点。而在八叉树算法中，

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

从前往后简单地对节点排序来确定表面颜色值。一旦根据对象的定义建立起相应的八叉树表示，即可用同样的算法对可见面进行判别，而无需对不同的曲面类型做专门的考虑。

我们常常利用一组多边形网格来近似表示曲面，然后使用以前讨论的隐藏面消隐算法进行处理。对于其他一些对象，例如球体，使用光线投射和曲面方程则会更高效和更准确。

9.12.1　曲面表示

可以利用一个隐式方程 $f(x, y, z) = 0$ 或参数方程（参见附录 A）来表示一个曲面。例如，通常利用参数方程来描述样条曲面。在一些情况下，也可能需要显式的曲面方程，如表示 xy 平面上的高度函数：

$$z = f(x, y)$$

我们感兴趣的许多对象都有其二次方程表示，如球体、椭球体、圆柱体、锥体等。通常将其用于分子结构、滚桶、环、轴等对象的建模。

在扫描线算法和光线投射算法中，数值近似技术也经常用于解决曲面与扫描线或光线之间的求交操作。另外，为了求解常见对象的曲面相交方程，人们开发了并行计算、快速硬件实现等技术。

9.12.2　曲面的层位线显示

在数学、物理、工程及其他领域的应用中，经常使用一组表示曲面形状的线来显示曲面函数。可以使用方程来描述曲面，也可以用数据表来表示曲面，如海拔高度或人口密度的拓扑数据。根据显式的曲面函数，可以画出可见面的层位线，并消除层位中的隐藏部分。

为了得到曲面的 xy 平面投影图，可以按照以下形式来表示曲面：

$$y = f(x, z) \tag{9.8}$$

在 xy 平面上，可能根据选定范围的 z 值和设定好的间距 Δz 绘制一组曲线。首先，由 z 的最大值开始，从前往后地逐个将函数值的 xy 范围映射至 xy 屏幕像素范围，并画出相应的曲线段，同时删除隐藏部分。然后，对于某一给定 z 值，按一定步长变化 x 值，并根据等式（9.8）计算出相应的 y 值。

一个判别曲面上可见曲线的算法是保存一张列表，记录所计算过的、对应于屏幕上水平位置为 x 的像素坐标的 y_{min} 和 y_{max} 值。当从某像素行进至下一像素时，需要将所求的 y 值与存储的 y_{min} 和 y_{max} 进行比较，若 $y_{min} \leqslant y \leqslant y_{max}$，则该点不可见，无需画出；若 y 值在当前存储的 y 范围之外，则该点可见，画出该点并对该像素位置重新设置 y 边界。按照同样的算法，可以将层位线投射至 xz 与 yz 平面。图 9.27 显示了一个曲面层位线图。

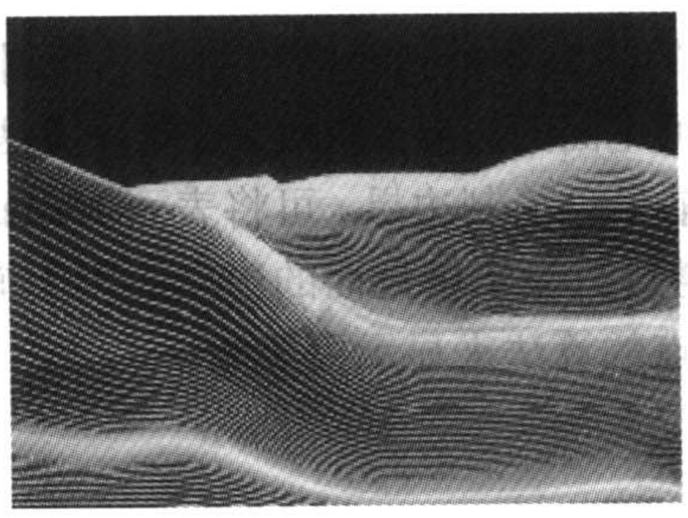


图 9.27　曲面层位线图（Los Alamos 实验室提供）

采用以上算法，可以处理通过确定等值线所得到的一组离散数据点。例如，若 xy 平面上 $n_x \times n_y$ 网格有一组离散的 z 值，则可由 8.27 节讨论的层位线算法得到曲面上 z 层位线的路径。每条层位线

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

将投影至观察平面并用直线段进行显示。同样，可以沿着从前至后的深度顺序将线段画在显示设备上，并消除在前面已显示（可见）层位线后面经过的层位线段。

9.13 线框可见性算法

场景往往不包含孤立的线条，除非显示条状图、流程图或网络布局图。但是我们常常为了快速显示对象特征而仅用轮廓线形式显示三维场景。生成场景线框图最快的方法是显示对象所有的边。然而，在这样的显示中很难确定对象的前后特征。一种解决方法是使用深度提示，让线段的显示强度成为它离开观察者距离的函数。另一种方法是应用可见性测试，将隐藏线消除或使用与可见边不同的方式进行显示。确定对象边可见性的过程通常称为**线框可见性算法**（wire-frame visibility method），也称为**可见线判别算法**（visible-line detection method）或**隐藏线判别算法**（hidden-line detection method）。另外，前面章节讨论的可见面判别算法也可用于边的可见性测试。

9.13.1 线框面可见性算法

判别场景中可见线的最直接算法是，依次将每条边与各个面进行比较。这个过程与任意形状窗口的线裁剪十分相似。即测试线段端点与指定区域边界的位置关系，但对可见性测试还需要比较边与面的深度值。当一线段投影边的端点均位于一个面的投影区域内时，将端点深度值与那些在位置(x, y)的面深度进行比较。如果两个端点均位于该面的后面，则该边是一条隐藏线。如果两个端点均位于该面的前面，则该边相对于该面可见。否则，必须计算交点位置并确定这些交点处的深度值。如果在周边各交点处该边的深度比该面大，那么该边的一部分被该面隐藏，如图9.28(a)所示。另一个可能是一条边在一个边界的交点有较大的深度，而在另一边界的交点深度小于该表面深度（假定面是凸的）。在这种情况下，需要确定该边在何处穿入该面的内部，如图9.28(b)所示。一旦确认了一边的隐藏部分，我们可消除它、将它显示成虚线或以其他区别于可见部分的方法来显示它。

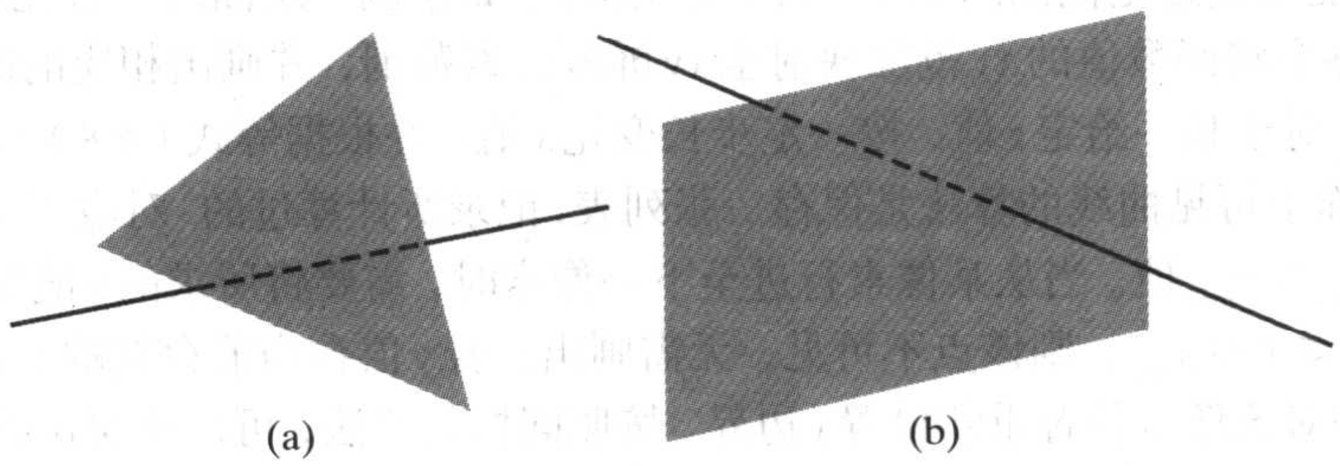


图 9.28 一条线段的隐藏部分（虚线）：(a)在表面后面；(b)穿过表面

一些可见面判别算法可以很容易地移植到线框图的可见性测试。使用后向面判别算法，可以判别出对象的所有后向面并仅显示可见面的边界。利用深度排序算法，可以由后往前地逐个处理表面，利用背景色填充表面内部，利用前景色绘制边界，可将表面填入刷新缓存。这样，隐藏线被前面的表面所消除。区域细分算法可以通过只显示可见面的边框而用于消除隐藏线。扫描线算法可以通过沿扫描线与可见面边界相交处画点的算法来显示可见线。

9.13.2 线框图深度提示算法

另一个显示可见性信息的方法是将场景中对象的亮度作为离观察位置距离的函数而改变。这种**深度提示方法**（depth-cueing method）一般使用下列线性函数：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$f_{\text{depth}}(d) = \frac{d_{\max} - d}{d_{\max} - d_{\min}} \tag{9.9}$$

这里 d 是一个点离观察位置的距离。最小和最大的深度，$d_{\min}$ 和 $d_{\max}$，在具体系统中设定为方便的值。最小和最大的深度也可设定为规范化的值：$d_{\min} = 0.0$ 和 $d_{\max} = 1.0$。每一个像素处理后，其颜色乘上 $f_{\text{depth}}(d)$。因此，较近的点用高强度显示，而最大深度的点的强度等于 0。

深度提示函数可用多种方法实现。在有些图形库中有通用的雾气函数（参见 10.3 节）来将深度提示与雾气效果结合以模拟烟或雾。因此，对象的颜色可用深度提示函数修改并与颜色混合。

9.14　OpenGL 可见性检查函数

使用 OpenGL 基本库提供的函数可以消除场景中的后向面并实现深度缓存可见性测试。另外，还可使用 OpenGL 函数，以构造消除了隐藏线的场景线框图显示并可显示带深度提示的场景。

9.14.1　OpenGL 多边形剔除函数

使用下列函数可消除后向面：

```
glEnable (GL_CULL_FACE);
glCullFace (mode);
```

这里的参数 mode 赋值为 GL_BACK。实际上，这一函数还可以用来消除前向面或将前向面和后向面都消除。例如，当观察位置在大楼内部时，我们只要观察后向面（房间内部）。这种情况下，可将参数设定为 GL_FRONT，或使用 4.14 节讨论的 glFrontFace 函数来改变前向多边形的定义。然后，当观察位置移到大楼的外面时，可以从显示中剔除后向面。在有些应用中，我们可能仅仅希望观察场景中的其他图元，如点集和单独的直线段。因此，要消除场景中所有的多边形面，必须将参数 mode 设定为 OpenGL 符号常量 GL_FRONT_AND_BACK。

默认时，glCullFace 函数中的参数 mode 取值 GL_BACK。因此，如果用 glEnable 函数激活剔除功能但不明确调用 glCullFace，则场景中的后向面将被消除。使用下列函数可关闭剔除功能：

```
glDisable (GL_CULL_FACE);
```

9.14.2　OpenGL 深度缓存函数

在使用 OpenGL 深度缓存可见性检测子程序之前，必须修改 GLUT 初始化功能，使显示模式包含对深度缓存及刷新缓存的请求。例如用下列语句：

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

深度缓存值的初始化使用：

```
glClear (GL_DEPTH_BUFFER_BIT);
```

正常情况下，使用将刷新缓存初始化为背景色的同一函数来对深度缓存进行初始化。但必须在每次显示新的一帧之前清除深度缓存。OpenGL 中，深度缓存规范化成 0.0 到 1.0 的范围，因此默认时前面的初始化将深度缓存值设为最大值 1.0。

OpenGL 深度缓存可见性检测子程序由下列函数激活：

```
glEnable (GL_DEPTH_TEST);
```

关闭深度缓存子程序则使用：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glDisable (GL_DEPTH_TEST);
```

在应用深度缓存可见性检测时也可以使用另外的值作为最大深度，该初始值用下列OpenGL函数选定：

```
glClearDepth (maxDepth);
```

参数maxDepth可设为0到1.0中的任何值。如果要将该初始值装进深度缓存，则必须引用glClear(GL_DEPTH_BUFFER_BIT)函数。否则，深度缓存用默认值1.0初始化。因为该函数对指定的最大深度之后的对象不进行表面颜色计算和其他处理，它在场景中包含许多位于前景对象之后的对象的情况下可用来加速深度缓存子程序。

OpenGL中的投影坐标规范化为-1.0到1.0的范围内，而在近和远裁剪平面之间的深度值进一步规范化为0.0到1.0的范围。值0.0与近裁剪平面（投影平面）对应，而值1.0与远裁剪平面对应。作为一个选择，我们可以用下列函数来调整规范化值：

```
glDepthRange (nearNormDepth, farNormDepth);
```

默认时，nearNormDepth = 0.0且farNormDepth = 1.0。但使用glDepthRange函数可将这两个参数设定为从0.0到1.0范围内的任何值，包括nearNormDepth > farNormDepth的情况。使用glDepthRange函数可将深度缓存测试限制在观察体内的任何区域，甚至将近平面和远平面反过来。

OpenGL中的另一选择是深度缓存子程序中使用的测试条件。我们使用下列函数来指定一个测试条件。

```
glDepthFunc (testCondition);
```

参数testCondition可赋以下列八个符号常量中的任意一个：GL_LESS、GL_GREATER、GL_EQUAL、GL_NOTEQUAL、GL_LEQUAL、GL_GEQUAL、GL_NEVER（没有处理任何点）、GL_ALWAYS（已处理所有点）。这些不同的测试用于各种应用中以减少深度缓存处理的计算。参数testCondition的默认值是GL_LESS，因此小于该深度缓存像素位置当前值的深度值被处理。

我们也可以将深度缓存状态设定为只读或读写状态。这通过下列函数实现：

```
glDepthMask (writeStatus);
```

当writeStatus = GL_TRUE（默认值）时，我们既可从深度缓存读也可向其中写。writeStatus = GL_FALSE时，深度缓存的写模式失效，因而只能从中取值进行测试比较。当我们要使用与不同的前景对象显示同样复杂的背景时这一特点是有用的。在对深度缓存中的背景排序后，我们让写模式失效并处理前景。这可以生成有不同前景对象的一系列帧或为动画序列在不同位置使用同一对象。因此，仅存储背景的深度值。glDepthMask函数的另一应用是显示透明效果（参见10.20节）。在这种情况下，我们只要为可见性测试存储不透明对象的深度，而不是透明表面位置的深度。因此在处理到透明表面时关闭深度缓存的写模式。类似的命令可用来为其他缓存设定写状态（颜色、索引和模板）。

9.14.3 OpenGL线框面可见性方法

在OpenGL中通过请求只生成边，可获得一般图形对象的线框显示。设定多边形模式函数便可做到这一点，例如

```
glPolygonMode (GL_FRONT_AND_BACK,GL_LINE);
```

但这既显示可见的也显示隐藏的边。

要在线框显示中消除隐藏线，可使用4.14节中叙述的深度位移方法。即先用前景颜色指定对象的线框版本，然后使用深度位移和用于内部填充的背景色来指定内部填充版本。深度位移保证了

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

背景色填充不会影响可见边的显示。作为一个例子，下列程序段使用白色前景色和黑色背景色来生成一个对象的线框图显示。

```
glEnable (GL_DEPTH_TEST);
glPolygonMode (GL_FRONT_AND_BACK, GL_LINE);
glColor3f (1.0, 1.0, 1.0);
\*  Invoke the object-description routine.  */

glPolygonMode (GL_FRONT_AND_BACK, GL_FILL);
glEnable (GL_POLYGON_OFFSET_FILL);
glPolygonOffset (1.0, 1.0);
glColor3f (0.0, 0.0, 0.0);
\*  Invoke the object-description routine again.  */

glDisable (GL_POLYGON_OFFSET_FILL);
```

9.14.4 OpenGL 深度提示函数

使用下列函数可按对象离观察位置距离的函数来改变对象的亮度：

```
glEnable (GL_FOG);

glFogi (GL_FOG_MODE, GL_LINEAR);
```

这将等式（9.9）中的线性深度函数及 $d_{min} = 0.0$、$d_{max} = 1.0$ 应用到对象颜色上。但我们可以用下列函数调用来为 d_{min} 和 d_{max} 设定不同的值。

```
glFogf (GL_FOG_START, minDepth);
glFogf (GL_FOG_END, maxDepth);
```

在这两个函数中，参数minDepth和maxDepth赋以浮点值，尽管在将函数后缀改为i时可使用整数值。

另外，我们可以在应用线性深度提示函数后用 glFog 函数设定将与对象颜色混合的雾气颜色。其他的雾气效果也可以进行建模，这些不同的选择将在 10.20 节中讨论。

9.15 小结

后向面检测算法是最简单的可见性测试，它在进一步的可见性测试中快速而有效地完成了许多多边形的消除。对于单个凸多面体，后向面判别可以消除所有隐藏面，但通常情况下，它无法保证能识别全部隐藏面。

深度缓存算法是一个常用的判别场景中可见面的算法。在将其应用于标准的图形对象时，该过程是高效的，但它要求额外的存储器。该算法需要两个缓存，一个保存像素的颜色，另一个则保存每个像素所对应的可见面的深度。该算法采用快速增量算法来扫描场景中的每一个面以计算表面深度，一旦所有表面处理结束，也就更新完了两个缓存。A 缓存是深度缓存算法的一个改进，该算法为显示反走样和透明表面提供了手段。

本章还介绍了其他一些可见面判别算法。扫描线算法为每一行同时处理所有面。使用深度排序（画家算法）算法时，对象按其离观察位置的距离画在刷新缓存中。判定场景可见部分的细分算法包括BSP树方法、区域细分方法和八叉树算法。可见面还可以用光线投射算法来检测，它将从像素平面的线条投影到场景中，以确定沿投影线的对象相交位置。光线投射方法是光线跟踪方法的一个集成部分，光线跟踪方法则是提供全局照明效果的一种方法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

可见性判别算法也可用于显示三维线框图。对于曲面，我们可以显示其层位线，而对于线框方式的多面体，则在场景中寻找从观察平面可以看见的线段。

通过建立自己的子程序，可在应用程序中实现任何一种可见性检测方法，但图形库一般仅提供后向面消除和深度缓存算法的函数。在高级计算机图形系统中，深度缓存算法用硬件实现。

OpenGL 核心库中提供用于多边形剔除和深度缓存可见性判定的函数。使用多边形剔除函数，我们可以消除标准图形对象的后向面、前向面或两者。使用深度缓存函数可设定深度测试的范围和执行的深度测试类型。用OpenGL多边形模式和多边形位移操作可获得线框图。还可以用深度提示效果生成 OpenGL 场景。在表 9.1 中，总结了第 4 章末尾讨论的多边形模式函数和其他相关操作。

表 9.1 OpenGL 可见性检测函数小结

函数	描述
glCullFace	在用glEnable (GL_CULL_FACE)激活时，为剔除操作指定多边形前向和后向面
glutInitDisplayMode	用变量 GLUT_DEPTH 指定深度缓存操作
glClear (GL_DEPTH_BUFFER_BIT)	用默认值 1.0 或由 glClearDepth 函数指定的值来初始化深度缓存
glClearDepth	指定深度缓存的初始化值
glEnable (GL_DEPTH_TEST)	激活深度测试操作
glDepthRange	指定规范化深度值的范围
glDepthFunc	指定深度测试条件
glDepthMask	指定深度缓存的写状态
glPolygonOffset	当应用背景填充颜色时指定线框显示中消除隐藏线的位移
glFog	指定线性深度提示操作和深度提示计算中的最小和最大深度值

参考文献

有关可见性判别算法的另外来源可参见Elber and Cohen(1990)、Franklin and Kankanhalli(1990)、Segal(1990)和 Naylor, Amanatides, and Thibault(1990)。A 缓存方法在 Cook, Carpenter, and Catmull (1987)、Haeberli and Akeley(1990)和 Shilling and Strasser(1993)中给出。层位线方法的总结在 Earnshaw(1985)中给出。

可见性测试的各种程序设计技术可在Glassner(1990)、Arvo(1991)、Kirk(1992)、Heckbert(1994)和 Paeth(1995)中找到。而 Woo, Neider, Davis, and Shreiner(1999)给出了对 OpenGL 可见性检测函数的其他讨论。完整的 OpenGL 核心库和 GLU 功能清单在 Shreiner(2000)中给出。

练习题

9.1 开发一个基于后向面判别算法的程序，用于任何一个输入的具有不同颜色外表面的凸多面体的可见面判别。假设多面体在右手观察系统中定义且观察方向由用户输入来指定。

9.2 实现上一练习题的一个程序，使用正交平行投影对一个凸多面体进行观察。假设整个对象都在观察平面之前，且将其投影到屏幕视区中显示。

9.3 实现一个练习题9.1的程序，利用透视投影对一个输入凸多面体进行观察。假设整个对象都在观察平面之前。

9.4 编写实现一个凸多面体的动画的程序，对象绕一个平行于观察平面的轴加速旋转。假设整个对象位于观察平面之前，使用正平行投影将视野投影至观察平面。

9.5 实现深度缓存算法以显示任何输入多面体的可见面。用于深度缓存的数组可设定为方便的尺寸，如 500 × 500。并思考如何从对象的定义得出深度缓存所需的存储量。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

9.6 修改上一练习题的程序，可显示包含任意多个多面体的场景的可见面。请采用有效算法来完成场景中不同对象的存储和处理。

9.7 修改上一练习题的程序，利用 A 缓存算法，实现一个同时包含透明面和非透明面的场景的显示。

9.8 扩展上一练习题的程序，使其包含反走样处理。

9.9 开发一个程序，利用扫描线算法实现一个给定多面体的可见面显示。请用多边形表来保存对象的定义，并利用连贯性算法来计算扫描线上及扫描线间点的值。

9.10 编写实现一个扫描线的程序，以完成包含多个多面体的场景的显示。请用多边形表和边表来保存对象的定义，并利用连贯性算法来计算扫描线上及扫描线间点的值。

9.11 请利用画家算法，编写实现凸多面体的可见面显示的程序。即将面片按深度顺序由远及近地画到屏幕上。

9.12 请利用深度排序算法编程，实现任意给定的由平面围成的对象的可见面显示。

9.13 编写实现深度排序算法的程序，完成一个包含几个多面体的场景的可见面显示。

9.14 请编写一个程序，利用 BSP 树算法实现凸多面体的可见面显示。

9.15 举例说明在何种情况下，区域细分算法中条件(3)讨论的两种算法将无法正确判别出遮挡其他表面的包围表面。

9.16 编写一个算法，测试一个给定平面表面与某个矩形区域之间的关系，决定该表面是包围型、重叠型、内含型还是分离型。

9.17 编写一个算法，利用区域细分算法来确定四叉树各节点的值，以生成可见面的四叉树表示。

9.18 编写一个算法，将某对象的给定四叉树表示读入帧缓存中进行显示。

9.19 编写一个程序，显示一个以八叉树表示的对象的可见面。

9.20 推导一个算法，显示一个使用光线投射算法的球体。

9.21 讨论如何将反走样处理加入各种隐面消除算法中。

9.22 编写一个程序来实现一个曲面函数 $f(x, y)$ 的层位线绘制。

9.23 编写一个算法，实现在场景中将每条线段与每个表面进行比较，以判别可见线段。

9.24 讨论如何利用本章中的几个可见面判别算法来实现线框显示。

9.25 编写一个程序，实现一个多面体的线框显示，用虚线画出隐藏线。

9.26 编写一个程序，使用 OpenGL 多边形剔除函数来消除选择的面并显示一个多面体，每一多边形面给以不同颜色，且每一面可由用户选择消除。同样，观察位置和其他观察参数也由输入值指定。

9.27 修改上一练习题的程序，使用深度缓存函数而不是多边形剔除函数从任何位置观察该多面体。

9.28 编写一个程序，使深度范围和深度测试条件可作为用户输入来指定。

9.29 用 9.14 节中讨论的 glPolygonMode 和 glPolygonOffset 函数创建一个多面体的线框图显示。

9.30 修改上一练习题的程序，使用深度提示函数 glFogi 显示该多面体。

9.31 修改上一练习题的程序，显示深度上分布的多个多面体。深度提示范围由用户输入来设定。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第10章　光照模型与面绘制算法

对场景的对象进行透视投影，然后在可见面上产生自然光照效果，可以实现场景的真实感显示。**光照明模型**（illumination model），也称**光照模型**（lighting model）或明暗模型（shading model），主要用于对象表面某光照位置的颜色计算。**表面绘制算法**（surface-rendering method）是使用光照模型为对象的所有投影位置确定像素颜色。该光照模型可应用于每一投影位置，或者先对表面上少数点使用光照模型计算然后再进行颜色插值。扫描线、像空间算法一般使用插值模式，而光线跟踪算法则在每一像素点处使用照明模型。有时，面绘制过程也称为明暗方法，即使用明暗模型来计算表面颜色，但这会导致这两个概念之间的某种混淆。为了避免由于使用类似术语造成的误解，我们将表面上单个点的光强度计算模型称为光照明模型或光照模型，而将应用一个光照模型获得所有投影的表面位置的像素颜色的过程称为表面绘制。

计算机图形学中真实感成像包括两部分内容：表面特性的精确表示和场景中光照效果的物理描述。表面光照效果包括光的反射、透明性、表面纹理和阴影。

为可见对象的颜色和光照效果建立模型是一个非常复杂的过程，它同时包含物理学和心理学的内容。从根本上来说，可以用一个表示场景对象表面电磁能量相互作用的模型来描述光照效果。一旦光线到达了我们的眼睛，它刺激感觉器官使我们“看见”屏幕上的对象。物理上的光照模型包含许多因素，如材料特性、对象相对于光源及其他对象的位置以及场景中的光源属性；对象可以是不透明的，也可以有或多或少的透明度；其表面可以是光亮的，也可以是阴暗的，还可以带有各种各样的表面纹理。不同形状、颜色、位置的光源可以为一个场景带来不同的光照效果。一旦确定出对象表面的光学属性参数，场景中表面的相对位置关系，以及光源的颜色、位置和观察平面的位置、朝向等信息，就可根据光照模型计算出对象表面上某点在观察方向上所投射的光强度值。

计算机图形学中的光照模型可以由描述对象表面光照效果的物理公式推导出来。为了减少光强度计算，大多数软件包采用简化的光照计算经验模型。一些较精确的计算模型，如辐射度算法，则需考虑场景中光源与对象表面间辐射能量的传递来计算光强度。在下面的内容中，我们首先介绍常用于一些图形软件包中的基本光照模型，然后讨论一些较精确但较复杂的确定光照表面外貌的方法。最后，我们提出几种面绘制算法，将光照模型用于获得自然场景的有效显示。

10.1　光源

任一发出辐射能量的对象称为一个**光源**（light source），它对场景中其他对象的光照效果有贡献。我们可以使用各种形状和特征来建立光源的模型，而多数发光体仅作为场景中的光照源。然而在有些应用中，我们可能希望建立既是光源又是光反射体的对象。例如，在一个塑料球内放置一个灯泡，这样在球表面上既发光也反射光。我们也可以把该球面看成一个包围光源的半透明表面。但对于大型萤光板一样的对象来说，比较适合于作为发光体和反射体的混合。

一个光源可定义许多特性。我们可指定其位置、发射光颜色、发射方向及它的形状。如果该光源也是一个光反射表面，还需给出它的反射特性。另外，我们可以建立一个在不同方向发射不同颜色光的光源。例如，可定义一个在一侧发射红光而在另一侧发射绿光的光源。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在许多应用特别是实时图形显示中，常使用简单的光源模型以避免过多的计算。为每一 RGB 颜色分量使用单个值来赋以光发射特性，可描述为颜色分量的总量或“强度”。颜色参数和光源模型将在第 12 章中详细讨论。

10.1.1　点光源

使用三个 RGB 分量指定的单个颜色的**点光源**（point light source）是发光体的最简单的模型。一个场景的点光源通过给出其位置和发射光颜色来定义。如图 10.1 所示，光线由光源向四周发散。这种光源模型是对场景中比对象小得多的光源的合理逼近。离场景不是太近的大光源也可以利用点光源模型来模拟。光照模型中点光源的位置用来确定场景中哪些对象由该光源照明，并计算到选定对象表面位置的光线方向。

10.1.2　无穷远光源

离场景非常远的大型光源（如太阳）也可用一个点发光体逼近，但在方向效果上有所不同。与场景中照明各方向对象的光源相对比，远距离光源仅在一个方向照明场景。从远距离光源到场景中任一位置的光线路径接近不变，如图 10.2 所示。

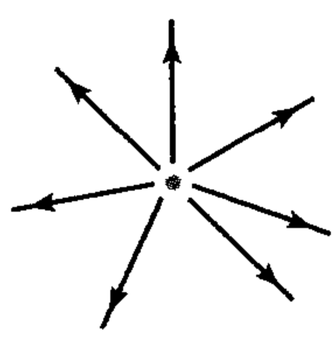


图 10.1　从点光源发出的光线路径

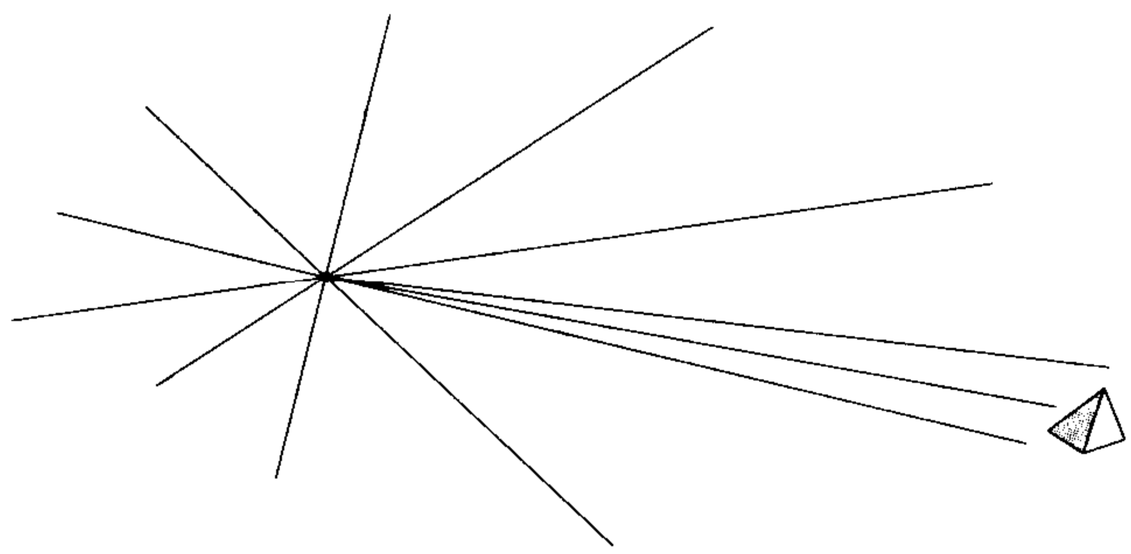


图 10.2　从无穷远光源发出的光线以几何平行的光路径照明一个对象

我们可以使用一个颜色值和从该光源发出的一个固定光线方向来模拟一个无穷远距离光源。在光照计算中，仅需要一个发射方向的向量及光源颜色而不需要光源位置。

10.1.3　辐射强度衰减

辐射光线从一点光源出发并在空间中传播，离光源距离为 d_l 时它的振幅将按因子 $1/d_l^2$ 进行衰减。这表明接近光源的表面将得到较高的入射光强度，而较远的表面则强度较小。因此，如果要得到真实感的光照效果，在光照明模型中必须考虑光强度衰减。否则对所有表面赋以同样的光强度，而不考虑它们离光源的距离，必定会导致非期望的显示效果。例如，当两个具有相同光学参数的平行表面互相遮挡时，将无法区分这两个平行表面。因此，忽略它们离光源的相对距离，这两个表面看起来如同一个表面。

然而，如果采用因子 $1/d_l^2$ 进行光强度衰减，那么简单的点光源照明并不总能产生真实感的图形。对于接近光源的对象，$1/d_l^2$ 会产生过大的强度变化，而 d_l 很大时变化又太小，这是因为实际的光源并不是无穷小的点，而用点发光体照明一个场景是真实光照效果的简单近似。要用点光源生成更具真实感的显示，可使用包含一个线性项的 d_l 的二次函数的倒数来衰减光强度：

$$f_{\text{radatten}}(d_l) = \frac{1}{a_0 + a_1 d_l + a_2 d_l^2} \tag{10.1}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

用户可以调整系数a_0、a_1、a_2的值，以得到场景中不同的光照效果。例如，当d_l非常小时可赋予常数项a_0一个大的值来防止$f_{\text{radatten}}(d_l)$值变得太大。另外，在图形软件包中经常提供多组不同的衰减系数值赋给场景中的每一个点光源。

对无穷远点光源不能使用强度衰减来计算（10.1）式，因为离光源的距离是不确定的。同样，场景中各点离很远的光源的距离几乎都相等。要同时考虑远距离和局部光源，可将强度衰减函数表达成

$$f_{l,\text{radatten}} = \begin{cases} 1.0, & \text{如果光源在无穷远处} \\ \dfrac{1}{a_0 + a_1 d_l + a_2 d_l^2}, & \text{如果光源是局部光源} \end{cases} \tag{10.2}$$

10.1.4 方向光源和投射效果

一个局部光源稍加修改就可产生方向光束或投影光束。如果一个对象位于一个光源的方向范围之外，则它得不到该光源的光照。建立方向光源的一个办法是除了位置和颜色外还要确定一个向量方向和从该向量方向开始的角度范围θ_l。这定义了以光源向量方向为轴的一个圆锥（参见图10.3）。使用多个方向向量且每一方向都不同的发射颜色，可以建立一个多色点光源模型。

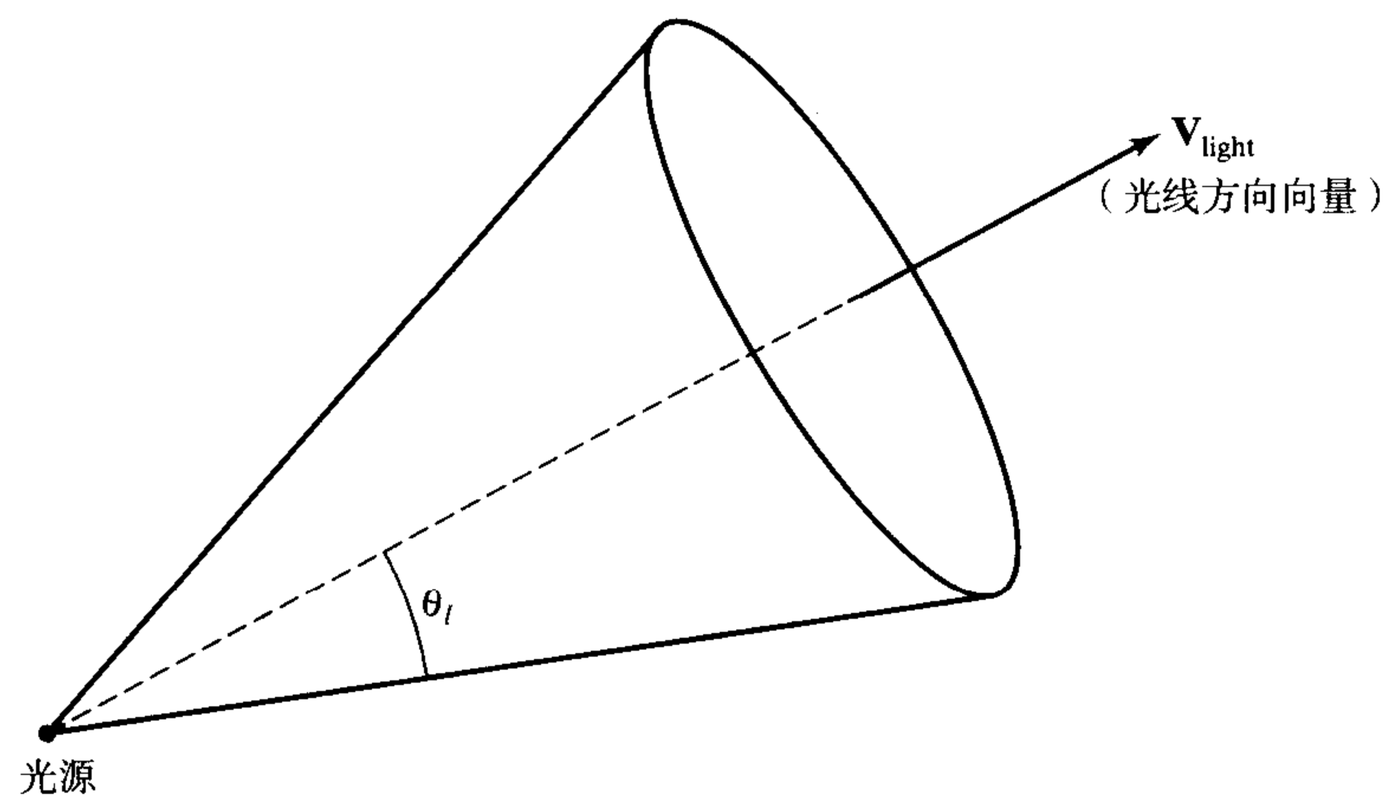


图 10.3 一个方向点光源。单位光方向向量定义光锥的轴，而角θ_l定义圆锥的角度范围

记$\mathbf{V}_{\text{light}}$为光源方向的单位向量且记$\mathbf{V}_{\text{obj}}$为从光源位置到一个对象位置的方向上的单位向量。则

$$\mathbf{V}_{\text{obj}} \cdot \mathbf{V}_{\text{light}} = \cos\alpha \tag{10.3}$$

这里α是从光方向向量到对象的角距离。如果把任一光锥角度范围限制为$0° \leqslant \theta_l \leqslant 90°$，则当$\cos\alpha \geqslant \cos\theta_l$时对象位于投影光内，如图 10.4 所示。但如果$\mathbf{V}_{\text{obj}} \cdot \mathbf{V}_{\text{light}} < \cos\theta_l$，则对象位于光锥体之外。

10.1.5 角强度衰减

对于一个方向光源，可按照从点光源位置出发的光强度角计算衰减。即可模拟一个光锥，沿其圆锥轴光强最大，离开圆锥轴时强度逐渐减弱。常用的方向光源角强度衰减函数是

$$f_{\text{angatten}}(\phi) = \cos^{a_l}\phi, \qquad 0° \leqslant \phi \leqslant \theta \tag{10.4}$$

这里衰减指数a_l赋以某个正值而角ϕ从圆锥轴开始度量。在圆锥轴上，$\phi = 0°$而$f_{\text{angatten}}(\phi) = 1.0$。衰减指数$a_l$的值越大，在角值$\phi > 0°$时角强度衰减函数的值越小。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

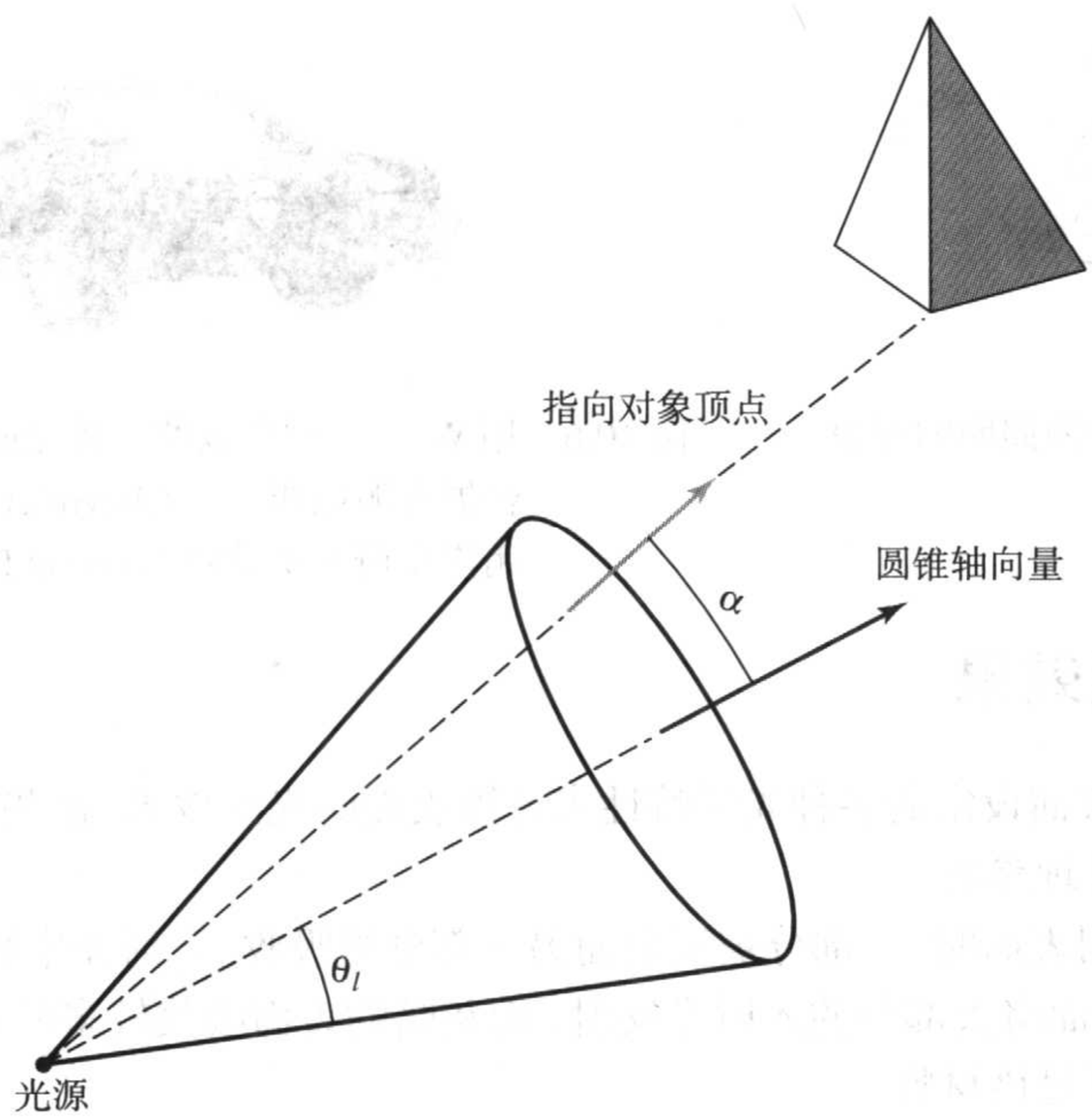


图 10.4　一个由方向点光源照明的对象

在实现光强度函数时有几个特殊情况要考虑。如果光源不是方向光源（即不是投影光源）则没有角强度衰减。同样，如果一对象位于投影光锥体外的任何地方，则它得不到该光源的光照。要获得沿光源位置到场景中一个表面位置间连线的角强度衰减因子，可使用等式（10.3）中的点积计算方法来计算离圆锥轴的方向角的余弦。指定 $\mathbf{V}_{\text{light}}$ 为光源方向（沿该圆锥轴）的单位向量且 $\mathbf{V}_{\text{obj}}$ 为光源到一对象位置的方向的单位向量。使用这两个单位向量并假定 $0° < \theta_l \leqslant 90°$，可给出角强度衰减的一般公式：

$$f_{l,\text{angatten}} = \begin{cases} 1.0, & \text{如果光源不是一个投影光源} \\ 0.0, & \mathbf{V}_{\text{obj}} \cdot \mathbf{V}_{\text{light}} = \cos\alpha < \cos\theta_l \\ & \text{（对象位于投射圆锥之外）} \\ (\mathbf{V}_{\text{obj}} \cdot \mathbf{V}_{\text{light}})^{a_l}, & \text{其他} \end{cases} \tag{10.5}$$

10.1.6　扩展光源和 Warn 模型

当需要在接近场景对象的位置引入一个如图 10.5 所示的霓红灯的大光源时，可以使用一个发光表面来逼近。一种做法是用方向点光源的网格来模拟。可按在其后的对象得不到光照的效果来对光源建模。还可以引入另外的控制来限制接近光源边的发射光方向。

Warn 模型（Warn model）提供了一种模拟立体光照效果的方法。它通过使用一组多参数的点发光器来模拟挡光板、快门和照相用聚光灯控制。聚光灯用前面讨论的圆锥体实现，而快门和挡光板提供了其他的方向控制。例如，为 x、y 和 z 方向中的每一个建立两个快门来进一步限制发射光线。某些图形软件包实现了对这种光源的模拟，图 10.6 给出了可用 Warn 模型获得的光照效果。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

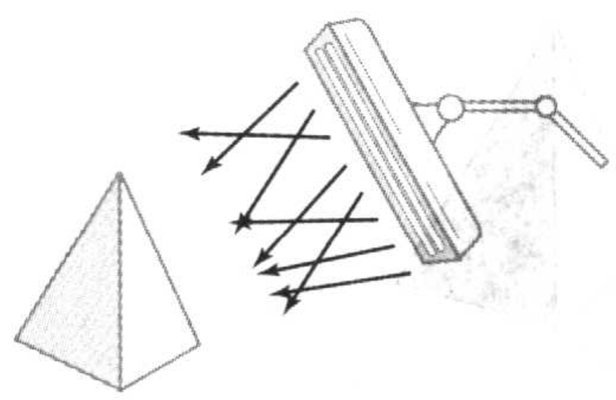


图 10.5 一个用靠近的光源照明的对象

图 10.6 用 Warn 模型生成的立体光照效果，使用了 5 个扩展光源照明一个 Chevrolet Camaro 汽车（由通用汽车研究实验室的 David R. Warn 提供）

10.2 表面光照效果

光照模型使用为表面设定的各种光学特性来计算表面的光照效果。这些特性包括透明度、颜色反射系数及各种表面纹理参数。

当光入射到不透明表面时，一部分被反射而另一部分被吸收。入射光中被表面反射的总量依赖于材料类型。光亮的表面将大部分的入射光反射，而灰暗的表面吸收较多的入射光。对于透明的表面，入射光的一部分穿过该材料。

粗糙或颗粒状表面会将反射光向各方向发散出去。这种发散称为**漫反射**（diffuse reflection）。非常粗糙、不光滑的表面主要产生漫反射，因而从任何角度来看表面的亮度均相同。图10.7示例了从一表面发散出的漫反射光。当使用由各种颜色混合而成的白光照明时，对象的漫反射光的颜色称为该对象的颜色。例如，蓝色对象将白色光中的蓝色成分反射而将其他成分吸收。如果蓝色对象在红色光下观察，则它表现为黑色，因为所有入射光均被吸收。

除了发散的漫射光之外，有些反射光会集中成醒目的或明亮的一个点，这称为**镜面反射**（specular reflection）。这种醒目效果在光亮表面发生得比灰暗表面多。在从特定角度观看照亮的光亮表面（如抛光的金属、苹果或人的额头）时，可以看到镜面反射。图 10.8 给出了镜面反射的示意。

场景中的**背景光**（background light）或**环境光**（ambient light）是光照模型中必须考虑的另一因素。由于从得到照明的邻近对象发出的反射光，使并不直接暴露在光源下的表面仍是可见的。因此，场景中的环境光是该场景各个表面的反射光生成的光照效果。图10.9给出了这一背景光的光照效果。一个表面上反射光的全体是光源光的反射和其他被照明对象反射光的反射两者之和。

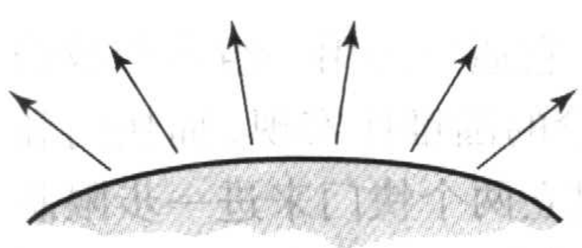


图 10.7 一个表面的漫反射

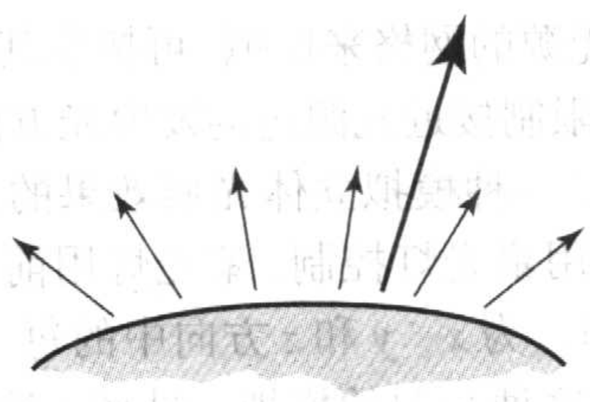


图 10.8 加在漫反射向量中的镜面反射

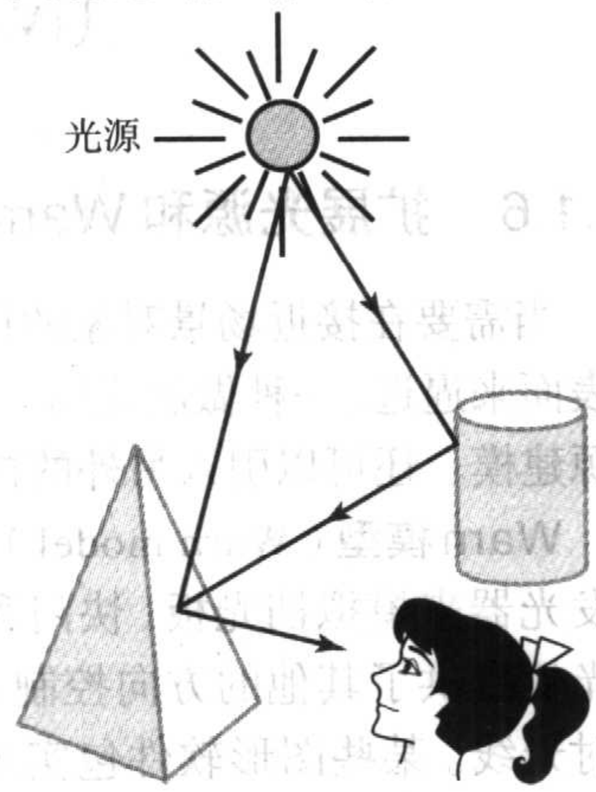


图 10.9 表面光照效果由光源光和其他表面反射光混合生成

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.3　基本光照模型

精确的光照模型按照入射光能量与对象的材料组成之间的交互结果进行计算。为了简化表面光照计算，可使用在前面部分中讨论的生成光照效果的物理过程的近似表示。本节叙述的经验模型生成比较好的结果，并且在多数图形软件包中得到了实现。

基本光照模型中的发光体一般限于点光源。然而，许多图形软件包提供处理方向光源（投射光源）和扩展光源的函数。

10.3.1　环境光

在基本光照模型中，可通过设定场景一般亮度级来引入背景光。这生成了对所有对象都相同一个环境光，且近似地给出了各个照明表面的全局漫反射。

假设我们正在叙述单色光的光照效果（例如灰色阴影），用强度参数 I_a 表示环境光的层次。这样，场景中的每个表面都得到这个背景光的照明。由环境光照明生成的反射是漫反射的一个简单情况，它依赖于观察方向和表面空间朝向的关系。然而，反射出的入射环境光总量依赖于决定入射光能量中多少被反射出去及多少被吸收的表面光学特性。

10.3.2　漫反射

我们可以在建立表面的漫反射模型时假设入射光在各个方向以相同强度发散而与观察位置无关。这样的表面称为**理想漫反射体**（ideal diffuse reflector）。因为从表面上任一点反射出的辐射光能量用**朗伯余弦定律**（Lambert’s cosine law）计算，因此也称为**朗伯反射体**（Lambertian reflector）。该定律表明：在与对象表面法向量夹角为 ϕ_N 的方向上，每个面积为 dA 的平面单位所发散的光线与 $\cos\phi_N$ 成正比（参见图 10.10）。该方向的光强度可用单位时间辐射能总量除以表面积在辐射方向的投影来计算：

$$
\begin{aligned}
\text{强度} &= \frac{\text{单位时间辐射能}}{\text{投影面积}} \\
&\alpha \frac{\cos\phi_N}{dA\cos\phi_N} \\
&= \text{常数}
\end{aligned}
\tag{10.6}
$$

这样，对于朗伯反射，光强度在所有观察方向都相同。

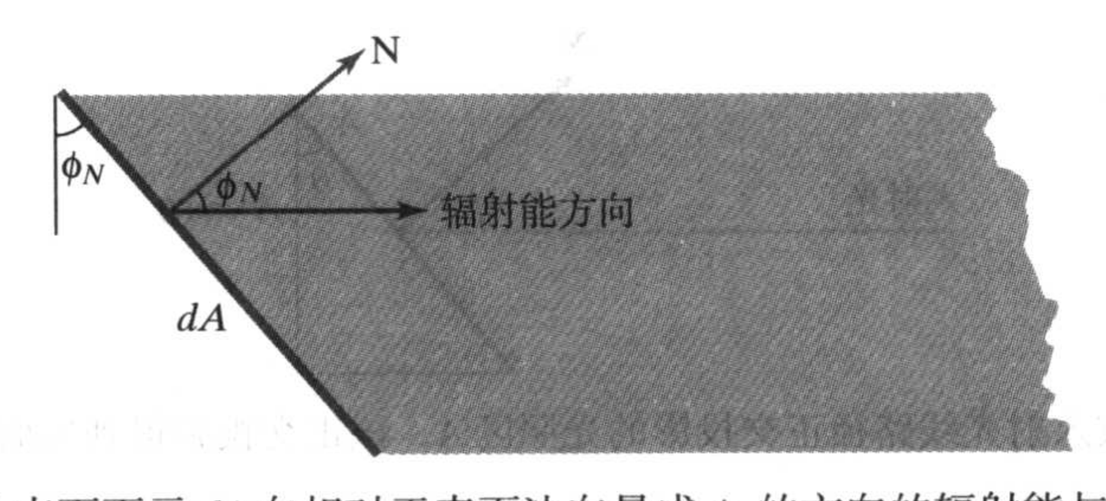


图 10.10　从表面面元 dA 向相对于表面法向量成 ϕ_N 的方向的辐射能与 $\cos\phi_N$ 成正比

假设每一表面都按理想漫反射体（朗伯反射体）对待，则可为确定将要按漫反射发散的入射光部分的每一个表面设定一个参数 k_d。该参数称为**漫反射系数**（diffuse-reflection coefficient）或**漫反射性**（diffuse reflectivity）。这样，任何方向的漫反射是一个常数，它等于入射光强度乘以漫反射系数。对于一个单色光源，按我们所希望的表面反射特性对参数 k_d 赋以 0.0 到 1.0 之间的一个常数值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如果希望一个强反射表面，则将参数k_d的值设成接近1.0。这会生成接近于入射光的反射光强度从而使表面很亮。如果要模拟一个能吸收大部分入射光的表面，则将反射性设定为接近0.0。

对于背景光效果，可假定每一表面都使用对场景设定的环境光I_a来照明，因此，环境光对一表面上各点的漫反射贡献可简单地表示为

$$I_{\text{ambdiff}} = k_d I_a \tag{10.7}$$

环境光是独立的，但它会对表面产生不期望的平面阴影（参见图10.23(b)），因此场景很少仅用环境光绘制。场景中至少要有一个点光源，该点光源通常位于观察位置。

当强度为I_l的光源照明一个表面时，从该光源来的入射光总量依赖于表面与光源的相对方向。与入射光方向接近垂直的表面和一个与入射光方向成斜角的表面相比，从光源接收较多的光。这一光照效果可以用一张白纸或平滑卡片平行放在有阳光的窗口来获得。在将该纸片慢慢转离窗口方向时，表面的亮度逐渐变小。图10.11给出了这一效果，来自远距离光源（平行入射光线）的入射光线束落在两个面积相同但与光线方向关系不同的平表面上。

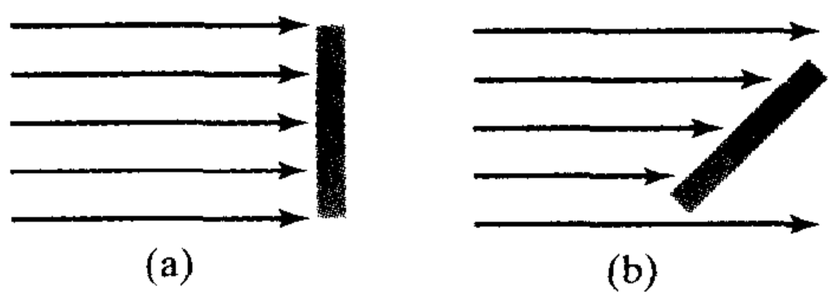


图10.11 与入射光方向垂直的一个表面(a)比同面积但与入射光方向有斜角度的表面(b)得到较多的光照

从图10.11中，我们看到与表面相交的光线数量与该表面投影到入射光方向的面积成正比。如果把入射光方向与表面法线方向之间的**入射角**（angle of incidence）记做θ（参见图10.12），则垂直于光线方向的面片投影面积与$\cos\theta$成正比。因此，一个强度为I_l的光源的入射光总量为

$$I_{l,\text{incident}} = I_l \cos\theta \tag{10.8}$$

使用等式（10.8），可用下式计算强度为I_l的光源的漫反射：

$$\begin{aligned} I_{l,\text{diff}} &= k_d I_{l,\text{incident}} \\ &= k_d I_l \cos\theta \end{aligned} \tag{10.9}$$

当光源的入射光与特定位置的表面垂直时，$\theta = 90°$而$I_{l,\text{diff}} = k_d I_l$。随着入射角的增加，光源的照明减少。因此，仅当入射角在0°~90°之间时（$\cos\theta$在0.0到1.0之间），点光源才照亮面片。如果$\cos\theta < 0.0$，则该点光源位于面片后面。

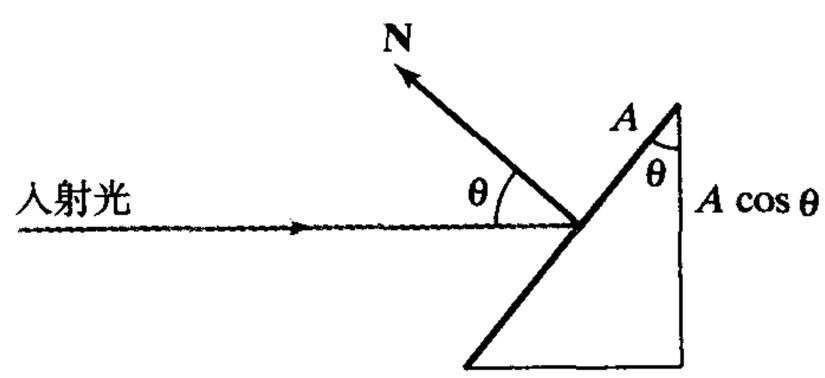


图10.12 按入射光线路径正交投影的光照区A，该正交投影得到的面积为$A\cos\theta$

在表面上任一点处，记单位法向量为**N**及指向点光源的单位方向向量为**L**，如图10.13所示。然后，$\cos\theta = \mathbf{N} \cdot \mathbf{L}$，而一个点光源在表面照明的漫反射公式可表示为

$$I_{l,\text{diff}} = \begin{cases} k_d I_l (\mathbf{N} \cdot \mathbf{L}), & \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \mathbf{N} \cdot \mathbf{L} \leqslant 0 \end{cases} \tag{10.10}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

指向附近点光源的单位方向向量 **L** 使用表面上的位置和光源位置来计算：

$$\mathbf{L} = \frac{\mathbf{P}_{\text{source}} - \mathbf{P}_{\text{surf}}}{|\mathbf{P}_{\text{source}} - \mathbf{P}_{\text{surf}}|} \tag{10.11}$$

然而，“无穷远”的光源没有位置，仅有一个传播方向。此时，使用所赋予的光源发射方向的逆向作为向量 **L** 的方向。

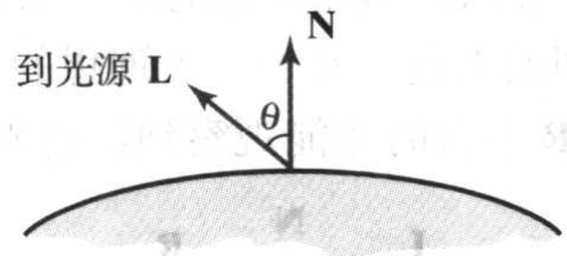


图 10.13　在单位光源方向向量 **L** 和表面位置单位法向量 **N** 之间的入射光角度 θ

图 10.14 给出了等式（10.10）在球面上的应用，其中参数 k_d 在 0 到 1 的范围内选定。$k_d = 0$ 时，没有反射光且对象表面表现为黑色。k_d 的值增大时漫反射强度随之增加，生成浅灰色的明暗效果。该表面对应的每个投影像素位置用漫反射公式计算出的结果来赋值。该图中的表面绘制表示了在没有其他光作用下单个点光源的光照效果。这符合在一个完全黑暗的房间里点亮一个笔式小手电筒时所期望看到的对象效果。然而对一般的场景，我们希望除了点光源所生成的光照效果外还有环境光来形成表面反射。

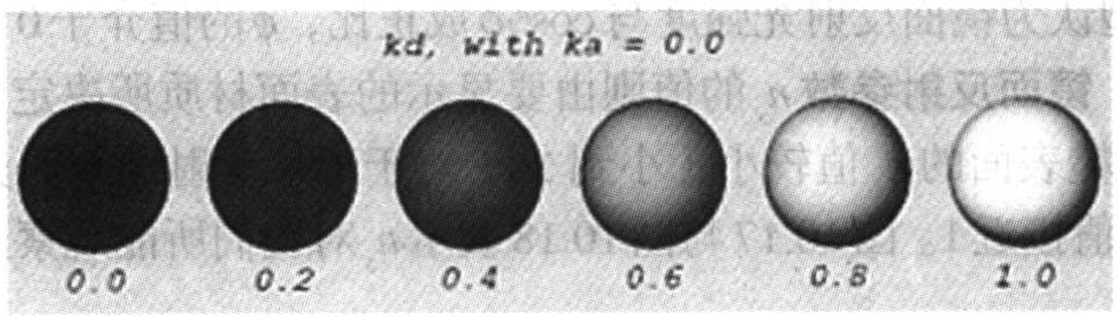


图 10.14　漫反射系数 k_d 介于 0 和 1 之间时球面在单个点光源照明下产生的漫反射

组合环境光和点光源的强度计算，可以得到在一个表面位置的全部漫反射表达。另外，许多图形软件包引入为每一表面指定的**环境光反射系数** k_a 来修改环境光强度 I_a。这使我们能够简单地利用该参数来调节场景中的光照效果。使用参数 k_a，可将单个点光源的全部漫反射公式表示为

$$I_{\text{diff}} = \begin{cases} k_a I_a + k_d I_l (\mathbf{N} \cdot \mathbf{L}), & \mathbf{N} \cdot \mathbf{L} > 0 \\ k_a I_a, & \mathbf{N} \cdot \mathbf{L} \leqslant 0 \end{cases} \tag{10.12}$$

其中，k_a 和 k_d 都依赖于表面材质，对单色光照效果而言其值介于 0 和 1.0 之间，图 10.15 给出了 k_a 和 k_d 介于 0 和 1.0 之间对使用公式计算表面强度的球面显示。

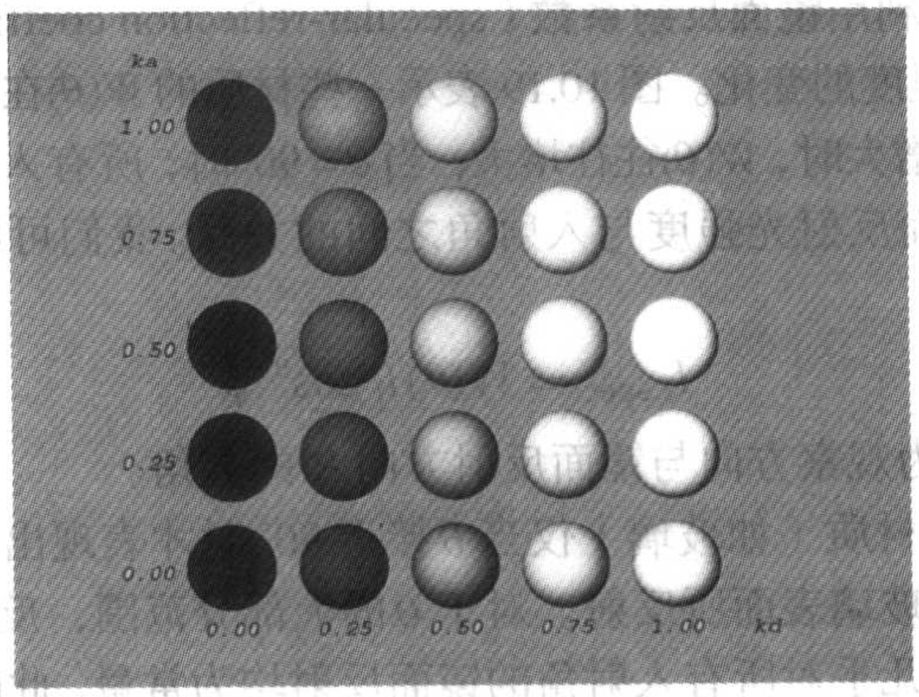


图 10.15　k_a 与 k_d 值介于 0.001 到 1.0 之间的球面在环境光与一个点光源照射下产生的漫反射

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.3.3 镜面反射和 Phong 模型

在光滑表面上看到的高光或镜面反射是由接近**镜面反射角**（specular-reflection angle）的一个会聚区域内入射光的全部或绝大部分成为反射光所导致。图 10.16 给出了照明表面一个位置的镜面反射方向。镜面反射角等于入射角，它们位于表面的单位法向量 **N** 的两侧。在图中我们用 **R** 表示一个理想镜面反射方向的单位向量，**L** 表示指向点光源的单位向量，**V** 为指向视点的单位向量，角度 ϕ 是观察方向与镜面反射方向 **R** 之间的夹角。对于一个理想的反射体（镜子），入射光仅在镜面反射方向有反射现象，此时仅当 **V** 与 **R** 重合时才能观察到反射光线（即 $\phi=0$）。

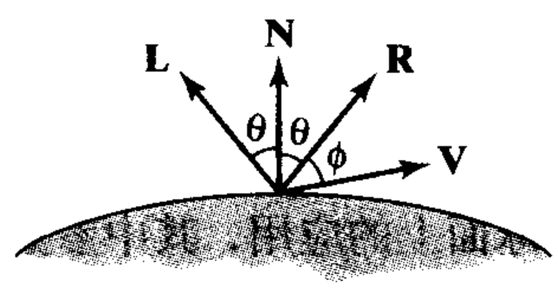


图 10.16 镜面反射角等于入射角 θ

非理想反射体系统的镜面反射方向分布在向量 **R** 周围的有限范围内。较光滑表面的镜面反射范围较小，而粗糙的对象表面则有较大的镜面反射范围。Phong Bui Tuong 曾提出一个计算镜面范围的经验公式，称为 **Phong 镜面反射模型**（Phong specular-reflection model），或简称为 Phong 模型（Phong model）。该模型认为镜面反射光强度与 $\cos^{n_s}\phi$ 成正比，ϕ 的值介于 0° 到 90° 之间，因而 $\cos\phi$ 的值介于 0 与 1.0 之间。**镜面反射参数** n_s 的值则由要显示的表面材质所决定。光滑表面的 n_s 值较大（如 100 或更大），而粗糙表面的 n_s 值较小（小到 1）。对于理想反射器 n_s 是无限的。而粗糙对象表面，如粉笔或煤渣的 n_s 值接近 1。图 10.17 与图 10.18 表示 n_s 对我们所能观察到的镜面反射角度范围的影响。

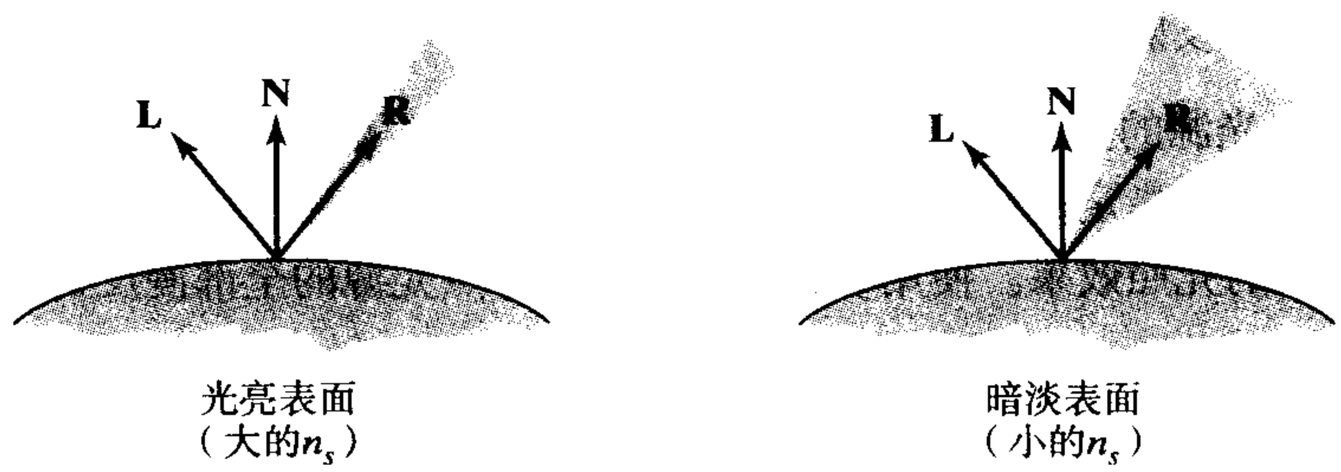


图 10.17 用参数 n_s 模拟镜面反射（阴影区）

镜面反射的光强度主要由对象表面材质属性、光线入射角以及一些其他因素（如极性、入射光线的颜色等）所决定。可以利用**镜面反射系数**（specular-reflection coefficient）$W(\theta)$，为每一个表面近似表示单色镜面反射光强度的变化。图 10.19 表示一些材质的 $W(\theta)$ 在 $\theta=0°$ 与 $\theta=90°$ 之间的大致变化。一般而言，当入射角增大时，$W(\theta)$ 往往增大；当 $\theta=90°$ 时，所有入射光均被反射（$W(\theta)=1$）。Fresnel 反射定律描述了镜面反射光强度与入射角之间的关系。我们可以将 Phong 镜面反射模型表示为

$$I_{l,\text{spec}} = W(\theta) I_l \cos^{n_s}\phi \tag{10.13}$$

其中，I_l 为光源的强度，ϕ 为观察方向与镜面反射方向 **R** 的夹角。

如图 10.19 所示，透明材质（如玻璃）仅当 θ 接近 90° 时才表现出明显的镜面反射。当 $\theta=0°$ 时，大约 4% 的入射光会被玻璃表面所反射。对于 θ 的大部分范围，反射强度小于入射光的 10%，而对于许多不透明的材质，几乎对所有入射角的镜面反射均为常量。此时，可以用一个恒定的镜面反射系数 k_s 来取代 $W(\theta)$。k_s 可以简单地设置为 0 与 1.0 之间的某个值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

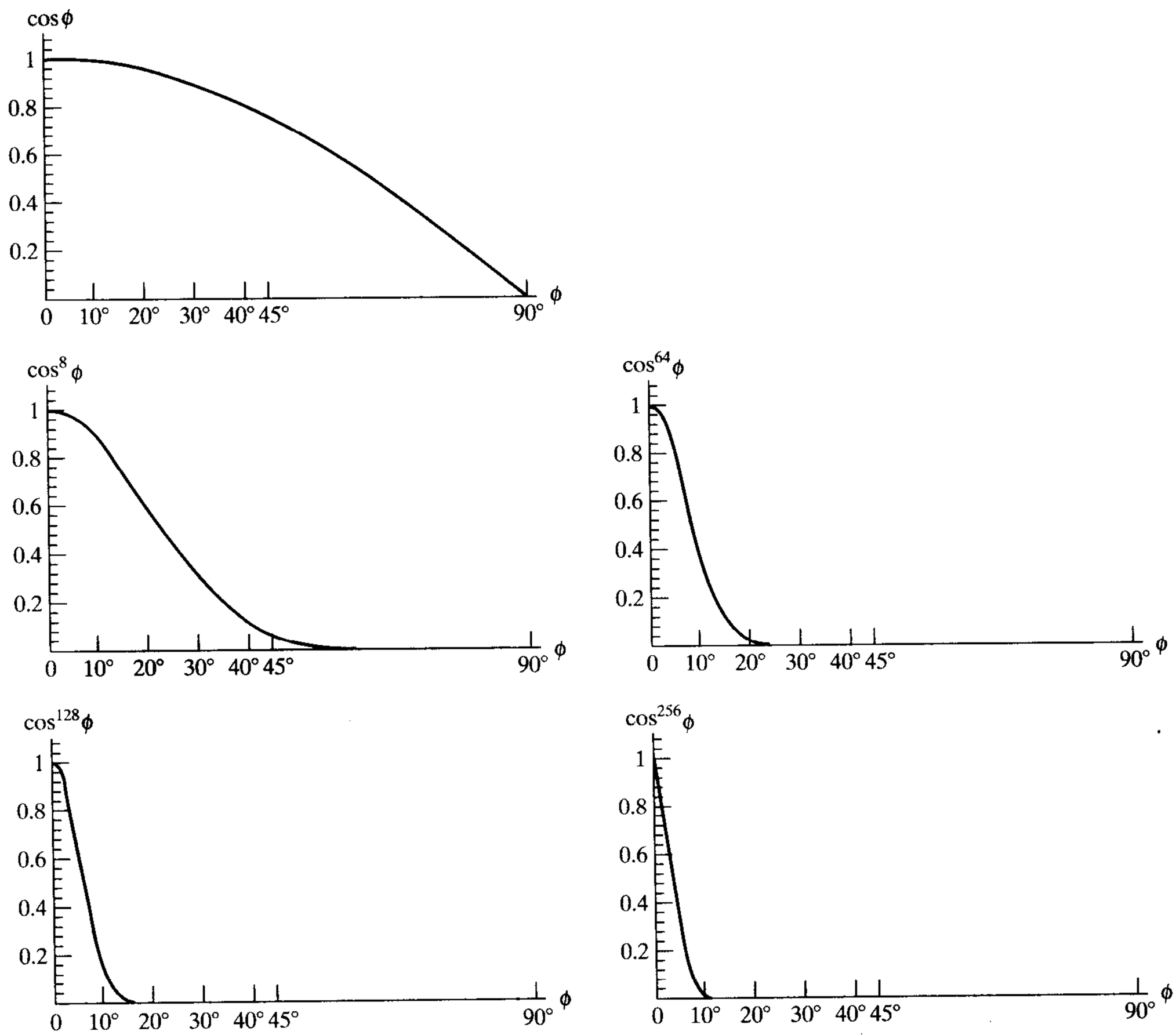


图 10.18　五个不同的镜面指数 n_s 对应的 $\cos^{n_s}\phi$

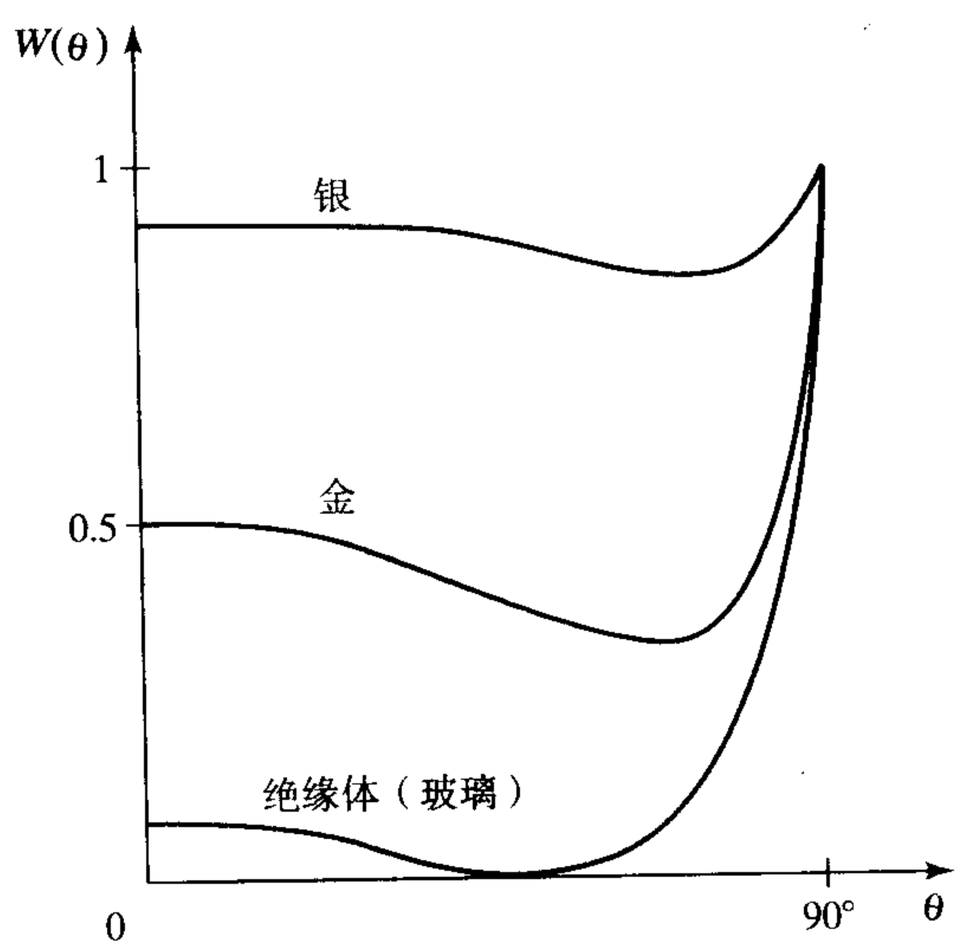


图 10.19　对于不同材质，镜面反射系数可近似表示为关于入射角的函数

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

由于**V**与**R**是观察方向和镜面反射方向的单位向量，可以利用点积**V**·**R**来计算cosϕ的值。另外，如果**V**和**L**位于法向量**N**的同一侧或光源在表面的后面，表面显示中不会有镜面反射效果。因此，假定镜面反射系数对任何材质都是一个常数，则可以根据下列等式来计算对象表面上某点处由点光源生成的镜面反射：

$$I_{l,\text{spec}} = \begin{cases} k_s I_l (\mathbf{V}\cdot\mathbf{R})^{n_s}, & \mathbf{V}\cdot\mathbf{R} > 0 \quad \text{和} \quad \mathbf{N}\cdot\mathbf{L} > 0 \\ 0.0, & \mathbf{V}\cdot\mathbf{R} < 0 \quad \text{或} \quad \mathbf{N}\cdot\mathbf{L} \leqslant 0 \end{cases} \tag{10.14}$$

反射向量**R**的方向可通过向量**L**与**N**计算出来。如图10.20所示，可以通过点积**N**·**L**得到向量**L**在法向量方向上的投影，这也等于单位向量**R**向**N**方向的投影。因此从图示中，可以有

$$\mathbf{R} + \mathbf{L} = (2\mathbf{N}\cdot\mathbf{L})\mathbf{N}$$

而镜面反射向量可以由下列等式计算出来：

$$\mathbf{R} = (2\mathbf{N}\cdot\mathbf{L})\mathbf{N} - \mathbf{L} \tag{10.15}$$

图10.21给出了在一个点光源的场景中，对应于不同的k_s与n_s的值，所得到的球面的镜面反射光照效果。

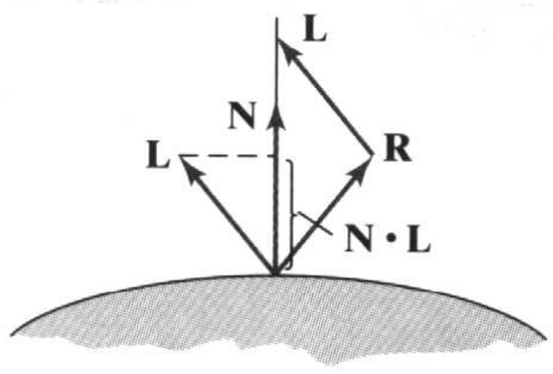


图 10.20 **L**和**R**向表面法向量**N**方向的投影都等于**N**·**L**

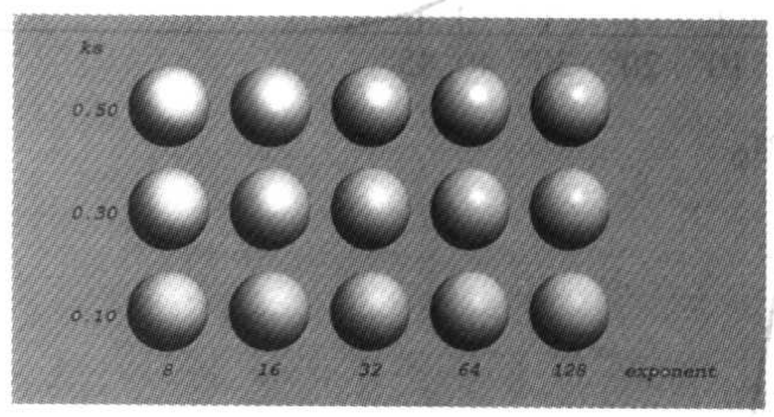


图 10.21 对于不同的镜面参数值计算单个点光源场景中球面的镜面反射效果

我们使用获得单位向量**L**的方法（参见等式（10.11））及表面位置和观察方向来计算**V**。但如果对场景中所有位置都使用一个固定的观察方向，则可设定**V** = (0.0, 0.0, 1.0)，即正z轴上的单位向量。使用常数**V**来计算镜面反射，所花时间较少，但真实感不够好。

使用向量**L**与**V**间的**半角向量**（halfway vector）**H**来计算镜面反射范围可以得到简化的Phong模型。只须以**N**·**H**替代Phong模型中的点积**V**·**R**，并利用经验cosα计算来替代经验cosϕ计算（参见图10.22）。半角向量可以从下列等式计算得到：

$$\mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{|\mathbf{L} + \mathbf{V}|} \tag{10.16}$$

对于非平面表面，**N**·**H**比**V**·**R**所需计算量较少，因为在每个表面点的**R**计算包含变化的**N**向量。若观察者与光源离对象表面足够远，且**V**与**L**均为常量，则面上所有点的**H**也为常量。如果**H**和**N**之间的夹角大于90°，则**N**·**H**为负，而我们设定镜面反射的贡献为0。

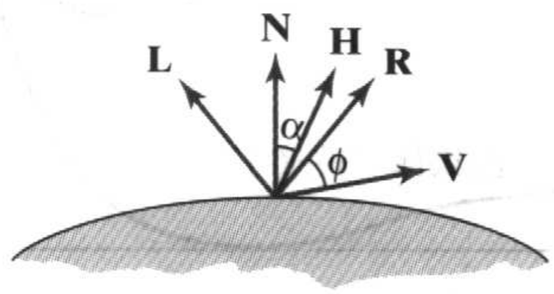


图 10.22 半角向量**H**与向量**L**、**V**的角平分线方向一致

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于给定的光源和视点，向量 **H** 是在观察方向上产生最大镜面反射的表面朝向，因此，**H** 有时称为最大亮度的表面朝向。另外，若向量 **V** 与 **L** 和 **R**（及 **N**）共面，则角 α 的值为 $\phi/2$，当 **V**、**L** 与 **N** 不共面时，$\alpha > \phi/2$，这取决于三个向量的空间关系。

10.3.4　漫反射和镜面反射的合并

对于单个点光源，可以将光照表面上某点处的漫反射和镜面反射表示为

$$\begin{aligned} I &= I_{\text{diff}} + I_{\text{spec}} \\ &= k_a I_a + k_d I_l (\mathbf{N} \cdot \mathbf{L}) + k_s I_l (\mathbf{N} \cdot \mathbf{H})^{n_s} \end{aligned} \tag{10.17}$$

如果光源位于表面之后，则表面仅有环境光的光照，而如果 **V** 和 **L** 位于法向量 **N** 的同一侧则没有镜面反射效果。图 10.23 显示了等式（10.17）中各项所产生的对象表面的光照效果。

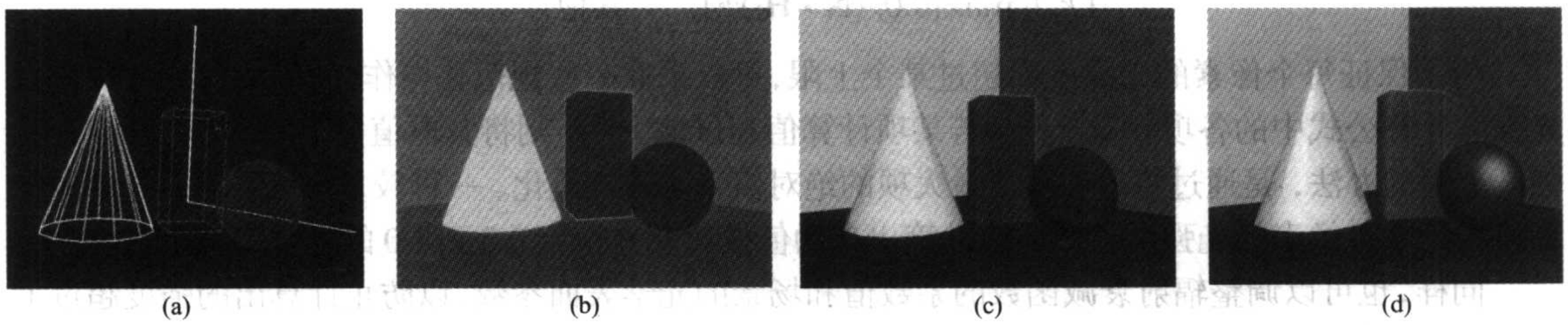


图 10.23　(a)中为一场景的线框图；(b)中只考虑环境光效果，使用不同颜色来表示每个物体的不同的表面；(c)中的光照效果考虑了从单个点光源产生的（设置表面 $k_s = 0$）环境光和漫反射；(d)中的光照效果是在单点光源场景中同时考虑环境光、漫反射和镜面反射的结果

10.3.5　多光源的漫反射和镜面反射

我们可以在场景中放置多个点光源。对于多个点光源的情况，可以在任意一个表面点上叠加各个光源所产生的光照效果：

$$\begin{aligned} I &= I_{\text{ambdiff}} + \sum_{l=1}^{n} [I_{l,\text{diff}} + I_{l,\text{spec}}] \\ &= k_a I_a + \sum_{l=1}^{n} I_l [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s}] \end{aligned} \tag{10.18}$$

10.3.6　表面的光发射

除了从光源反射光之外，场景中某些表面还会发射光。例如，房间的场景可以包括灯泡或吸顶灯，而门外夜景中可以包括街灯、商店招牌和汽车前灯。我们可在光照模型中简单地加入发射项 $I_{\text{surfemission}}$ 来模拟表面光发射，这与使用环境光等级来模拟背景光的方法一致。然后将这一表面发射项加到由光源导致的表面反射及背景光照明上。

要考虑发光表面对其他对象的照明，可在该表面后放一个方向光源来生成一个通过该表面的光锥。或者可用一组分布在该表面的点光源来模拟，然而，由于额外的计算时间的关系，基本光照模型中一般不会使用发光表面来照明其他表面，而宁愿将发光表面当做扩展光源来逼近。这将为表面生成白炽效果。模拟发光表面的更有真实感的模型是 10.12 节将讨论的辐射度模型。

10.3.7　考虑强度衰减和高光的基本光照模型

包含多光源、衰减因子、方向光效果（高光）、无穷远光源和发光表面的表面反射的通用、单色照明模型为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$I = I_{\text{surfemission}} + I_{\text{ambdiff}} + \sum_{l=1}^{n} f_{l,\text{radatten}} f_{l,\text{angatten}}(I_{l,\text{diff}} + I_{l,\text{spec}}) \tag{10.19}$$

辐射衰减函数$f_{l,\text{radatten}}$用等式（10.2）进行计算，而角度衰减函数用等式（10.5）进行计算。对每一个光源，表面点的漫反射计算如下：

$$I_{l,\text{diff}} = \begin{cases} 0.0, & \mathbf{N} \cdot \mathbf{L}_l \leqslant 0.0 \text{（对象后面的光源）} \\ k_d I_l (\mathbf{N} \cdot \mathbf{L}_l), & \text{其他} \end{cases} \tag{10.20}$$

由点光源照明导致的镜面反射项用类似的公式计算：

$$I_{l,\text{spec}} = \begin{cases} 0.0, & \mathbf{N} \cdot \mathbf{L}_l \leqslant 0.0 \\ & \text{（对象后面的光源）} \\ k_s I_l \max\{0.0, (\mathbf{N} \cdot \mathbf{H}_l)^{n_s}\}, & \text{其他} \end{cases} \tag{10.21}$$

为了保证每个像素的光强度不超过某个上限，可以采取一些规范化操作。一种简单的方法是对光强度计算公式中的各项设置上限。若某项计算值超过该上限，则将其取值为上限。另一种弥补光强度上溢的办法，是通过将各项除以最大项的绝对值来实现规范化。一种较复杂的方法是：首先计算出场景中各像素的光强度，然后将计算出来的值按比例变换至0.0到1.0的光强度范围内。

同样，也可以调整辐射衰减函数的系数值和场景的光学表面参数，以防止计算出的强度超过上限。这是在单一光源照明一个场景时限制强度的有效方法。然而，一般都不允许计算所得强度值超过1.0，并且要将负值调整为0.0。

10.3.8 RGB颜色考虑

对于RGB颜色描述，光照模型中的每一强度描述是一个指定该强度的红、绿和蓝分量的三元素向量。这样，对每一个光源，$I_l = (I_{lR}, I_{lG}, I_{lB})$。类似地，反射系数也用RGB分量指定：$k_a = (k_{aR}, k_{aG}, k_{aB})$、$k_d = (k_{dR}, k_{dG}, k_{dB})$和$k_s = (k_{sR}, k_{sG}, k_{sB})$。然后使用分开的表达式计算表面颜色的每个分量。例如，一个点光源的漫反射和镜面反射的蓝色分量用等式（10.20）和（10.21）的修改表达式计算：

$$I_{lB,\text{diff}} = k_{dB} I_{lB} (\mathbf{N} \cdot \mathbf{L}_l) \tag{10.22}$$

和

$$I_{l,\text{spec}} = k_{sB} I_{lB} \max\{0.0, (\mathbf{N} \cdot \mathbf{H}_l)^{n_s}\} \tag{10.23}$$

表面常常用白色光源来照明，但为了特殊效果或室内光，也可能使用其他颜色的光源。然后再设定反射系数以便模拟特定的表面颜色。例如，如果希望一个对象的表面为蓝色，则只须为蓝色反射系数k_{dB}在0.0到1.0范围内选择一个非零的值，而将红色和绿色反射分量置为0（$k_{dR} = k_{dG} = 0.0$）。这样，将吸收入射光线中所有非零的红色或绿色分量，而只反射蓝色分量。

Phong在他最初的镜面反射模型中，将参数k_s设置为一个与对象表面颜色无关的常量，这使镜面反射光线颜色与入射光相同（通常为白色），使对象表面看上去像是塑料。对于非塑料材料，镜面反射的颜色是关于表面材质属性的函数，可能会与入射光线及漫反射光颜色不同。我们可以将镜面反射系数与颜色相关联（如等式（10.23）），从而近似模拟这些表面上的镜面反射效果。图10.24给出了一个粗糙对象表面的颜色反射。图10.25及图10.26给出了金属表面反射的图形。图10.27给出了多个彩色光源在对象表面的反射光线。

设置表面颜色的另一种方法是，为每个表面定义漫反射和镜面反射的颜色向量，而将反射参数定为单值常数。例如，对于RGB颜色表示，两个表面颜色向量的分量可表示为(S_{dR}, S_{dG}, S_{dB})及(S_{sR}, S_{sR}, S_{sB})，反射光线的蓝色分量（参见等式（10.22））可以按下列等式进行计算：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$I_{lB,\text{diff}} = k_d S_{dB} I_{lB}(\mathbf{N} \cdot \mathbf{L}_l) \tag{10.24}$$

该方法提供了较大的灵活性，因为表面颜色参数可以独立于反射率进行设置。

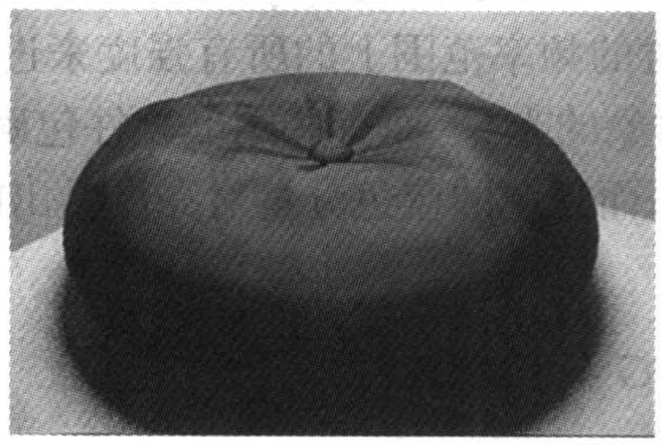


图 10.24　一个黑色尼龙坐垫表面的光线反射，它选用织布材质，并由 Monte-Carlo 光线跟踪法绘制而成（Cornell 大学计算机图形学项目组的 Stephen H.Westin 提供）

图 10.25　一个茶壶的光线反射，其反射参数设置为模拟磨光的铝表面，采用 Monte-Carlo 光线跟踪法绘制而成（Cornell 大学计算机图形学项目组的 Stephen H.Westin 提供）

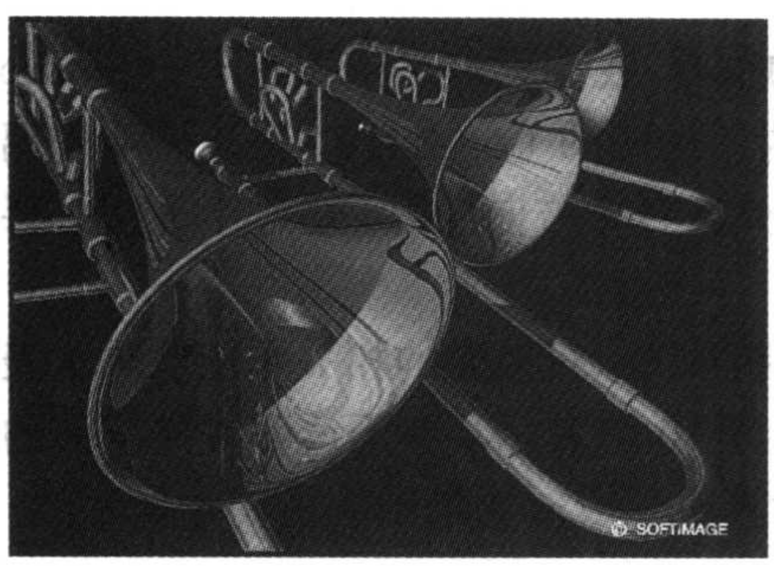


图 10.26　长号的光线反射，其反射参数设置为模拟光滑黄铜表面（SOFTIMAGE 公司提供）

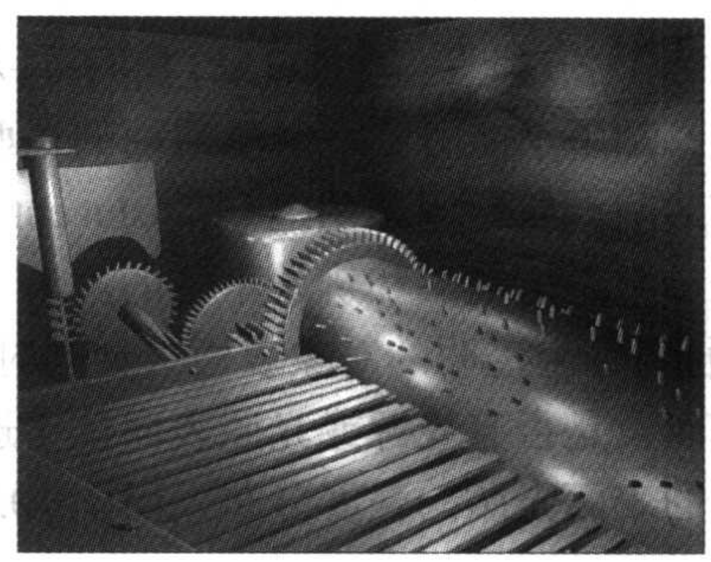


图 10.27　由多个不同颜色的光源所产生的光线反射（SUN 公司提供）

在有些图形软件包中，通过允许对一个光源赋以多种颜色来提供光照参数，其中每一种颜色对表面贡献一部分光照效果。例如，一种颜色用做场景中的一般背景光。类似地，另一光源颜色可用于漫反射计算，而第三种光源颜色可用于镜面反射计算。

10.3.9　其他颜色表示

除了 RGB 表示之外，还可以使用多种描述颜色的模型。例如，可以用青色、品红和黄色分量来表示一个颜色，或用特定的色彩、亮度和色饱和度来描述颜色。我们可以在光照模型中使用这些表示方法中的任一种，包括使用多于三个分量的描述。作为一个例子，等式（10.24）可使用波长 λ 来表达任何光谱颜色：

$$I_{l\lambda,\text{diff}} = k_d S_{d\lambda} I_{l\lambda}(\mathbf{N} \cdot \mathbf{L}_l) \tag{10.25}$$

计算机图形学中使用的各种颜色表示将在第 12 章详尽讨论。

10.3.10　亮度

亮度（luminance）是颜色的另一特征，有时也称为光能量。亮度给出了颜色亮或暗的程度信息，是我们在观察光照时感知亮度变化的心理上的度量。

在物理上，颜色用可见辐射能（光）的频率范围来描述，而亮度则按在特定照明中的强度成分的加权总和来计算。由于任一实际照明中包含连续范围的各种波长，因此亮度值计算如下：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\text{luminance} = \int_{\text{visible}\,f} p(f)\,I(f)\,df \tag{10.26}$$

该计算式中的参数$I(f)$表示在特定方向辐射的波长为f的光成分。参数$p(f)$是按经验确定的比例函数，它按频率和光照程度而变化。该集成针对光中包含的频率范围上的所有强度来进行。

对灰度和单色显示，我们只需要用亮度值来描述对象的光照。而有些图形软件包确实用亮度来表示光照参数。光源的绿色分量对亮度做出的贡献最大，而光源的蓝色分量对亮度做出的贡献最小，因此RGB光源的亮度一般计算如下：

$$\text{亮度} = 0.299R + 0.587G + 0.114B \tag{10.27}$$

有时，增加每一颜色绿色分量的贡献可得到更好的光照效果。一种推荐的计算是$0.2125R + 0.7154G + 0.0721B$。亮度参数经常用符号$Y$表示，它对应于$XYZ$颜色模型（参见12.3节）中的$Y$分量。

10.4 透明表面

对于如窗玻璃一类的对象，我们可以看到其后面的东西，我们称该对象为透明的（transparent）。类似地，如果不能看到一对象后面的东西，则该对象为不透明的（opaque）。另外，毛玻璃和某些塑料等透明对象是**半透明的**（translucent），因此透过的光在各个方向漫射。透过半透明材料观察到的对象常常是模糊的和辨认不清的。

通常，一个透明的对象表面上会同时产生反射光和折射光。折射光的相关作用取决于表面的透明程度以及是否有光源或光照表面位于透明表面之后。图10.28表示了在不透明对象前面的一个透明对象的表面光照的强度分布。而图10.29给出了在计算机生成的场景中的透明效果。

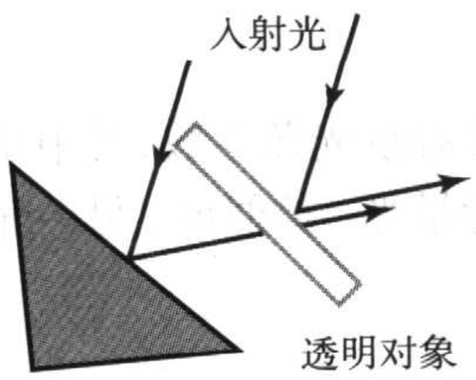


图10.28 从一个透明表面发散出来的光线通常由反射光和折射光两部分组成

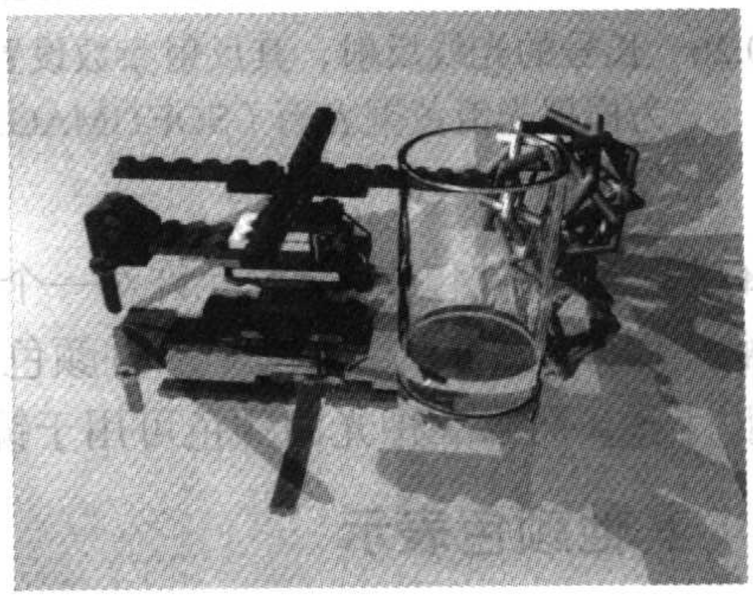


图10.29 包括一个透明玻璃杯、展示从杯子后面对象的光透射和玻璃表面的光反射的场景光线跟踪视图

10.4.1 半透明材料

在一个透明对象的表面，可能同时发生漫折射和镜面折射。当表示半透明对象时，漫折射效果很重要。经过一个半透明材料后，背景对象变成模糊影像。可以通过将背景对象均匀分布在一个有限范围或使用光线跟踪方法来模拟半透明性。但这些操作很耗时，并且基本的光照模型仅考虑了镜面透射效果。

10.4.2 光折射

通过考虑光**折射**（refraction），可以模拟真实透明效果。当光线入射到一个透明对象表面时，一部分光线被反射，另一部分被折射，如图10.30所示。由于不同对象中光线的速度不同，因此折

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

射光线的路径与入射光线也不同。折射光线的方向，用**折射角**（angle of refraction）指定，是关于各材质的**折射率**（index of refraction）及入射方向的函数。将材质的折射率定义为光线在真空中的速度与其在材质中的速度之比。折射角 θ_r 可用 **Snell 定律**计算：

$$\sin\theta_r = \frac{\eta_i}{\eta_r}\sin\theta_i \tag{10.28}$$

这里 θ_i 是入射角，η_i 是入射材料的折射率，而 η_r 是折射材料的折射率。

事实上，材质的折射率也依赖于其他参数，如材料的温度和入射光的波长。这样，入射白光的各种彩色成分以不同角度折射，按温度而变化。在诸如结晶石英等各向异性的材料中，光的速度依赖于方向，而某些透明材料表现为双重折射（double refraction），即生成两个折射光。在大多数情况下，可以赋给每一种材料一个平均折射率，如表 10.1 所列。空气的折射率接近 1.0，冕牌玻璃的折射率约为 1.61，使用等式（10.28），则当入射角为 30° 时，光通过冕牌玻璃时的折射角约为 18°。

表 10.1　普通材料的平均折射率

材料	折射率
真空（空气或其他气体）	1.00
冕牌玻璃材料	1.52
厚的冕牌玻璃	1.61
燧石玻璃材料	1.61
厚的燧石玻璃	1.92
岩盐	1.55
石英	1.54
水	1.33
冰	1.31

图 10.31 表示一束光线折射穿过一个薄玻璃片所经历的路径变化。折射的最终结果是将入射光线在离开该材料时进行平移。由于计算等式（10.28）中的三角函数很费时，可以将入射光路径平移一个小的位移，从而简单地表示给定材料的折射效果。

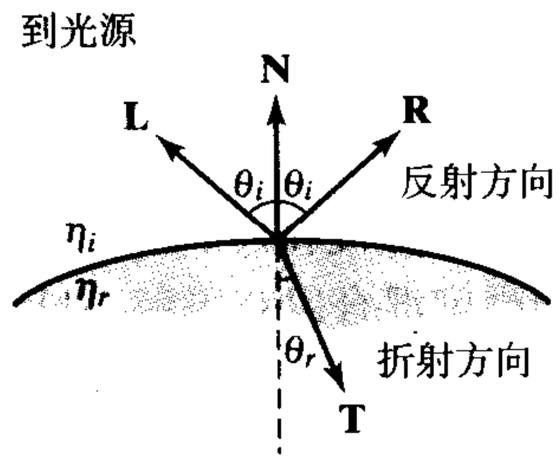


图 10.30　一束光线落在折射率为 η_r 的对象表面上，所产生的反射光线 **R** 及折射光线 **T**

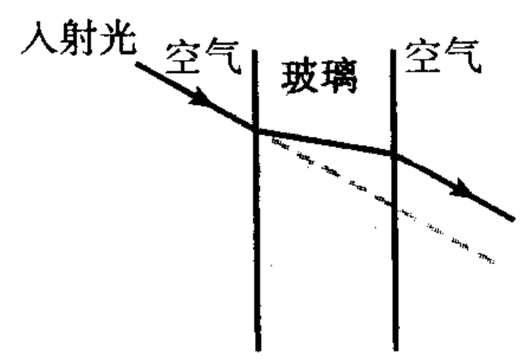


图 10.31　穿过一个薄玻璃片的光线折射，最终折射出来的光束方向平行于入射光线（虚线）

由 Snell 定律及图 10.30 中的图解，可以得到折射方向 θ_r 上的单位透射向量 **T**：

$$\mathbf{T} = \left(\frac{\eta_i}{\eta_r}\cos\theta_i - \cos\theta_r\right)\mathbf{N} - \frac{\eta_i}{\eta_r}\mathbf{L} \tag{10.29}$$

其中，**N** 为对象表面的单位法向量，**L** 为光源方向的单位向量。透射向量 **T** 可用于计算折射光与透明表面后的对象的交点。考虑场景中的折射效果，可以生成高度真实感的图形，但确定折射路径和对象求交需要相当大的计算量。大多数扫描线图像空间算法利用近似值来表示光线透射以减少处理时间。精确的折射效果要用光线跟踪法来显示（参见 10.11 节）。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.4.3 基本的透明模型

一个简单的表示透明对象的方法是不考虑折射导致的路径平移。在实际操作中，该方法假定各对象间的折射率不变，这样折射角总是与入射角相同。该方法加速了光强度的计算，并且对于较薄的多边形表面，可以生成合理的透明效果。

我们可以只用**透明系数**k_t，将从背景对象穿过表面的透射强度I_{trans}与由透明表面发出的反射强度I_{refl}结合在一起（参见图10.32）。给定参数k_t一个0.0与1.0之间的值以指定多少背景光线被透射，则对象表面的总的光强度可表示为

$$I = (1 - k_t)I_{refl} + k_t I_{trans} \tag{10.30}$$

其中，项$(1-k_t)$为**不透明因子**（opacity factor）。例如，如果设定透明因子为0.3，则30%的背景光与70%的反射的表面光照相混合。

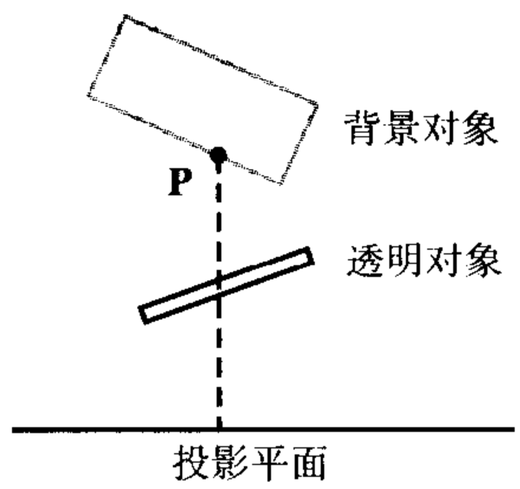


图10.32 背景对象上点P处的光强度可以与垂直投影线（虚线）所经过的透明对象表面的反射强度相结合考虑

只要使用深度优先（从后到前），上面的过程可用来混合任意多透明和不透明对象的光照效果。例如，透过图10.29中的玻璃杯，可看到在两个透明表面后面的不透明对象。类似地，透过汽车的挡风玻璃可以看到汽车中的对象及后窗后面的对象。

对于高度透明的对象，可以将k_t设置为接近1.0的值，而几乎不透明的对象仅由背景对象透射出极少的光，则可以设置k_t的值接近0.0。也可以将k_t设置为一个关于表面位置的函数，这样，对象的不同部分可以根据k_t的值来决定折射多少背景光强度。

可以修改深度排序可见性算法来实现透明效果，先将表面按深度排序，再确定是否有可见面是透明的。如果有，其反射的表面强度与在其后对象的表面强度混合以获得每一投影表面点的像素强度。

透明效果还可用修改的深度缓存方法实现。将场景中的表面分成两组，先处理所有不透明的表面。这时，帧缓存包含了可见面的强度，而深度缓存包含了它们的深度。然后，将透明对象的深度值与先前存储在深度缓存中的值进行比较。如果所有透明对象表面均可见，则计算出反射光强度并与先前存储在帧缓存中的不透明表面的光强度相结合。可以对该方法进行修改以得到更准确的效果，即增加对透明表面深度及其他参数的存储。这样，透明表面间的深度值与不透明表面的深度值可以互相进行比较，可以通过将可见透明表面的强度与其后面的可见不透明表面的强度相结合考虑，从而进行实际的绘制。

另一个方法是A缓存算法。对于A缓存中每一个像素位置，将保存所有覆盖它的表面并按深度顺序排序。然后根据正确的可见性顺序将深度上重叠的透明和不透明的表面强度相结合考虑，以产生该像素点的最终平均强度。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.5　雾气效果

对象颜色上的雾气效果是光照模型中有时会考虑的另一因素。雾气使颜色变淡、使对象变模糊。因此我们可以按照模拟空气的灰尘、烟或雾的多少来指定一个函数修改表面颜色。雾气效果常用指数衰减函数来模拟：

$$f_{\text{atmo}}(d) = e^{-\rho d} \tag{10.31}$$

或

$$f_{\text{atmo}}(d) = e^{-(\rho d)^2} \tag{10.32}$$

赋给d的值是从观察位置到对象的距离。我们还在这两个指数函数中使用参数ρ来为雾气设定一个正密度。较高的ρ值生成较稠密的雾气且导致表面更柔和。在一对象的表面颜色计算出来后，将该颜色与雾气函数之一相乘来减少其强度，减少量依赖于为雾气密度设定的值。

我们可以通过使用线性的深度提示函数（9.9）来简化雾气衰减函数从而代替指数函数。这使得远距离对象的表面颜色强度减弱，但这样就失去了改变雾气密度的控制。

我们有时可能要模拟雾气的颜色。例如，有烟房间的空气可以用蓝灰色或用淡灰色建模。混合雾气颜色和对象颜色的计算如下：

$$I = f_{\text{atmo}}(d) I_{\text{obj}} + [1 - f_{\text{atmo}}(d)] I_{\text{atmo}} \tag{10.33}$$

这里的f_{atmo}是一个指数或线性雾气衰减函数。

10.6　阴影

可以使用隐藏面算法确定光源所不能照明的区域。将视点置于光源位置，可以确定哪些表面是不可见的，这些就是阴影区域。一旦对所有光源确定出阴影区域，这些阴影可以看做是表面图案而保存于图案数组中。图 10.33 显示了动画特写中的一张脸。图中的所有阴影区域即为在天花板光源位置所看不见的表面。因此，抬起的头和手臂得到照明，但沿光源视线的手臂后的面部是在阴影中。图 10.29 中的场景表示由多个光源所产生的阴影效果。

图 10.33　在电影*Final Fantasy: TheSpirits Within*的一个动画特写中映射到 Aki Ross 博士脸上的阴影图案（Square Pictures 公司提供）

只要光源位置不变，则对于任意选定的观察位置，由隐藏面算法所生成的阴影图案均是有效的。从视点所看到的对象表面，可以根据光照模型结合纹理因素进行绘制。我们既可以显示仅考虑环境光影响的阴影区域，也可以将环境光与特定的表面纹理相结合。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.7 照相机参数

到目前为止，所考虑的观察和照明过程生成了鲜明的图像，与使用针孔照相机对场景摄取的一样。但我们对实际场景照相时，可通过调节照相机来选择对象范围。其他对象或多或少离开焦点，这依赖于场景中该对象的深度分布。我们可以通过将每一位置投影到覆盖多个像素位置的区域，使该对象合并到另外对象中生成模糊的投影图案，从而在计算机图形程序中模拟聚焦位置之外的景色。这一过程与反走样中使用的方法类似，且我们可以将该照相机效果结合到扫描线或光线跟踪算法中。包括了聚焦效果后可使计算机生成的场景看起来更真实，但聚焦计算很耗时。设定照相机参数来模拟聚焦效果的方法将在 10.11 节中讨论。

10.8 光强度显示

由光照模型计算出来的光强度可以是 0.0 到 1.0 之间的任意值，但计算机图形系统只能显示一个有限强度集。因此，计算出的强度值必须转换为特定图形系统中的一个系统允许值才能进行显示。另外，系统强度等级的允许数目必须按其对应到我们的眼睛能感受到强度差别的程度来分布。在使用 2 级系统显示场景时，可将计算出的强度转换成半色调图案，如 10.9 节中所讨论的那样。

10.8.1 分配系统强度等级

对于任何系统，强度等级的允许数量可分布到从 0.0 到 1.0 的范围内，以使该分布对应于我们对不同等级之间有相同光强度间隔的感觉。我们对光强度的感觉与对声音强弱的感觉是相同的，即按对数等级变化。这表示如果两个强度的比率与另外两个强度之间的比率相同，则我们所感觉到的两个强度之间的差异也相同。例如，我们所感知的强度 0.20 与 0.22 之间的差异，这与 0.80 与 0.88 之间的差异是相同的。因此，为了显示 $n+1$ 个感觉上亮度间隔相等的连续强度等级，必须将监视器上的强度等级按下式进行分布，从而使得相邻强度之间的比率为常量：

$$\frac{I_1}{I_0} = \frac{I_2}{I_1} = \cdots = \frac{I_n}{I_{n-1}} = r \tag{10.34}$$

这里 I 表示光的一个颜色分量的强度。监视器上所能显示的最低级为 I_0，最高级为 I_n，任意的中间强度可以用 I_0 项来表示：

$$I_k = r^k I_0 \tag{10.35}$$

为一个特定系统给定 I_0 值和 n，即可将 $k=n$ 代入前面表达式中计算出 r 的值。因为 $I_n = 1.0$，所以

$$r = \left(\frac{1.0}{I_0}\right)^{1/n} \tag{10.36}$$

这样，在等式（10.35）中计算 I_k 可以写为

$$I_k = I_0^{(n-k)/n} \tag{10.37}$$

例如，若一个系统中 $I_0 = 1/8$ 且 $n=3$，则 $r=2$，四个强度等级分别为：1/8、1/4、1/2、1.0。

最低强度值 I_0 由监视器中的属性所决定，通常在 0.005 至 0.025 之间。这一视频监视器上残留的强度是由屏幕荧光粉的反射光所导致的。因此，监视器上的“黑色”区域的强度值总是大于零。对于每像素 8 位（$n=255$）且 $I_0 = 0.01$ 的黑白监视器，相邻强度等级的比率约为 $r = 1.0182$。其 256 个强度等级的近似值为 0.0100, 0.0102, 0.0104, 0.0106, 0.0107, 0.0109,..., 0.9821，1.0000。

RGB 颜色系统使用类似的方法。例如，可将 k 级强度的蓝色分量通过最低等级的蓝色值表达为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$I_{Bk} = r_B^k I_{B0} \tag{10.38}$$

其中

$$r_B = \left(\frac{1.0}{I_{B0}}\right)^{1/n} \tag{10.39}$$

n 为强度等级的数目。

10.8.2　gamma 校正与视频查找表

在视频显示器上显示彩色或黑白图像时，感觉到的亮度变化是非线性的，但光照模型产生一组线性变化的强度值。由光照模型所得的 RGB 色(0.25, 0.25, 0.25)表示颜色(0.5, 0.5, 0.5)强度的一半。通常，将这些计算出来的强度作为0到255范围中的整数值存储于一个图像文件中，RGB 三分量中的每一个对应一个字节。该强度文件也是线性的。这样，对应值为(64, 64, 64)的像素的光强度是值为(128, 128, 128)的像素的一半。控制冲击屏幕荧光粉电子数目的电子枪电压生成由图10.34中的**监视器响应曲线**（monitor response curve）确定的亮度等级。因此，显示强度值（64, 64, 64）表现为值(128, 128, 128)一半的亮度。

为了校正监视器的非线性，图形软件包中使用**视频查找表**（video lookup table）来调整线性像素值。利用分量函数可以描述监视器响应曲线：

$$I = aV^{\gamma} \tag{10.40}$$

参数 I 为显示强度，参数 V 为电子枪电压，参数 a 与 γ 的值取决于图形系统中监视器的属性。这样，如果希望显示一个特定的强度值 I，则产生该强度的电压值为

$$V = \left(\frac{I}{a}\right)^{1/\gamma} \tag{10.41}$$

这种计算称为光强度的 **gamma 校正**（gamma correction），监视器中的 gamma 值通常为 1.7 到 2.3 之间。美国国家电视系统委员会（National Television System Committee，NTSC）的信号标准为 $\gamma = 2.2$。图 10.35 表示一个使用 NTSC gamma 值的 gamma 校正曲线，其中强度和电压均规范化到 0 到 1.0 之间。等式（10.41）可用于建立视频查找表，将图像文件中的整数像素值转换为控制电子枪电压的值。

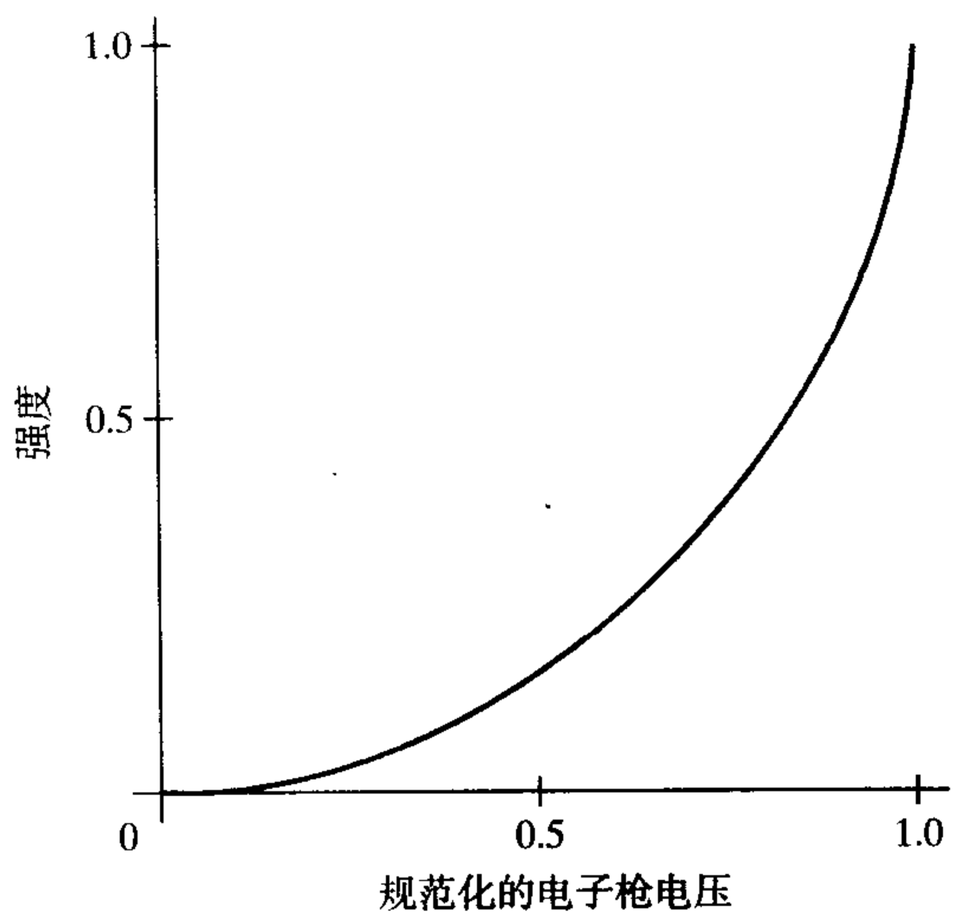


图 10.34　一个典型的监视器响应曲线，其中显示的强度（亮度）表示为规范化电子枪电压的函数

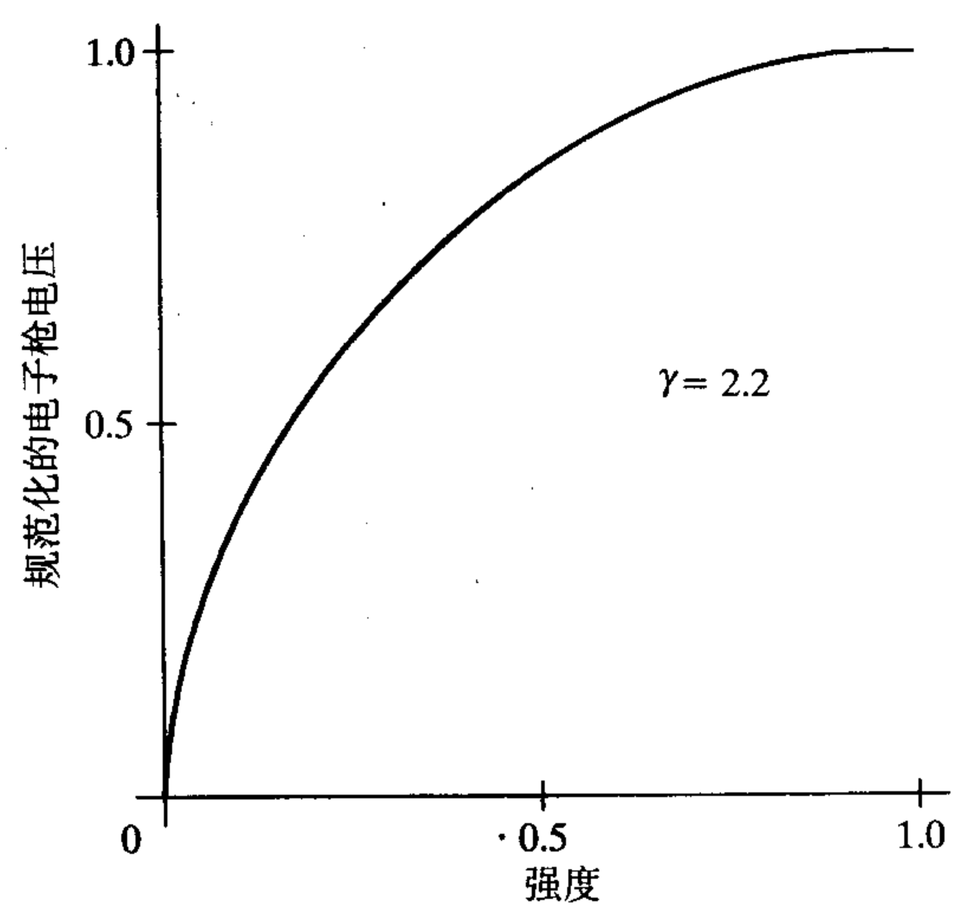


图 10.35　使用 gamma 校正（$\gamma = 2.2$）将规范化强度映射至电子枪电压的视频查找校正曲线

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们可以将gamma校正与对数强度映射相结合来生成查找表。如果I为一光照模型中的输入强度值，那么首先从等式（10.34）或等式（10.37）所生成的数值表中找到最接近的强度I_k。或者，通过下列计算确定这种强度值的等级：

$$k = \text{round}\left[\log_r\left(\frac{I}{I_0}\right)\right] \tag{10.42}$$

然后，使用等式（10.37）计算出该等级的强度值。一旦有了强度值I_k，就可以计算出电子枪的电压：

$$V_k = \left(\frac{I_k}{a}\right)^{1/\gamma} \tag{10.43}$$

我们可将V_k的值置入查找表，而将k的值存入帧缓存的像素单元中。如果某个特定系统中没有查找表，则将计算出来的V_k值直接存入帧缓存中。这种将强度向对数等级转化并根据等式（10.43）计算V_k的方法也称为gamma校正。

如果监视器的视频放大器设计成直接将线性输入像素值转化为电子枪电压，则无法将两个强度转换过程结合在一起。在这种情况下，gamma校正内置在硬件中，必须预先计算好对数值I_k并将其存入帧缓存（或颜色表）中。

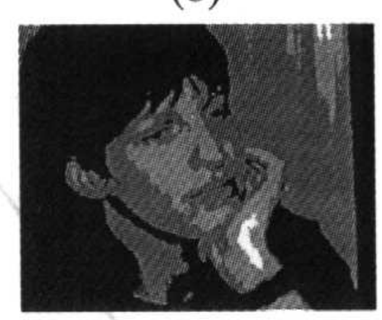
10.8.3 显示连续色调的图像

通常，高质量的计算机图形系统为每个颜色分量至少提供了256个强度等级，但在许多应用场合中，使用等级数目不到256的显示设备也能得到较好的结果。一个4级系统为连续色调图像提供了最基本的绘制能力，而真实感的图像则需要系统具备32至256种强度等级。

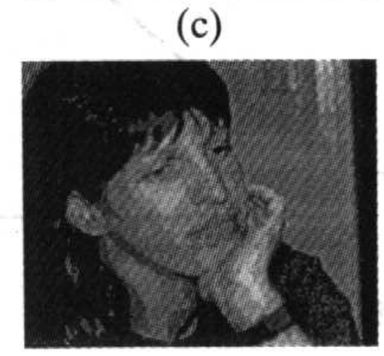
图10.36给出了几种不同强度等级所显示出的连续色调图像。当使用较少的强度等级来显示原来的连续色调图像时，不同强度区域的边界（称为轮廓线）将清晰可见。在2级系统中，照片中脸部特征几乎无法辨认。使用4级系统，可以开始辨认出原来的明暗图像，但轮廓效果仍过于强烈。采用8个强度等级，轮廓效果仍然明显，但我们已经开始得到较好的原有图像。若有16或更多的强度等级，轮廓效果就会消失而显示效果越来越接近原来的图像。使用32个以上的强度等级，将使生成的连续色调图像与原图像的差别变得细微。

(a)

(b)



(c)



(d)

图 10.36 (a)一个连续色调图像，(b)用两个强度等级打印，(c)用 4 个强度等级，(d)用 8 个强度等级

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.9　半色调模式和抖动技术

当一个输出设备的光强度范围较小时，可以将多个像素单元组合起来表示一种强度值，从而使可以得到的强度数目明显增加。当我们观察一个包含几个像素的小区域时，眼睛往往通过取整或将细节取平均而得到一个总体的强度效果。因而，一些输出设备特别是2级监视器和打印机，可以利用这种视觉效果来产生原来需要多种强度值的图像。

连续色调的图像被复制后可在报纸、杂志和书刊上出版，复制过程中有一个打印步骤，称为**半色调处理**（halftoning），而复制的图片称其为**半色调图像**（halftone）。对于一张黑白照片，每个强度区域将作为白色背景上的一组黑色圆点进行复制。每个圆点的直径与强度区域所需的黑暗程度成正比，较黑暗的区域用较大的圆点打印，而较亮的区域则用较小的圆点（较白区域）来表示。图 10.37 显示了放大的一个灰度半色调复制图像的一部分。如图 10.38 所示，可以使用不同颜色和尺寸的点来印制彩色的半色调图像。书刊与杂志上的半色调图像，则是使用每厘米约 60 到 80 个不同直径的圆点打印在高质量的纸张上而形成的。报纸则使用较低质量的纸张和较低的分辨率（大约每厘米 25 至 30 个点）。

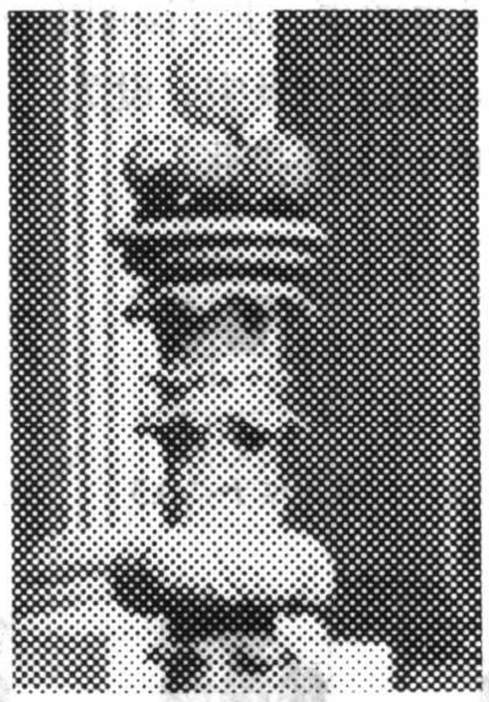


图 10.37　一个放大的用半色调技术复制的照片的一部分，示出了如何用不同尺寸的点来表示色调

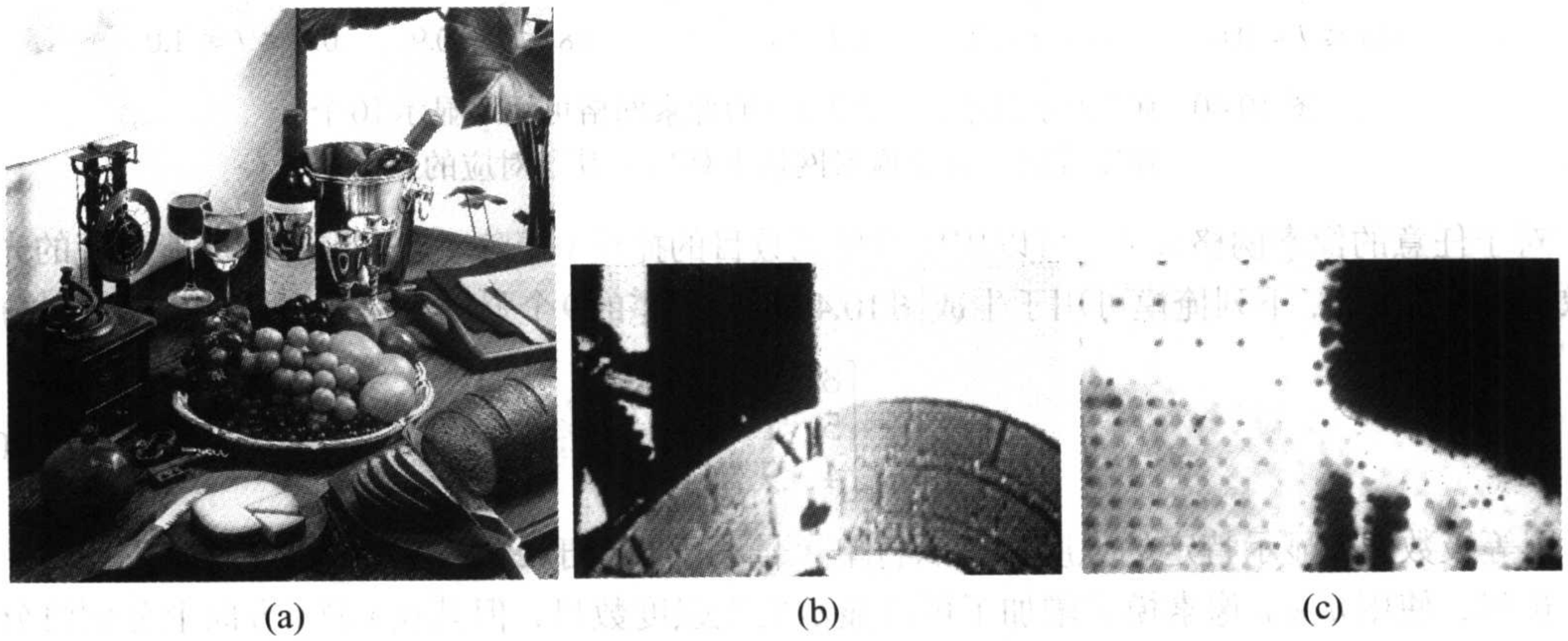


图 10.38　彩色半色调图案。(a)中的彩色半色调图案时钟的顶部在(b)中被放大 10 倍，而在(c)中被放大 50 倍（IRIS Graphics 公司提供）

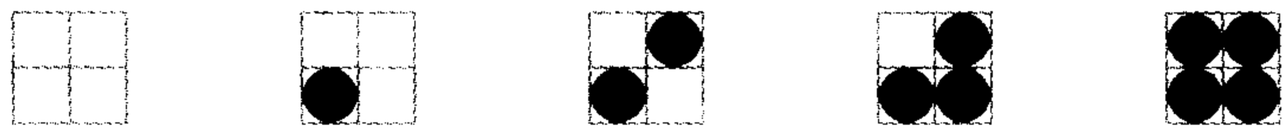
10.9.1　半色调近似

在计算机图形学中，通常使用矩形像素区域来近似半色调复制，称其为**半色调近似模式**（halftone approximation pattern）或**像素模式**（pixel pattern）。该方法可显示的强度等级数目，完全

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

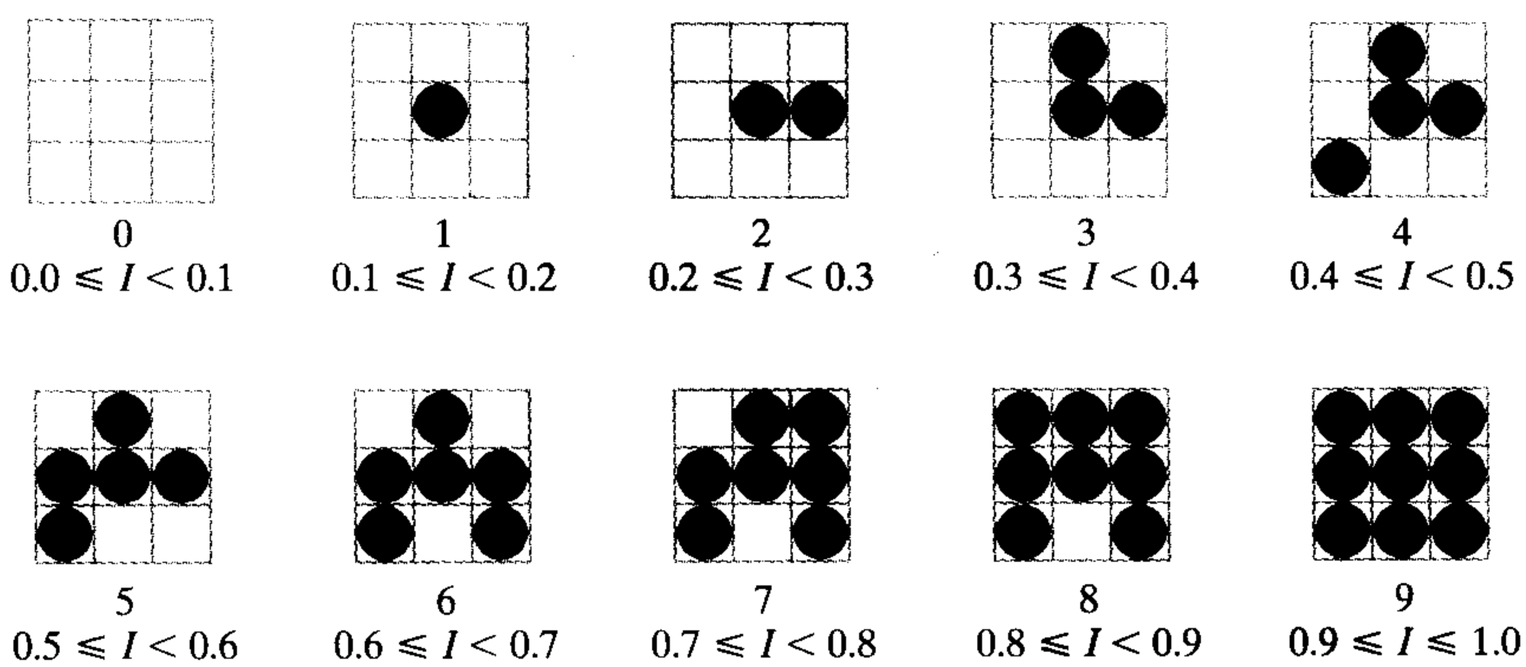
取决于矩形网格中所包含的像素数目以及系统所能显示的等级数目。在2级系统中，如果每个网格包含 $n \times n$ 个像素，则可表示 n^2+1 级光强度。图10.39表示一种在2级系统中设置像素模式以表示5级强度的方法。在模式0中，所有像素均置为off（白色），在模式1中，有一个像素置为on（黑色）；在模式4中，所有4个像素均置为on。根据图中每个网格下所列出的范围，将场景中的每个强度值 I 映射至一个特定的模式，模式0用于 $0.0 \leqslant I < 0.2$；模式1用于 $0.2 \leqslant I < 0.4$，而模式4用于 $0.8 \leqslant I \leqslant 1.0$。



0 $0.0 \leqslant I < 0.2$　1 $0.2 \leqslant I < 0.4$　2 $0.4 \leqslant I < 0.6$　3 $0.6 \leqslant I < 0.8$　4 $0.8 \leqslant I \leqslant 1.0$

图10.39　在2级系统中，用于显示5个强度等级的 2×2 像素网格，用红色圆表示“on”的像素。每个像素模式下方列出其所对应的强度值

在2级系统中，用 3×3 像素网格可以显示10个光强度等级。图10.40绘出了一个设置10个像素模式的方法，即每个等级选择不同的像素，使得该图案接近于在半色调复制图案中所使用的不断增加的圆点尺寸。也就是在较低的光强度等级中，设为“on”的像素单元接近于网格中心，而当强度等级增大时，设为“on”的像素单元向外扩张。



0 $0.0 \leqslant I < 0.1$　1 $0.1 \leqslant I < 0.2$　2 $0.2 \leqslant I < 0.3$　3 $0.3 \leqslant I < 0.4$　4 $0.4 \leqslant I < 0.5$

5 $0.5 \leqslant I < 0.6$　6 $0.6 \leqslant I < 0.7$　7 $0.7 \leqslant I < 0.8$　8 $0.8 \leqslant I < 0.9$　9 $0.9 \leqslant I \leqslant 1.0$

图10.40　在2级系统中，一个 3×3 的像素网格可用于显示10个强度等级。每个像素网格下列出了其所对应的强度值

对于任意的像素网格尺寸，可以用像素单元数目的掩模（矩阵）来表示各种可能出现的光强度的像素模式。例如，下列掩模可用于生成图10.40中大于零的9个强度等级所对应的 3×3 网格模式。

$$\begin{bmatrix} 8 & 3 & 7 \\ 5 & 1 & 2 \\ 4 & 9 & 6 \end{bmatrix} \tag{10.44}$$

为了用等级数 k 来显示特定的强度，可以将单元号小于或等于 k 的像素置为“on”。

虽然，使用 $n \times n$ 像素模式增加了可以显示的光强度数目，但其在 x 及 y 方向上分别将分辨率减少至原来的 $1/n$。例如，一个 512×512 的屏幕区域，采用 2×2 网格模式后将减小到包含 256×256 光强点的区域。使用 3×3 网格后，则 512×512 区域的每条边将减至128个光强单元。

使用像素网格的另一个问题是，当网格大小增加时，子网格模式变得更加明显。通常由显示的像素大小来确定网格的最佳尺寸。因此，对于较低分辨率的系统（每厘米有较少像素），必须使用较少的强度等级。另一方面，高质量的显示至少需要64个光强度等级。这意味着需要 8×8 的像素

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

网格。为了获得与书刊、杂志中半色调图案同等的分辨率，每厘米必须显示60个光强点。这样，每厘米共需要显示 60 × 8 = 480 个像素点。许多系统，如高质量的胶片记录仪可以满足这种分辨率。

用于半色调近似的像素网格模式，还必须使轮廓效果及原始图像中没有的其他视觉效果降为最低。可以通过逐个推导后继网格模式的方法将轮廓效果降至最小。例如，在等级 k–1 的网格模式上增加一个“on”像素单元以形成等级为 k 的模式（参见图 10.39 和图 10.40）。还可以通过避免对称模式来防止引入其他的视觉效果。例如，若采用 3 × 3 的像素网格，则大于 0 的第 3 级光强使用图10.41(a)中的模式进行表示，这比使用图10.41(b)中的所有对称排列都要好。该图中的对称模式较适于在大范围内使用第3级光强来绘制垂直、水平或对角线线段。对于胶片记录仪或打印机等设备的硬拷贝输出，单独像素的复制效率较差。因此，如图 10.42 所示，应避免那些只有单个“on”像素或所有“on”像素均不相邻的网格模式。

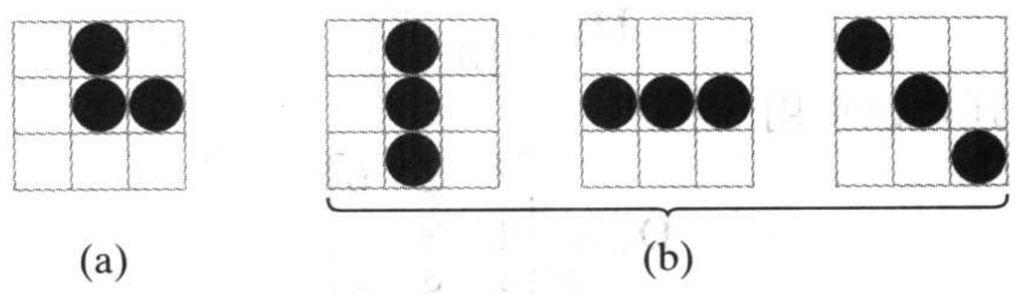


图 10.41　若采用 3 × 3 的像素网格，则大于 0 的第 3 级光强使用(a)中的模式进行表示要比(b)中的对称模式好

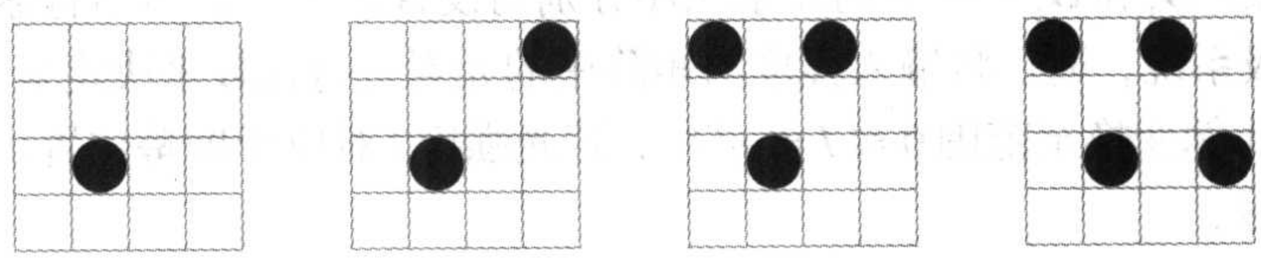


图 10.42　在一些硬拷贝设备上，使用带有孤立像素的半色调网格模式进行复制的效果较差

对于每像素可显示多于2种光强的系统，还可以使用半色调近似来增加其强度选择的数目。例如，在一个每像素可显示 4 个强度等级的系统中，可以使用 2 × 2 像素网格将强度等级数目由 4 扩展至 13。图 10.43 给出了一个分配像素强度以得到 13 种不同等级的方法，其中每个像素的强度等级均标识为 0、1、2 或 3。

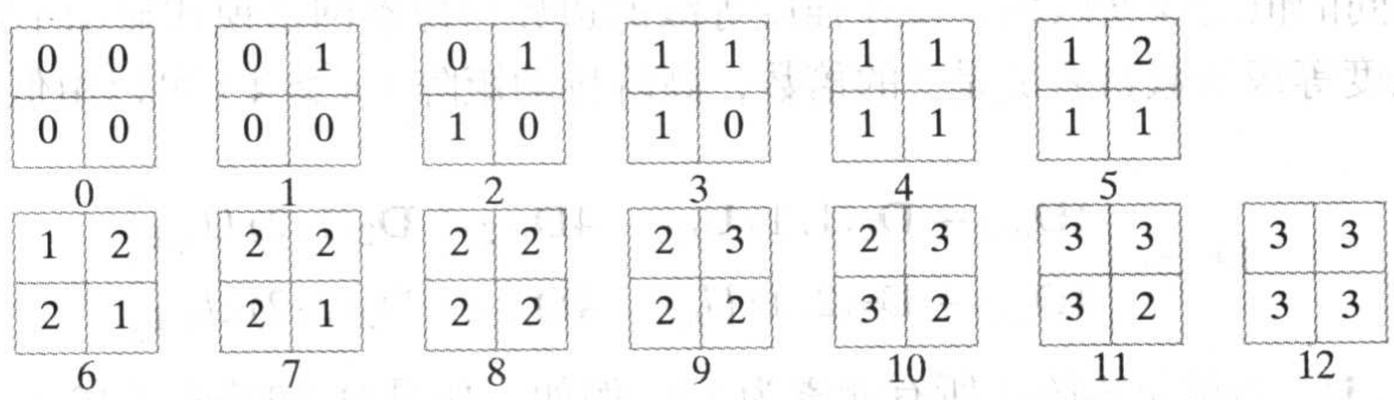


图 10.43　在一个标识为 0 到 3 的像素强度等级系统中使用 2 × 2 像素网格进行半色调近似，所得到的强度等级为 0 至 12

同样，可以在一个彩色系统中使用像素网格模式来增加显示的强度数目。例如，假定有一个每像素 3 位的 RGB 系统，在监视器中每个电子枪对应一位。这样，可以用三个荧光点显示一个像素，因此该像素可赋以 8 种颜色（包括白色与黑色）中的任一种。使用了 2 × 2 像素网格模式以后，如图 10.44 所示，将有 12 个可用于表示特定颜色值的荧光点，每个 RGB 颜色中的各分量在模式中对应 4 个荧光点。这使得每种颜色有 5 种可能的设置，也就提供了总共 125 种不同的颜色组合。

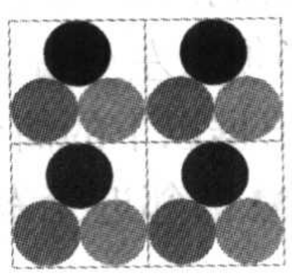


图 10.44　一个 2 × 2 像素网格模式的 RGB 图像

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.9.2 抖动技术

术语**抖动**（dithering）可用于不同的范围。起初，该术语表示不降低分辨率的半色调近似技术，例如像素网格模式。但目前该术语有时作为任一半色调近似方法的同义词，有时用做彩色半色调近似的另一术语。

我们常用**抖动噪音**（dither noise）来表示添加于像素强度之上以打乱轮廓的随机值，目前已有许多种算法可用于模拟数值的随机分布。抖动技术的效果就是在整个图像上增加噪音以柔化强度边界。

按序抖动（ordered dither）算法通过使用**抖动矩阵** $\mathbf{D}_n$ 选择强度等级，将场景中的点一一对应地映射至屏幕像素位置来生成光强度变化。为了得到 n^2 个强度等级，我们设立 $n \times n$ 抖动矩阵 $\mathbf{D}_n$，其中，每个元素表示 0 到 n^2-1 之间的不同正整数。例如，使用

$$\mathbf{D}_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \tag{10.45}$$

来生成四个光强度等级，还可以使用

$$\mathbf{D}_3 = \begin{bmatrix} 7 & 2 & 6 \\ 4 & 0 & 1 \\ 3 & 8 & 5 \end{bmatrix} \tag{10.46}$$

来生成9个强度等级。$\mathbf{D}_2$ 和 $\mathbf{D}_3$ 矩阵中的元素顺序分别与设置 2×2 与 3×3 像素网格时的像素掩模相同。对于一个2级系统，通过将输入强度与矩阵中的元素进行比较来决定显示的强度值。首先，将每个输入光强度比例变换至范围 $0 \leqslant I \leqslant n^2$ 中，若光强度 I 对应于屏幕位置 (x, y)，则按下式计算抖动矩阵的行、列号：

$$j = (x \bmod n) + 1, \qquad k = (y \bmod n) + 1 \tag{10.47}$$

若 $I > \mathbf{D}_n(j, k)$，则将位置 (x, y) 处的像素置为“on”，否则该像素值为“off”。对于RGB颜色应用来说，该过程为单个颜色分量（红、绿和蓝）分别实现。

抖动矩阵中的元素按前面所述的像素网格方法进行赋值，即在显示屏幕上尽量减少可能添加的视觉效果（如层位线效应）。当矩阵元素的值与半色调近似网格掩模相对应时，按序抖动将产生与像素网格模式相同的恒定强度区域。仅在强度等级的边界与像素网格模式显示不同。

通常，将强度等级的数目看成是2的倍数，高阶抖动矩阵（$n \geqslant 4$）可以由低阶矩阵通过迭代而获得：

$$\mathbf{D}_n = \begin{bmatrix} 4\mathbf{D}_{n/2} + \mathbf{D}_2(1,1)\,\mathbf{U}_{n/2} & 4\mathbf{D}_{n/2} + \mathbf{D}_2(1,2)\,U_{n/2} \\ 4\mathbf{D}_{n/2} + \mathbf{D}_2(2,1)\,U_{n/2} & 4\mathbf{D}_{n/2} + \mathbf{D}_2(2,2)\,U_{n/2} \end{bmatrix} \tag{10.48}$$

假定 $n \geqslant 4$，参数 $\mathbf{U}_{n/2}$ 为单位矩阵（所有元素为1）。例如，如果 $\mathbf{D}_2$ 如等式（10.45）所示，则由迭代关系——等式（10.48）可以得到：

$$\mathbf{D}_4 = \begin{bmatrix} 15 & 7 & 13 & 5 \\ 3 & 11 & 1 & 9 \\ 12 & 4 & 14 & 6 \\ 0 & 8 & 2 & 10 \end{bmatrix} \tag{10.49}$$

另一种将一个 $m \times n$ 点阵组成的图像映射至 $m \times n$ 像素的显示区域的方法是**误差分散**（error diffusion）。其中，一个给定像素单元处的输入光强度值与显示像素强度之间的误差，被分散至当前像素的右边或下方像素中。从扫描照片所得到的光强度值矩阵 **M** 开始，我们可以为屏幕上的一块区域组织一个像素强度值的数组 **I**。首先必须从左至右、从上向下扫描 **M** 的各列，并为 **M** 的每个元素确定最接近的像素强度等级。然后，根据以下的简化算法，将数组 **M** 中保存的值与每个像素单元中显示的强度等级之间的误差分散到 **M** 的邻近元素。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
for (j = 0; j < m; j++)
   for (k = 0; k < n; k++) {
      \*  Determine the available system intensity value
       *  that is closest to the value of M [j][k] and
       *  assign this value to I [j][k].
       */
      error = M [j][k] - I [j][k];
      I [j][k+1]   = M [j][k+1]   + alpha * error;
      I [j+1][k-1] = M [j+1][k-1] + beta  * error;
      I [j+1][k]   = M [j+1][k]   + gamma * error;
      I [j+1][k+1] = M [j+1][k+1] + delta * error;
   }
```

一旦将强度等级值赋给数组**I**中的元素，即可将数组映射至一个显示设备，如打印机或视频监视器。当然，不可能将误差分散至矩阵的最后一列（$k=n$）之后或最后一行（$j=m$）之下，而对于一个 2 级系统，可用的光强度等级为 0 和 1。可以选择参数来分散误差以满足以下关系：

$$\alpha+\beta+\gamma+\delta\leqslant 1 \tag{10.50}$$

一种可以产生较好结果的误差分散参数选择为$(\alpha, \beta, \gamma, \delta)=(7/16, 3/16, 5/16, 1/16)$。图 10.45 表示了使用这些参数值的误差分散。误差分散有时通过重复或反馈在图像中产生幻影，可能使图像中的某些部分，特别是面部特征（如头发和鼻子的轮廓）出现错误。这些错误可以通过为误差参数重新取值而使得总和小于 1，或者在分散误差之后再次变换矩阵来减少误差。一种变换的方法是将 **M** 中的所有元素乘以 0.8 再加 0.1。另一种提高图片质量的方法是改变扫描矩阵列“由右至左”和“从左到右”的顺序。

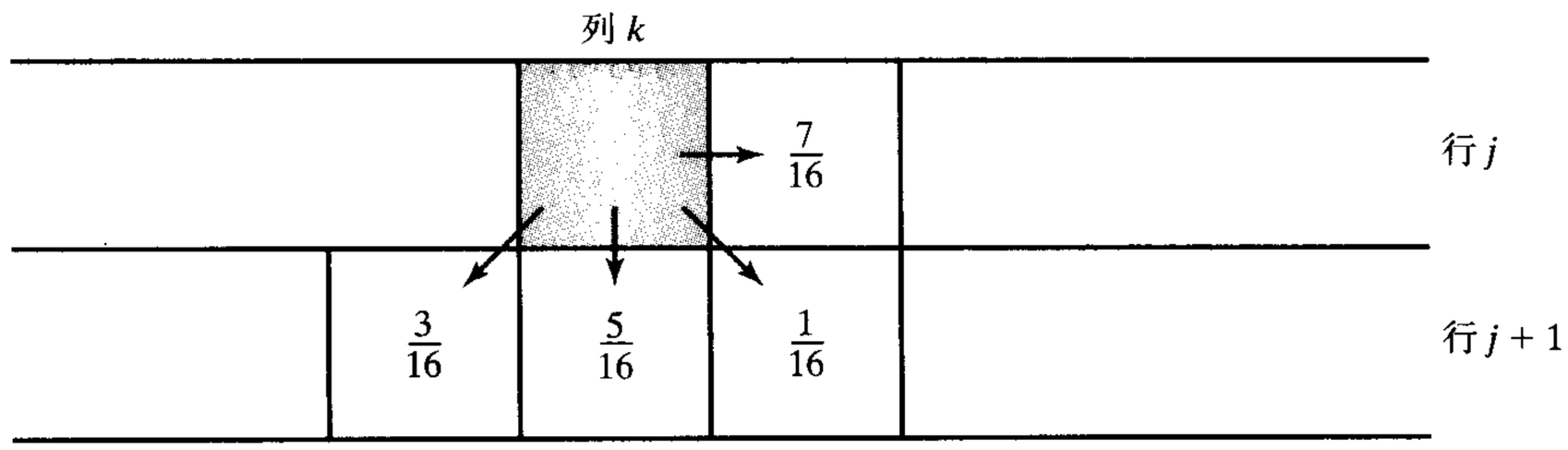


图 10.45　使用误差分散模式，可以将部分强度误差分散至相邻的像素单元

点分散（dot diffusion）是误差分散方法的一种变形。该方法将 $m \times n$ 的强度值矩阵分为 64 类，编号为 0 到 63，如图 10.46 所示。矩阵值与显示强度之间的误差将仅仅分布至类号更大的相邻矩阵的元素中。64 个类号进行分布的目标是，使那些被较低类号的元素完全包围的单元数目最小化，因为这种情况往往使周围元素的所有误差均集中到那个位置。

34	48	40	32	29	15	23	31
42	58	56	53	21	5	7	10
50	62	61	45	13	1	2	18
38	46	54	37	25	17	9	26
28	14	22	30	35	49	41	33
20	4	6	11	43	59	57	52
12	0	3	19	51	63	60	44
24	16	8	27	39	47	55	36

图 10.46　一种可能的分布模式，将强度矩阵分割成 64 个点分散类，由 0 至 63 进行编号

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.10 多边形绘制算法

光照模型中的强度计算可以通过多种方法应用于表面绘制。可将光照模型用于确定每一投影像素位置的表面强度，或将光照模型用于少量选定点且在其他表面位置近似计算强度。图形软件包一般使用扫描线算法实现表面绘制，通过仅处理多边形面及仅计算顶点处的表面强度来减少处理时间。然后在多边形面的其他位置插值。另一种更精确的多边形扫描绘制方法——光线跟踪算法则为曲面或平面计算每一投影表面点的强度。我们先考虑应用于多边形的扫描线表面绘制方法。然后在 10.11 节讨论可用于光线跟踪的方法。

10.10.1 恒定强度的明暗处理

最简单的绘制多边形面的方法是为其所有的投影点赋以相同的颜色。在这种情况下，我们使用光照模型为表面的一个位置，如多边形的一个顶点或中心点计算三个 RGB 颜色分量强度。这个方法称为**恒定强度表面绘制**（constant-intensity surface rendering）或**平面绘制**（flat surface rendering），为显示对象上的多边形面片提供了快速而简单的方法，可应用于快速生成一般曲面的大致外观，如图 10.50(b)所示。平面显示对希望快速标识模拟曲面的单个多边形面片的设计或其他应用也很有用。

通常，如果下列假定均成立，则多边形表面的平面明暗处理可以准确地绘制一个对象：

- 该多边形是一个多面体的一个面且不是一个近似表示曲面的多边形网格的一部分；
- 所有照明该多边形的光源离对象表面足够远，以使 $\mathbf{N}\cdot\mathbf{L}$ 与衰减函数对于对象表面是一个常数；
- 视点离该多边形足够远，以至于 $\mathbf{V}\cdot\mathbf{R}$ 对于多边形区域是一个常数。

即使所有的条件均不成立，如果对象的面片较小，则仍可以使用恒定强度表面绘制方法来合理地近似对象表面的光照效果。

10.10.2 Gouraud 明暗处理

这种由 Henri Gouraud 提出的方法称为 **Gouraud 表面绘制**（Gouraud surface rendering）或**强度插值表面绘制**（intensity-interpolation surface rendering）。这种方式通过在照明对象的表面上将光强度值进行线性插值来绘制该多边形表面。为绘制用多边形网格逼近的曲面而开发的Gouraud方法能使强度值沿相邻多边形的公共边均匀过渡。这种在多边形范围中的强度插值消除了在平面绘制中存在的光强度不连续的现象。

使用 Gouraud 表面绘制方法来绘制多边形表面时，需要进行下列计算：

(1) 确定每个多边形顶点处的平均单位法向量；
(2) 对于每个顶点根据光照模型来计算其光强度；
(3) 在多边形投影区域对顶点强度进行线性插值。

如图10.47所示，在多边形各顶点处，通过对共享该顶点的所有多边形表面的法向量取平均值，可以得到该点所对应的法向量。这样，对所有顶点 $\mathbf{V}$ 可以进行以下计算：

$$\mathbf{N}_V = \frac{\sum_{k=1}^{n}\mathbf{N}_k}{\left|\sum_{k=1}^{n}\mathbf{N}_k\right|} \tag{10.51}$$

一旦有了顶点法向量，就可以根据光照模型来确定该点的光强度值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

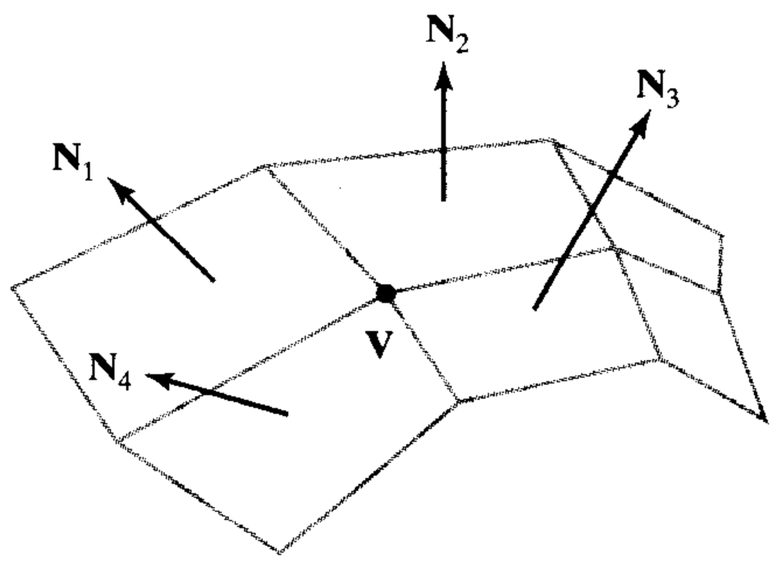


图 10.47　计算顶点 V 处的法向量时使用了共享该顶点的所有多边形表面法向量的平均值

计算出一个多边形面片所有顶点的强度以后，便可沿与该多边形投影区相交的扫描线位置对顶点值插值，如图 10.48 所示。对于每一条扫描线，它与多边形边交点处的强度可以根据边的两端点通过强度插值而得到。例如，在图 10.48 中，顶点 1、2 的多边形边与扫描线相交于点 4，一个快速获得点 4 处光强度的方法是，仅使用扫描线的垂直坐标在顶点 1 和 2 之间进行插值：

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2 \tag{10.52}$$

在这个表达式中，符号 I 表示一个 RGB 颜色分量的强度。同样，该扫描线的右交点（点 5）的光强度可以由顶点 2 与顶点 3 的强度值插值而得到。从这两个边界强度可通过线性插值来得到整个扫描线位置的像素强度。图 10.48 中点 **p** 的一个 RGB 分量的强度可以通过对点 4 和点 5 的边界强度进行插值而得到：

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5 \tag{10.53}$$

在 Gouraud 绘制的实现中，可以使用增量法有效地完成等式（10.52）和等式（10.53）中的强度计算。从与该多边形一个顶点相交的扫描线开始，可递增地获得与连接该顶点的一条边相交的其他扫描线的强度。假设该多边形面片是一个凸多边形，每一条与多边形相交的扫描线有两个边交点，如图 10.48 中的点 4 和 5。一旦获得了一条扫描线与两条边交点的强度，则可以使用增量法获得沿该扫描线的像素强度。

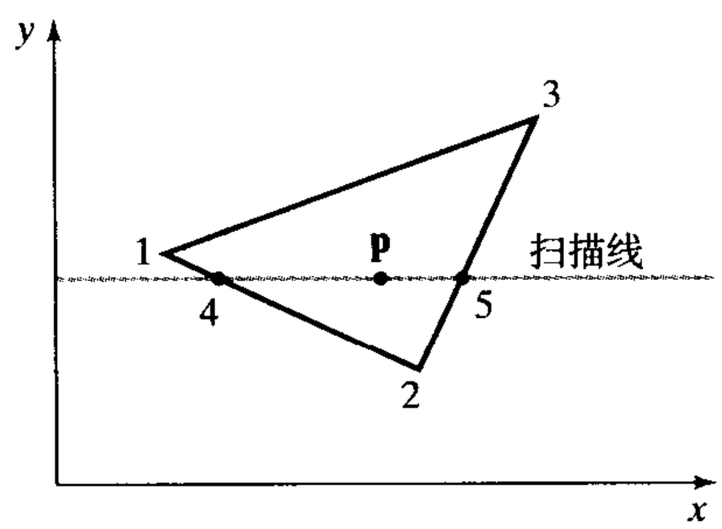


图 10.48　使用 Gouraud 表面绘制，点 4 处的强度由顶点 1 和 2 的强度进行线性插值得到，点 5 处的强度由顶点 2 和 3 的强度进行线性插值得到，然后从点 4 和 5 的强度继续线性插值得到内点 **p** 的强度值

作为强度增量计算的一个例子，考虑图 10.49 中的行 y 和 $y-1$，它们与多边形左边相交。如果扫描线 y 是强度值为 I_1 的顶点 y_1 下面相邻的扫描线，即 $y = y_1 - 1$，则可从公式计算扫描线 y 的强度 I（参见（10.52）式）如下：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$I = I_1 + \frac{I_2 - I_1}{y_1 - y_2}$$

继续沿该多边形的边向下，沿该边的下一条扫描线 $y-1$ 的光强度为

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2} \tag{10.55}$$

这样，沿该边向下的每一后继强度值可以简单地将常数项$(I_2 - I_1) / (y_1 - y_2)$加到前一强度值上而得。类似的计算可用来获得沿每一扫描线的后继水平像素的强度。

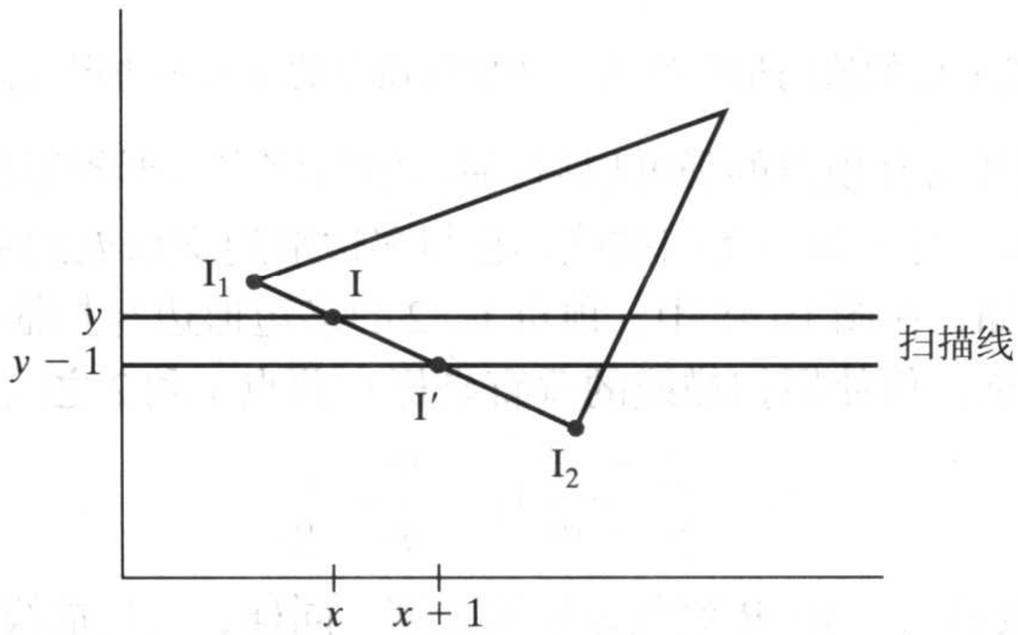
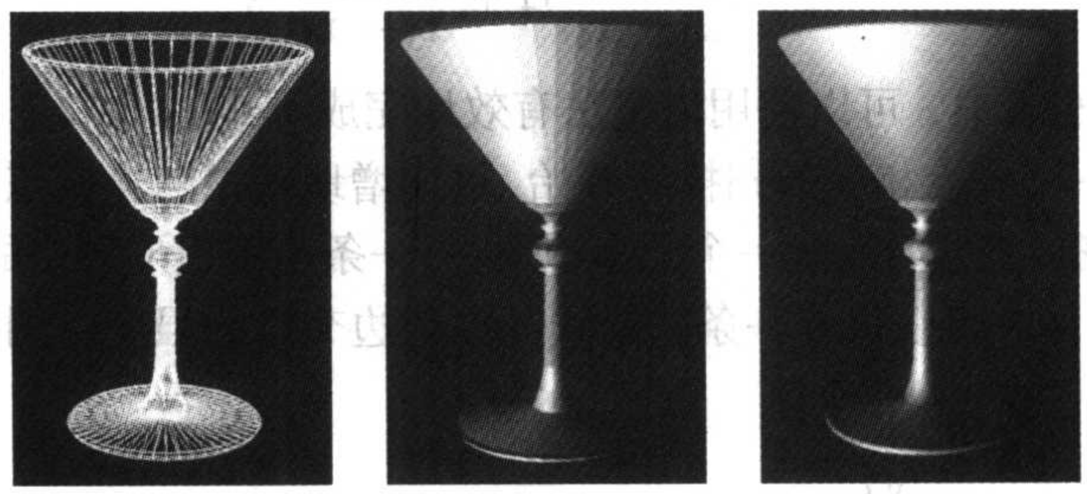


图 10.49 沿一条多边形边为后继扫描线进行光强度值的增量法插值计算

Gouraud表面绘制可以与隐藏面消除算法相结合，沿每条扫描线填充可见的多边形。图10.50(c)给出了使用Gouraud表面绘制来绘制对象的一个例子。



(a) (b) (c)

图 10.50 (a)一个对象的多边形网格近似；(b)平面明暗处理；(c)Gouraud 表面绘制

这种强度插值方法消除了平面明暗处理的不连续，但它仍有许多不足。表面上的高光有时会出现异常形状，线性光强度插值会造成表面上出现过亮或过暗的条纹，称为**马赫带**（Mach band）效应。可以将一个表面分割成许多的多边形面片或使用更精确的强度计算来消除这种现象。

10.10.3 Phong 明暗处理

多边形网绘制的一个更精确的方法随后由Phong Bui Tuong提出。该方法称为**Phong表面绘制**（Phong surface rendering）或**法向量插值绘制**（normal-vector interpolation rendering），对法向量进行插值来取代对强度插值。其结果是强度值的更精确计算，更真实的表面高光显示，以及更少的马赫带效应。然而，Phong 方法比 Gouraud 方法需要更多的计算。

按下列步骤可以实现 Phong 表面绘制方法对网格曲面每一多边形部分的绘制：

(1) 确定每个多边形顶点处的平均单位法向量；

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

(2) 在多边形投影区域上对顶点法向量进行线性插值；

(3) 根据光照模型，使用插值的法向量，沿每条扫描线计算投影像素的光强度。

在 Phong 方法中法向量的插值过程与 Gouraud 方法中的强度插值一样。图 10.51 中的法向量 **N** 在顶点 1 和 2 之间进行垂直插值：

$$\mathbf{N} = \frac{y - y_2}{y_1 - y_2}\mathbf{N}_1 + \frac{y_1 - y}{y_1 - y_2}\mathbf{N}_2 \tag{10.56}$$

使用同样的增量法，可以计算出后继扫描线和沿每条扫描线上后继像素位置的法向量：两种表面绘制方法的差别是我们现在必须在沿扫描线的每一个投影像素位置使用光照模型来获得表面强度值。

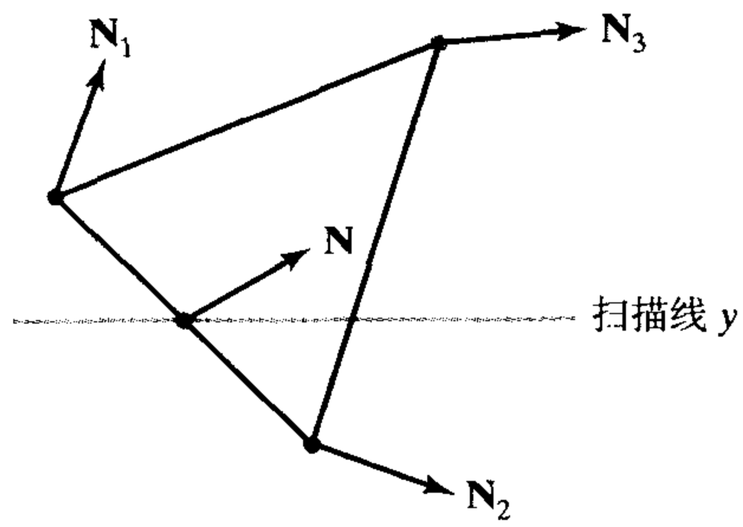


图 10.51　沿一多边形的边对表面法向量进行插值

10.10.4　快速 Phong 明暗处理

通过对光照模型进行近似计算的方法可减少Phong绘制方法的处理时间。**快速Phong表面绘制**（fast Phong surface rendering）利用截短的**泰勒扩展式**（Taylor-series expansion）和三角形面片来近似计算光强度。

在 Phong 明暗处理中，首先对各顶点法向量进行插值，可将三角形中任意点(x, y)处的表面法向量 **N** 表示为

$$\mathbf{N} = \mathbf{A}x + \mathbf{B}y + \mathbf{C} \tag{10.57}$$

其中，向量 **A**、**B**、**C** 由三个顶点方程确定：

$$\mathbf{N}_k = \mathbf{A}x_k + \mathbf{B}y_k + \mathbf{C}, \qquad k = 1, 2, 3 \tag{10.58}$$

(x_k, y_k)分别表示一个三角形顶点在像素平面的投影位置。

忽略反射率和衰减参数，可以使用下列等式表示表面上点(x, y)处的光源漫反射：

$$\begin{aligned} I_{\text{diff}}(x, y) &= \frac{\mathbf{L} \cdot \mathbf{N}}{|\mathbf{L}||\mathbf{N}|} \\ &= \frac{\mathbf{L} \cdot (\mathbf{A}x + \mathbf{B}y + \mathbf{C})}{|\mathbf{L}||\mathbf{A}x + \mathbf{B}y + \mathbf{C}|} \\ &= \frac{(\mathbf{L} \cdot \mathbf{A})x + (\mathbf{L} \cdot \mathbf{B})y + \mathbf{L} \cdot \mathbf{C}}{|\mathbf{L}||\mathbf{A}x + \mathbf{B}y + \mathbf{C}|} \end{aligned} \tag{10.59}$$

该表达式可改写为

$$I_{\text{diff}}(x, y) = \frac{ax + by + c}{[dx^2 + exy + fy^2 + gx + hy + i]^{1/2}} \tag{10.60}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

其中，参数 a、b、c、d 等用于表示各种点积，例如：

$$a = \frac{\mathbf{L} \cdot \mathbf{A}}{|\mathbf{L}|} \tag{10.61}$$

最后，可以将等式（10.60）的分母表示为泰勒扩展式，并保持各项在 x、y 上达到二次，这使得

$$I_{\text{diff}}(x, y) = T_5 x^2 + T_4 xy + T_3 y^2 + T_2 x + T_1 y + T_0 \tag{10.62}$$

其中，每个 T_k 为等式（10.60）中参数 a、b、c 等的一个函数。

利用向前差分，一旦确定了初始向前差分参数，即可根据等式（10.62）进行计算，它对每个像素(x, y)仅需两次加法的计算量。虽然快速Phong方法减少了Phong表面绘制的计算量，但它仍需相当于Gouraud表面绘制2倍的时间。基本Phong方法利用向前差分则需比Gouraud绘制多耗时6至7倍。

针对漫反射的快速Phong明暗处理可以将其扩展到考虑镜面反射，类似的计算将用于考察基本光照模型中的一些特殊项，例如$(\mathbf{N} \cdot \mathbf{H})^{n_s}$。另外，我们还可以开发出包含任意多边形和有限视点的算法。

10.11 光线跟踪方法

在8.20节中，我们介绍了光线投射的概念，它用来在结构实体几何法中确定沿从一像素位置出发的射线到表面的交点。另外，我们还讨论了利用光线投射作为确定场景中可见面的一种方法（参见9.10节）。**光线跟踪**（ray tracing）是光线投射思想的延伸，它并不仅仅为每个像素寻找可见面（如图10.52所示），该方法还跟踪光线在场景中的反射和折射，并计算它们对总的光强度的作用。这为获得全局反射和折射效果提供了一种简单有效的绘制方式。基本光线跟踪算法为可见面判别、明暗效果、透明及多光源照明等提供了实现方法。人们在此基础上为了生成真实感图形而进行了大量的研究工作。光线跟踪技术虽然能够生成高度真实感的图形，特别是对于表面光滑的对象，但它所需的计算量却大得惊人。图10.53显示了光线跟踪所能生成的全局反射和透射效果。

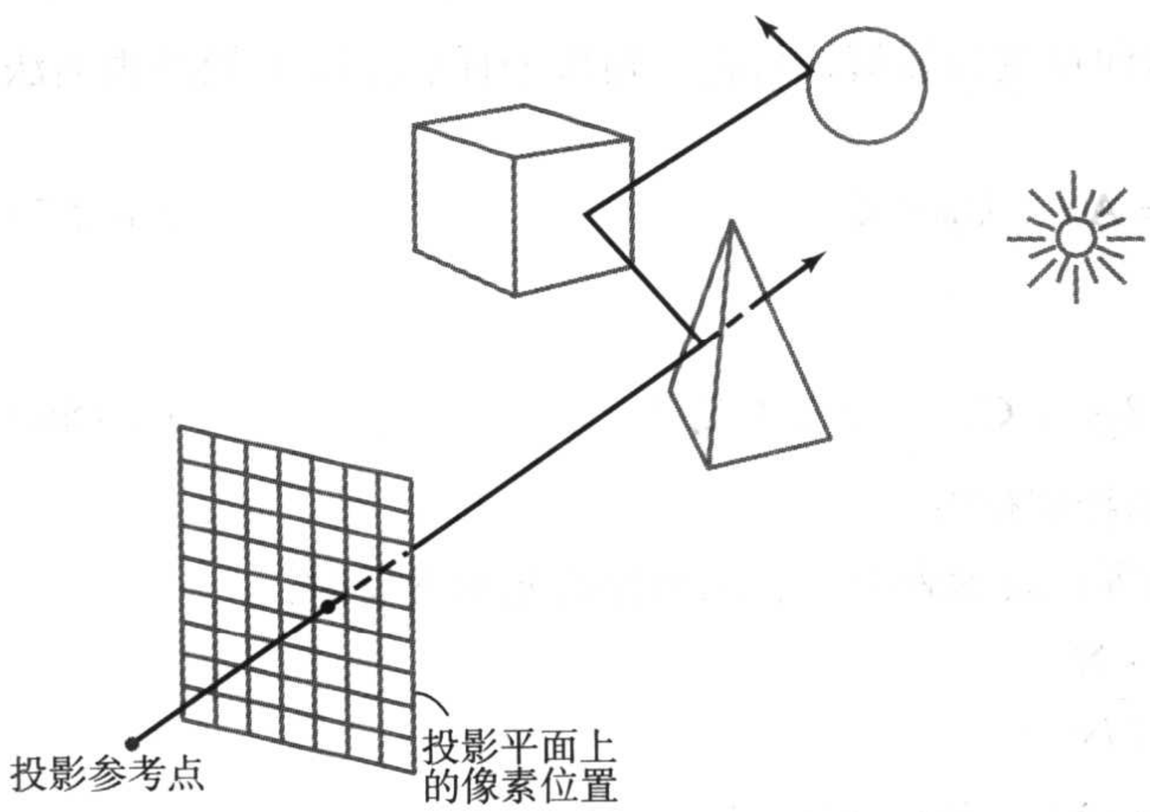


图10.52 由投影参考点出发跟踪一束光线，光线穿过一像素单元进入包含多个对象的场景，然后经过多次反射和透射

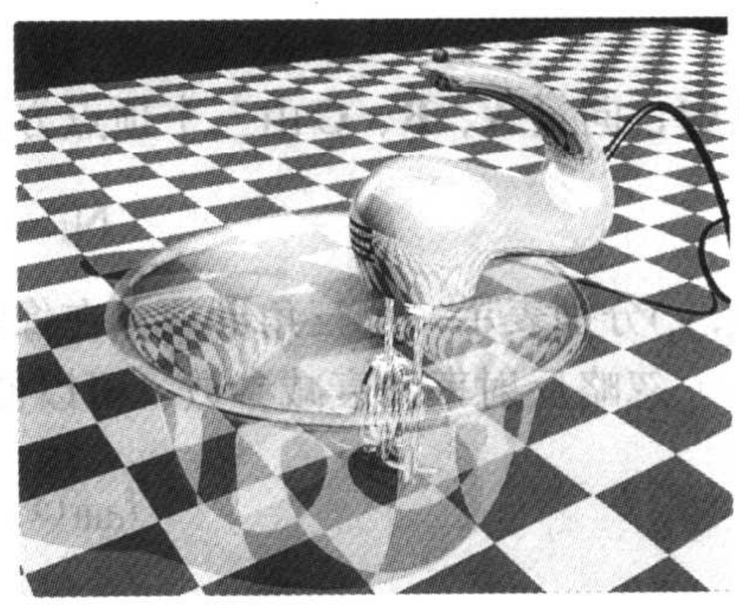


图10.53 一个被光线跟踪的场景，显示由物体表面生成的全局反射和透射光照效果（Evans & Sutherland 提供）

10.11.1 基本光线跟踪算法

首先，为光线跟踪算法建立一个投影参考点在 z 轴、像素位置在 xy 平面的坐标系统，如图10.54所示。然后，在该坐标系统中描述场景的几何数据并生成像素光线。对于场景的透视投影视图，每

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

一光线从投影参考点（投影中心）出发，穿过每个像素中心进入场景并沿反射和透射路径形成各种光线分支。然后在该表面交点计算像素强度的贡献。这一绘制方法建立在几何光学原理的基础上。场景中该表面的光线在各个方向发散，其中有的到达投影平面的像素位置。由于有无穷多数量的光线散射，我们通过从像素到场景反向跟踪光线路径来确定特定像素的强度贡献。在基本的光线跟踪算法中，为每一像素生成一条逆向光线，这和通过针孔照相机观察场景相近似。

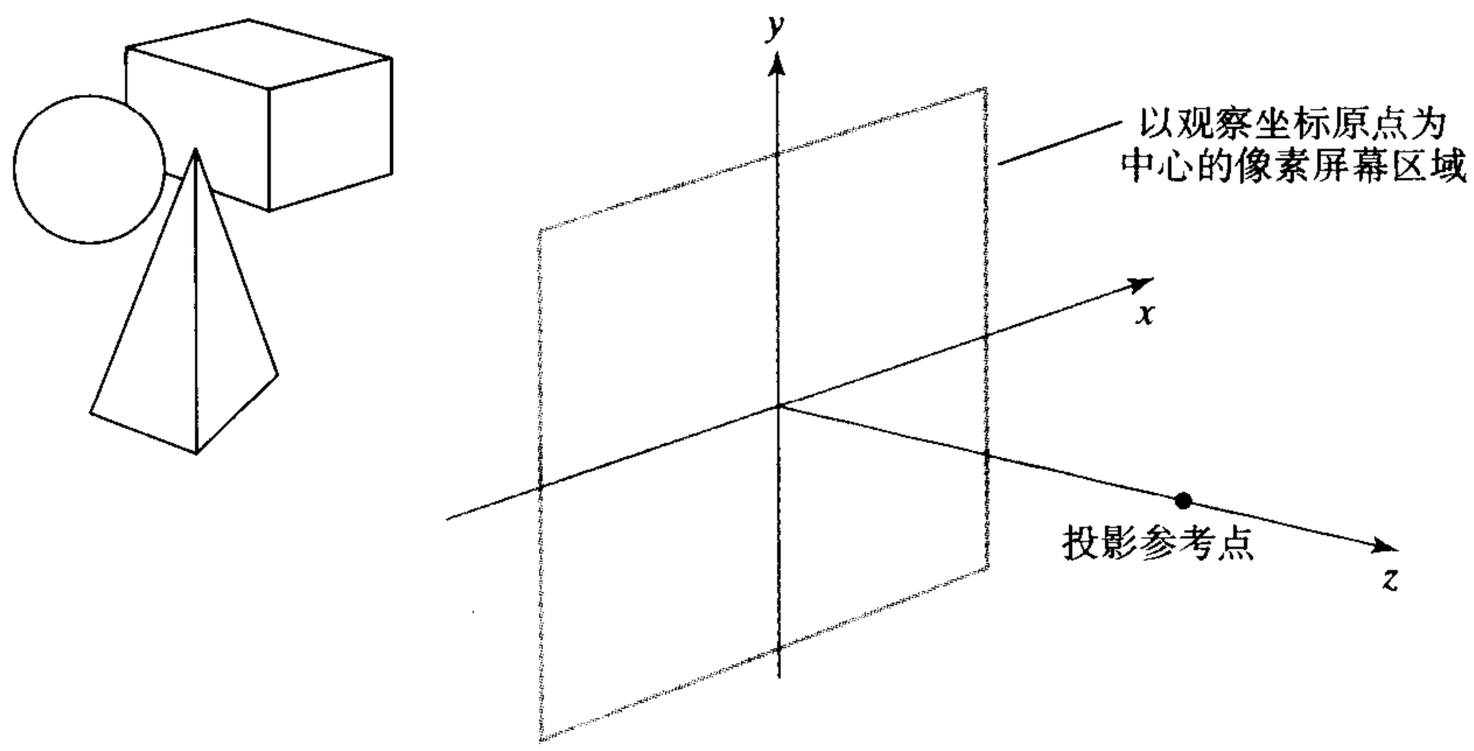


图 10.54　光线跟踪的坐标参照框架

生成每个像素光线后，需测试场景中的所有场景表面以确定其是否与该光线相交。如果该光线确实与一个表面相交，则计算出交点离像素的距离。在计算出所有表面和该光线的交点后，具有最小距离的交点即可代表该像素所对应的可见面。然后，将该光线在该可见面上沿镜面反射路径（反射角等于入射角）反射。若该表面是透明的，还需考察透过该表面的折射光线。我们将反射和折射光线统称为**从属光线**（secondary ray）。

接着对每条从属光线重复这个光线处理过程。如果有表面和其相交则确定最近的相交表面，然后，递归地在沿从属光线方向最近的对象表面上生成下一条折射和反射光线。当从每个像素出发的光线在场景中被反射和折射时，逐个将相交表面加入到一个**二叉光线跟踪树**（ray-tracing tree）中，如图 10.55 所示。树的左分支表示反射光线，右分支表示透射光线。光线跟踪的最大深度可由用户选定，或由存储容量决定。当下面任一条件满足时，就停止跟踪。

- 该光线不和任一表面相交。
- 该光线与一个光源相交且该光源不是一个反射面。
- 该树到达最大允许深度。

在每一个表面交点，引入基本光照模型来确定表面强度贡献。该强度值存放在像素树的表面节点位置。与非反射面光源相交的光线可用该光源对其强度赋值，尽管基本光线跟踪算法中的光源一般是位于场景坐标范围之外的点光源。图 10.56 给出了一个与光线相交的表面和用于反射光强度计算的单位法向量。单位法向量 **u** 是光线路径的方向，**N** 是单位表面法向量，**R** 是单位反射向量，**L** 是指向点光源的单位向量，而 **H** 是 **L** 和 **V** 的半角向量。对于光线跟踪计算，观察方向是 $\mathbf{V} = -\mathbf{u}$。沿 **L** 的光线称为**阴影光线**（shadow ray）。若它在表面和点光源之间与任何对象相交，则该表面位于点光源的阴影中。对象表面环境光强度为 $k_a I_a$，漫反射光与 $k_d(\mathbf{N} \cdot \mathbf{L})$ 成正比，镜面反射与 $k_s(\mathbf{H} \cdot \mathbf{N})^{n_s}$ 成正比。正如 10.3 节所述，从属光线 **R** 的镜面反射取决于对象表面法向量和入射光线的方向：

$$\mathbf{R} = \mathbf{u} - (2\mathbf{u} \cdot \mathbf{N})\mathbf{N} \tag{10.63}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

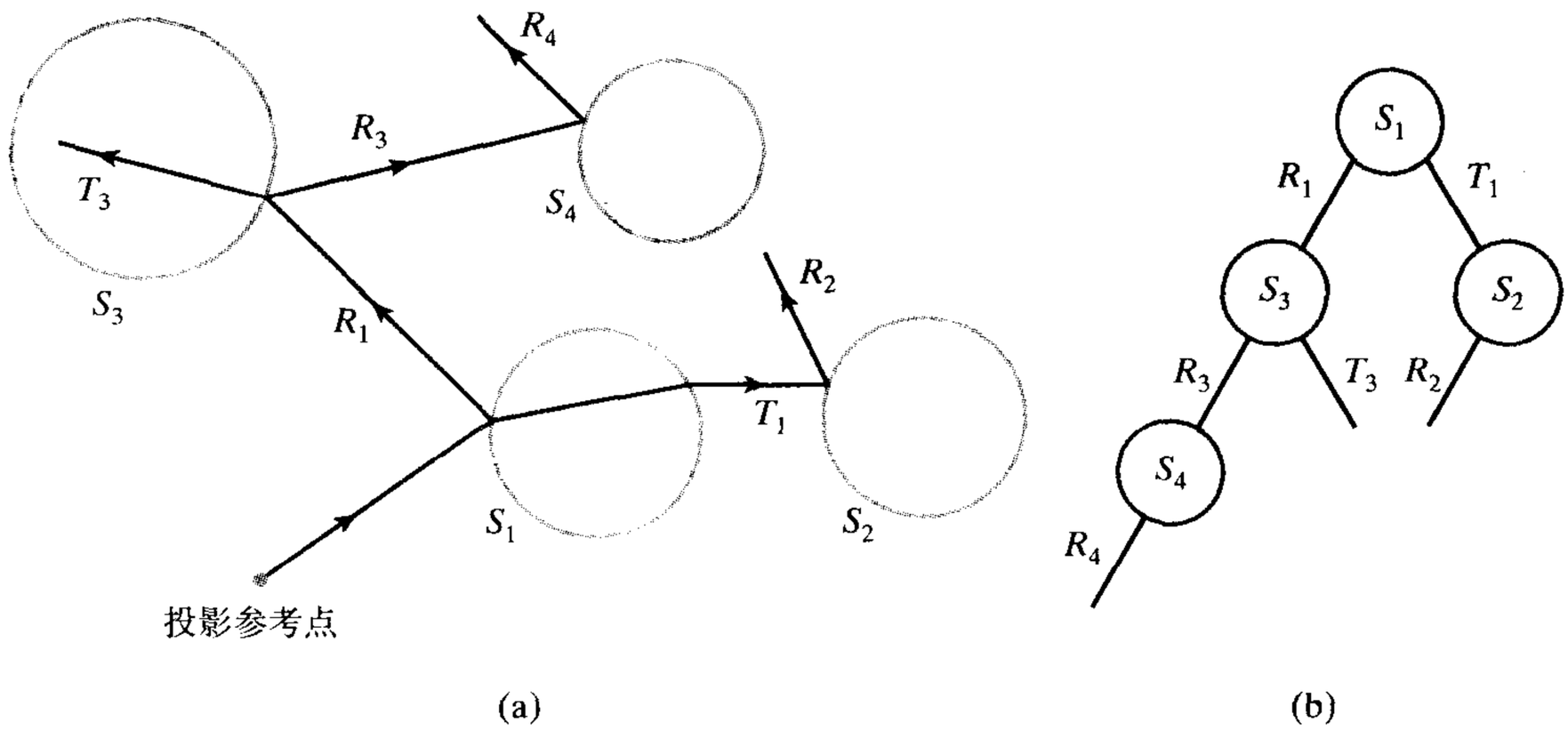


图 10.55 (a)由屏幕像素出发穿过场景的反射和折射光线；(b)二叉光线跟踪树

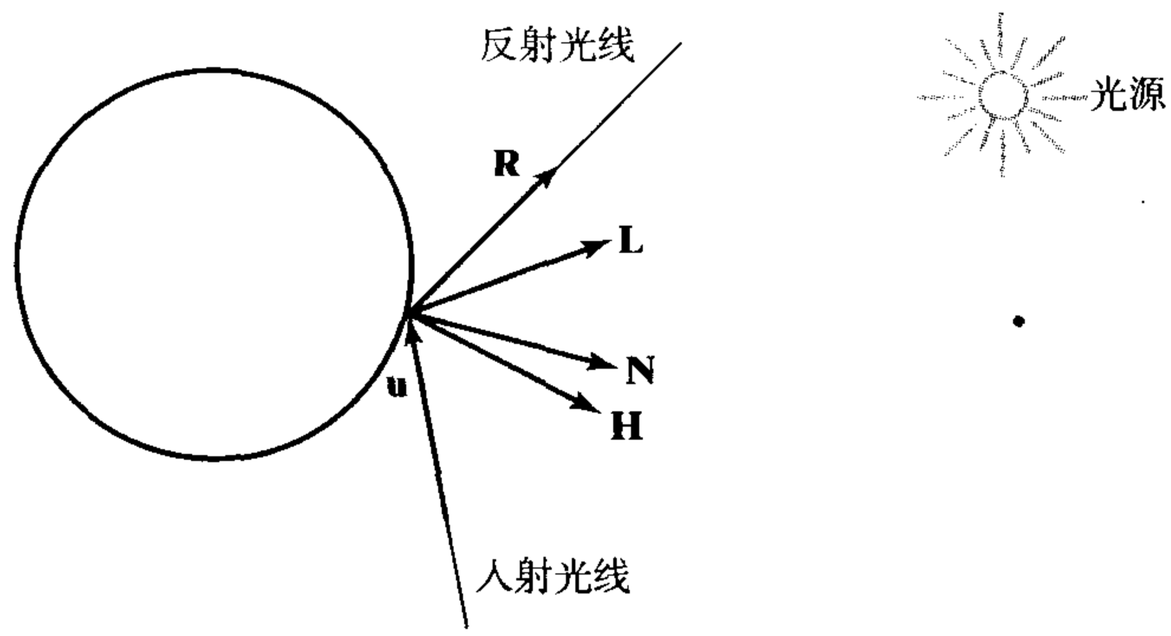


图 10.56 与入射光线 **u** 相交的对象表面的单位向量

对于一个透明表面，还需考察穿过对象的透射折射光线对总的光强度的作用。可以沿图 10.57 中的透射方向 **T** 跟踪从属光线，以确定其作用值。单位透射向量 **T** 则可以由向量 **u** 与 **N** 得到：

$$\mathbf{T} = \frac{\eta_i}{\eta_r}\mathbf{u} - \left(\cos\theta_r - \frac{\eta_i}{\eta_r}\cos\theta_i\right)\mathbf{N} \tag{10.64}$$

参数 η_i 和 η_r 分别为入射材质和折射材质的折射率，折射角 θ_r 可由 Snell 定律计算出来：

$$\cos\theta_r = \sqrt{1 - \left(\frac{\eta_i}{\eta_r}\right)^2 (1 - \cos^2\theta_i)} \tag{10.65}$$

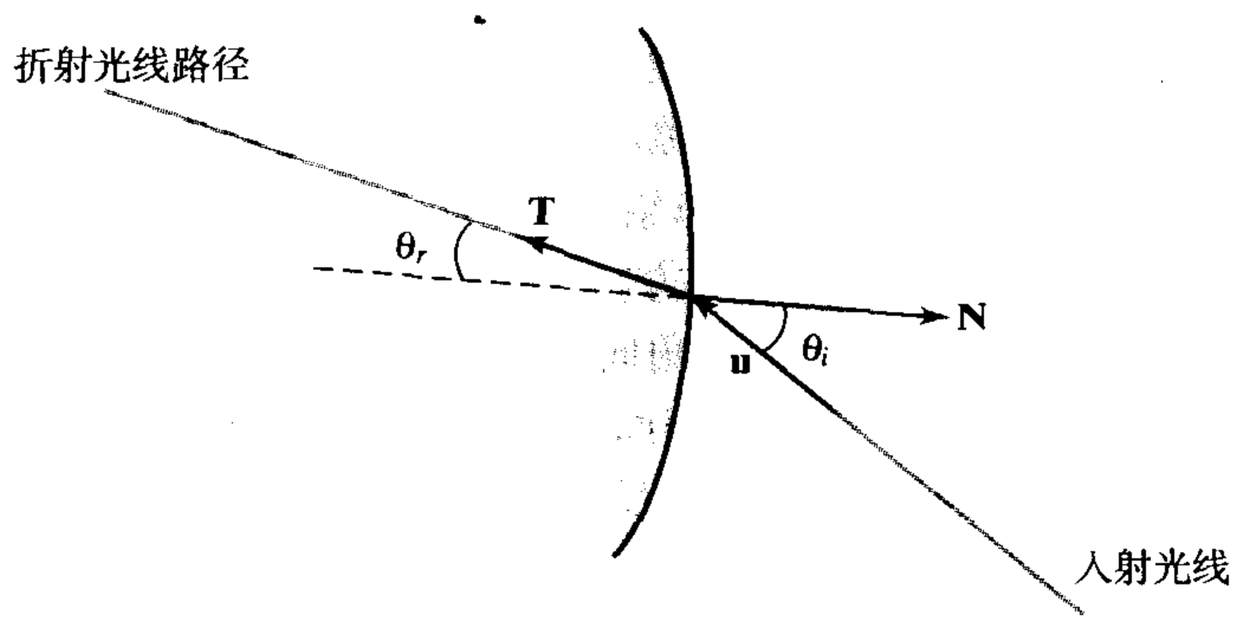


图 10.57 穿过一个透明对象的折射光线路径

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

一个像素的二叉树建立完毕后，从树的末端（终结节点）开始累计强度贡献。树的每一节点的表面强度因离开父节点（相邻上一节点）表面的距离而衰减并加入到父节点表面的强度中。赋给像素的强度是该光线树根节点的衰减后的强度总和。如果一像素的初始光线与场景中任一对象均不相交，则其光线树为空且用背景光强度对其赋值。

10.11.2　光线与对象表面的求交计算

如图 10.58 所示，一束光线可以由初始位置 $\mathbf{P}_0$ 和单位向量 $\mathbf{u}$ 来描述。沿光束方向离 $\mathbf{P}_0$ 距离为 s 的任意点 $\mathbf{P}$ 的坐标，可以由**光线方程**（ray equation）表示为

$$\mathbf{P} = \mathbf{P}_0 + s\mathbf{u} \tag{10.66}$$

最初，$\mathbf{P}_0$ 可设置为投影平面上的某像素点 $\mathbf{P}_{\text{pix}}$，或作为投影参考点。初始单位向量 $\mathbf{u}$ 则可以由投影参考点和该光线穿过的像素位置来得到：

$$\mathbf{u} = \frac{\mathbf{P}_{\text{pix}} - \mathbf{P}_{\text{prp}}}{|\mathbf{P}_{\text{pix}} - \mathbf{P}_{\text{prp}}|} \tag{10.67}$$

尽管 $\mathbf{u}$ 没有必要是一个单位向量，但成为一个单位向量可简化某些计算。

为了计算表面交点，可以使用表面方程求解 $\mathbf{P}$ 的位置，如方程（10.66）所示。这给出了参数 s 的值，即从 $\mathbf{P}_0$ 到光线与表面交点的距离。

每次与表面相交时，向量 $\mathbf{P}_0$ 和 $\mathbf{u}$ 由交点处的从属光线来更新。对于从属光线，$\mathbf{u}$ 的反射方向为 $\mathbf{R}$、透射方向为 $\mathbf{T}$。找出一个从属光线与表面的交点，求解光线方程和表面方程的联立方程组以获得交点坐标。接着更新二叉树并生成下一组反射和折射光线。

对多数可能出现的形状包括样条表面，都已经开发出高效的光线表面求交算法。一般的做法是联立光线方程和描述表面的方程，求解参数 s。多数情况下，使用数值求根方法和增量计算确定表面上的交点位置。对于复杂的对象，常将光线方程转换到定义对象的局部坐标系中。通过将对象变换成更适当的形状可简化许多情况。例如，通过转换光线和表面方程使光线跟踪椭球的问题变成光线与球面相交的问题。图 10.59 给出了一个包含多个对象的光线跟踪场景。

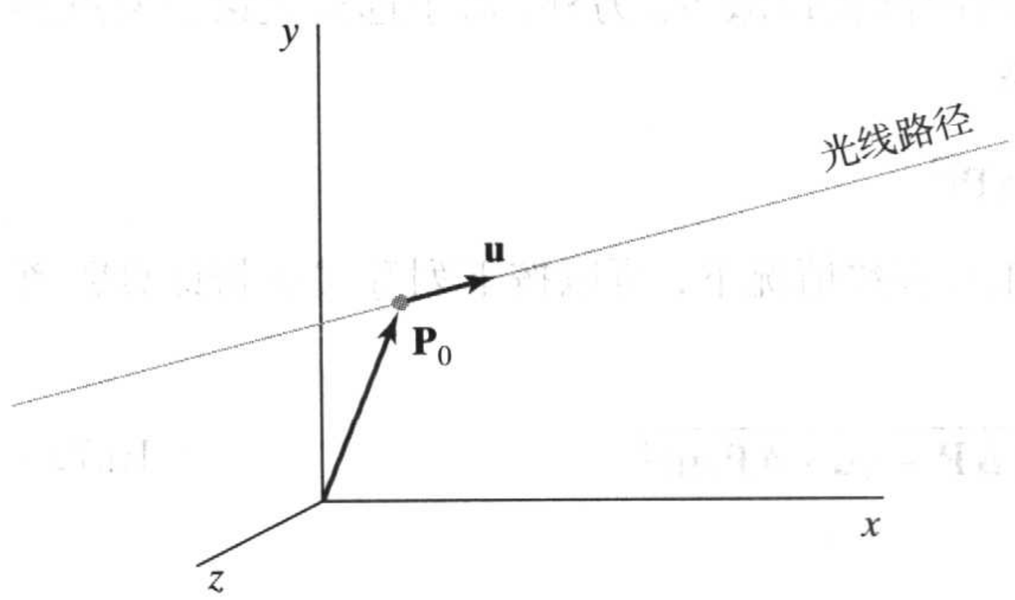


图 10.58　初始位置为 $\mathbf{P}_0$、单位方向向量为 $\mathbf{u}$ 的光线

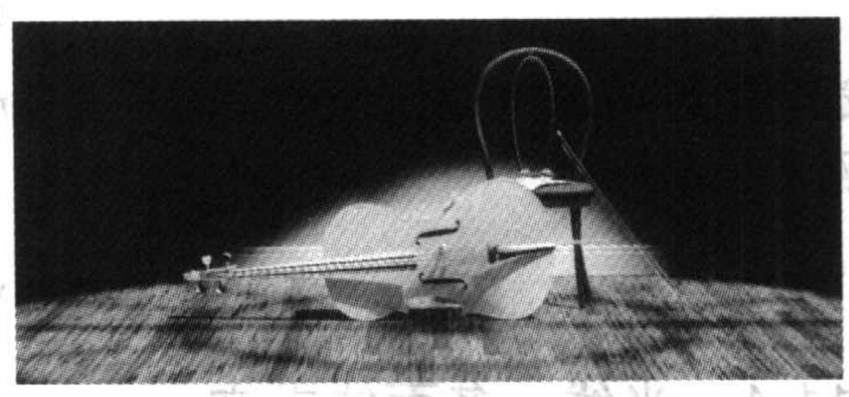


图 10.59　一个展示表面纹理全局反射的光线跟踪场景

10.11.3　光线－球面求交

光线跟踪中最简单的对象为球体。给定一个半径为 r、中心为 $\mathbf{P}_c$ 的球体（参见图 10.60），球面上任意点 $\mathbf{P}$ 均满足球面方程：

$$|\mathbf{P} - \mathbf{P}_c|^2 - r^2 = 0 \tag{10.68}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

将 **P** 的光线方程（10.66）代入上一方程，得到

$$|\mathbf{P}_0 - s\mathbf{u} - \mathbf{P}_c|^2 - r^2 = 0 \tag{10.69}$$

令 $\Delta\mathbf{P} = \mathbf{P}_c - \mathbf{P}_0$，并扩展点积，可得二次等式：

$$s^2 - 2(\mathbf{u} \cdot \Delta\mathbf{P})s + (|\Delta\mathbf{P}|^2 - r^2) = 0 \tag{10.70}$$

求解得

$$s = \mathbf{u} \cdot \Delta\mathbf{P} \pm \sqrt{(\mathbf{u} \cdot \Delta\mathbf{P})^2 - |\Delta\mathbf{P}|^2 + r^2} \tag{10.71}$$

若根为负，则光线与球面不相交或球面在 $\mathbf{P}_0$ 之后。对于这两种情况，都可不再考虑该球面，因为我们假定场景在投影平面的前面。当根不为负时使用等式（10.71）的两个根中较小的一个从光线方程（10.66）获得表面交点。图 10.61 给出了一个包含用光亮的球面形成的雪花图案的光线跟踪场景，展示了用光线跟踪生成的表面全局反射。

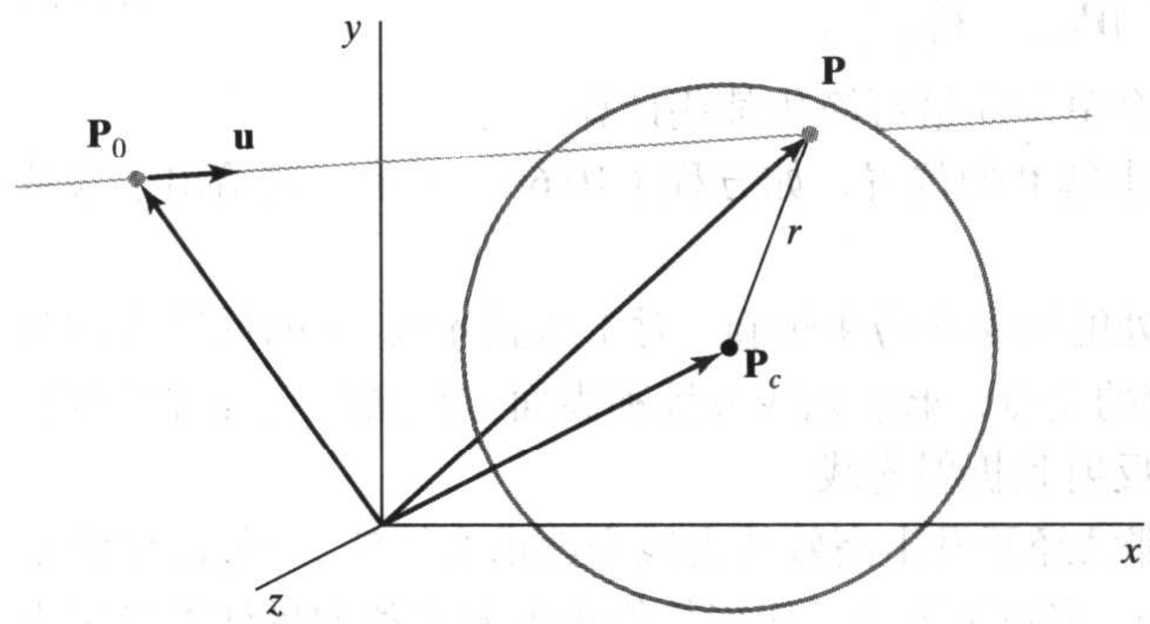


图 10.60 光线与半径为 r、中心为 $\mathbf{P}_c$ 的球面求交

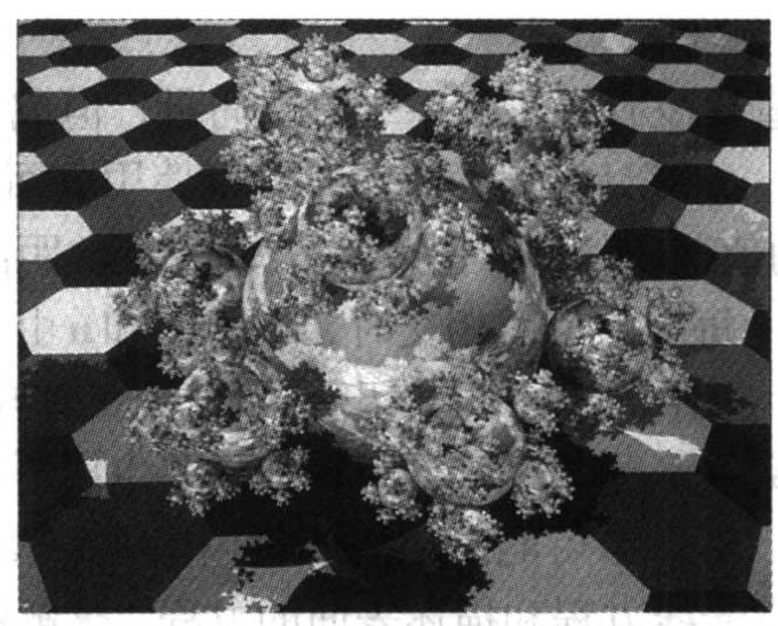


图 10.61 由 7381 个球面和 3 个光源绘制而成的 Sphereflake（Eric Haines 3D/EYE 公司提供）

光线–球面求交计算还可以进行一些优化使处理时间得以减少。另外，对于远离光束出发点的小球体，等式（10.71）易于出现取整误差。即如果

$$r^2 \ll |\Delta\mathbf{P}|^2$$

则我们可能在 $|\Delta\mathbf{P}|^2$ 的近似计算过程中丢失 r^2 项。在大多数情况下，可以按下列等式重新计算距离 s 以清除该误差：

$$s = \mathbf{u} \cdot \Delta\mathbf{P} \pm \sqrt{r^2 - |\Delta\mathbf{P} - (\mathbf{u} \cdot \Delta\mathbf{P})\mathbf{u}|^2} \tag{10.72}$$

10.11.4 光线 – 多面体求交

多面体的求交计算与球体的求交计算相比要复杂得多。因此，在一开始先利用包围体进行求交测试可提高效率。例如，图 10.62 表示一个被球体包围的多面体，若光束与球面无交点，则无需再对多面体进行测试。若光线与球面相交，则只需测试不等式：

$$\mathbf{u} \cdot \mathbf{N} < 0 \tag{10.73}$$

可以找到多面体的前表面。不等式中，**N** 为多面体的表面法向量。对于多面体中每个满足不等式（10.73）的表面，还须对表面上满足光线方程（10.66）的点 **P** 求解平面方程：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{N} \cdot \mathbf{P} = -D \tag{10.74}$$

其中，$\mathbf{N} = (A, B, C)$，D 为平面方程的第四个参数，如果

$$\mathbf{N} \cdot (\mathbf{P}_0 + s\mathbf{u}) = -D \tag{10.75}$$

则点 $\mathbf{P}$ 同时位于平面和光线上，而从初始光线位置到平面的距离为

$$s = -\frac{D + \mathbf{N} \cdot \mathbf{P}_0}{\mathbf{N} \cdot \mathbf{u}} \tag{10.76}$$

这给出该多边形一个表面所在的无穷平面上的一个点，但该点可能不在多边形边界内（参见图10.63），因此，还需通过“内外”测试法（参见3.15节）来确定光线是否与多面体的该表面相交。逐个对满足不等式（10.73）的表面进行测试，由距相交的多边形的最小距离 s 可确定出多面体表面的相交位置。若由等式（10.76）所得的交点均非内点，则该光线与多面体不相交。

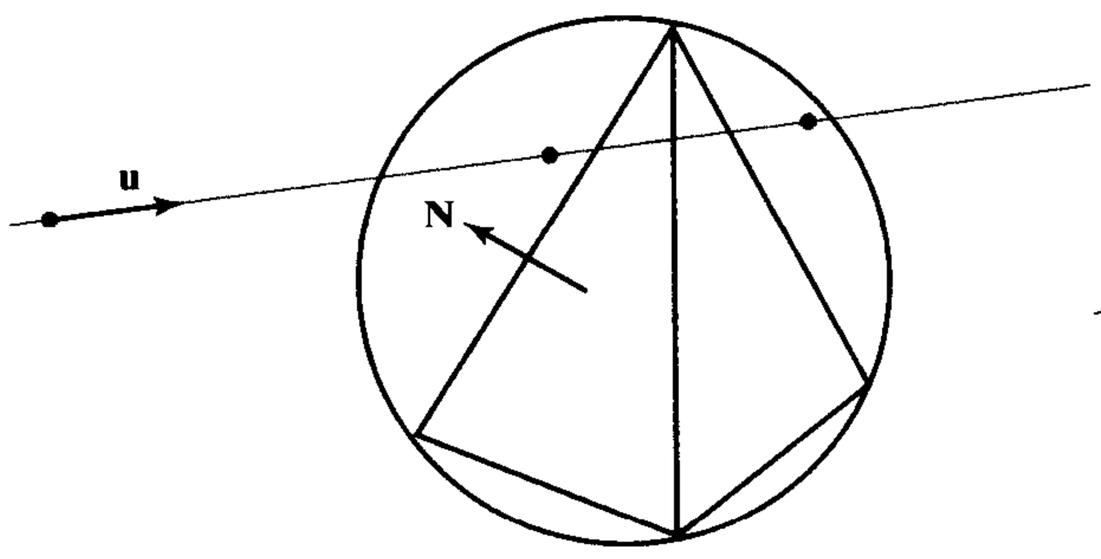


图 10.62　被一个包围球包围的多面体

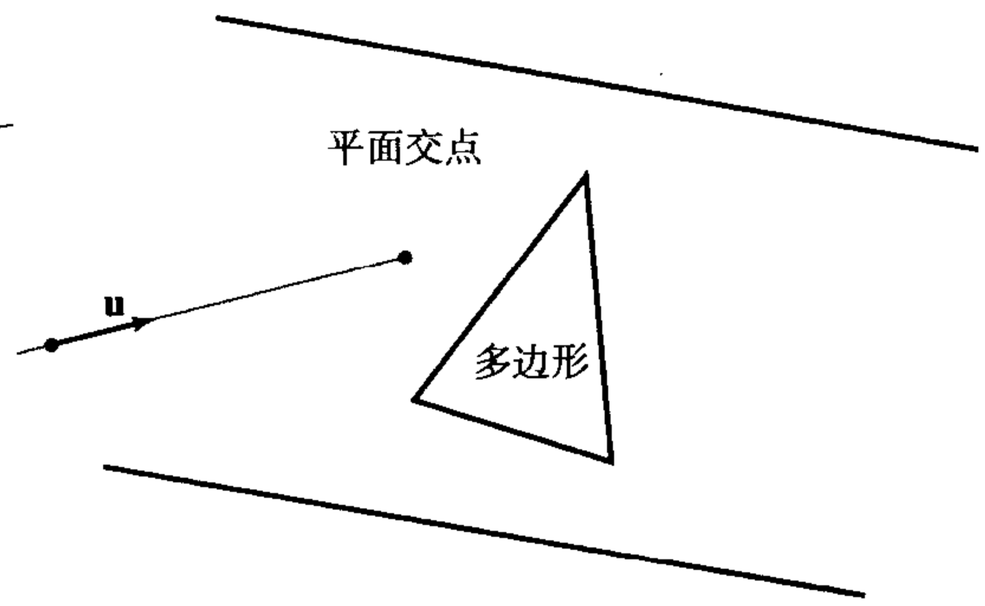


图 10.63　光线与一个多边形平面求交

10.11.5　减少对象求交计算量

在光线跟踪过程中，约有 95% 的时间用于光线与表面的求交计算。对于一个有多个对象的场景，大部分处理时间用于计算沿光束方向不可见的对象的求交。因此，人们开发出许多方法来减少在这些求交计算上所花费的时间。

减少求交计算量的一种方法是，将相邻对象用一个包围体（盒或球）包起来（参见图 10.64）。然后测试光线与包围体的交点。如果没有交点，则无需对被包围对象进行求交测试。这种方法可以扩充到利用包围体的层次结构。即将几个包围体包在一个更大的包围体中，以便层次式地进行求交测试。首先测试最外层的包围体，然后根据需要，逐个测试各层的包围体，依次类推。

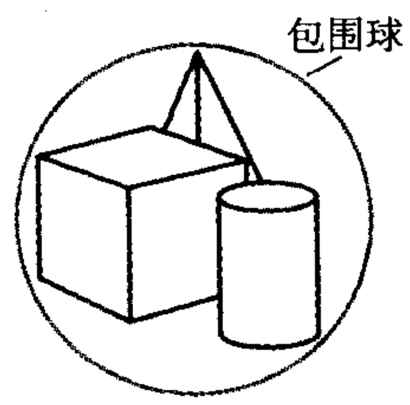


图 10.64　被一个包围球包围的一组对象

10.11.6　空间分割方法

另一种减少求交计算量的方法是**空间分割**（space-subdivision）技术。我们可以将整个场景包含在一个立方体中，然后将立方体逐次分割。直至每个子立方体（体元）所包含的对象表面或表面

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

数目小于等于一个预定的最大值。例如，可以要求每个体元中至多只包含一个表面。如果采用并行和向量处理技术，每个体元所包含的最大表面数目可以由向量寄存器的大小和处理器个数来决定。立方体的空间分割单元可以用一棵八叉树或一棵二叉分割树进行存储。另外，可以进行均匀分割（uniform subdivision），即每次将立方体分割为 8 个相同大小的体元。也可以采用适应性细分（adaptive subdivision），即仅对包含对象的立方体区域进行分割。

然后我们跟踪穿过立方体中体元的光线，仅需对包含表面的单元执行求交测试，光线所交的第一个对象表面即为可见面。另外，我们必须在体元大小和每个体元所含的表面数目之间进行取舍，如果每个体元中的最大表面数目定得过小，则体元体积也将过小，使得在求交测试中所节省的大部分时间都耗费在光线贯穿体元的处理中。

图10.65表示一束像素光线与包围场景的立方体的前表面的求交。该立方体前表面上的交点位置标识由该光线遍历的初始单元。然后，沿光线确定其贯穿体元时在每个单元的入口与出口位置（参见图 10.66）。在每一非空单元中测试相交表面。该过程继续到找到一个相交的对象表面，或光线射出包围场景的立方体。

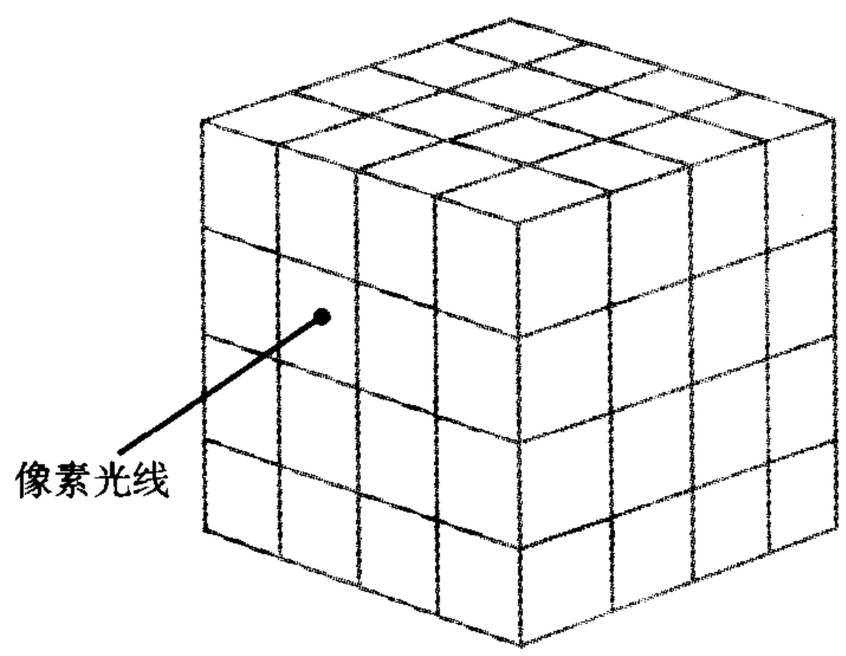


图 10.65　光线与包围场景中所有对象的立方体求交

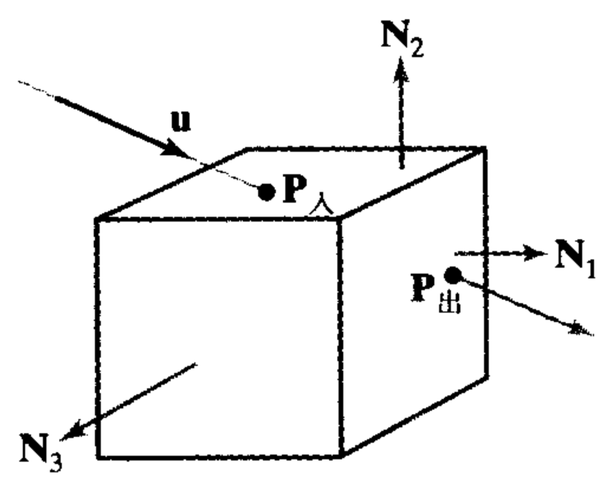


图 10.66　光线贯穿包围场景的立方体单元

给定一束光线的方向向量 **u** 和某单元的光线入口位置 $\mathbf{P}_{in}$，则潜在的出口表面一定满足：

$$\mathbf{u} \cdot \mathbf{N}_k > 0 \tag{10.77}$$

这里 $\mathbf{N}_k$ 表示该单元的面 k 的单位表面法向量。若图 10.66 中单元表面的法向量与笛卡儿坐标轴对齐，则

$$\mathbf{N}_k = \begin{cases} (\pm 1, 0, 0) \\ (0, \pm 1, 0) \\ (0, 0, \pm 1) \end{cases} \tag{10.78}$$

只须检查 **u** 中各分量的符号，就可确定出三个候选的出口表面。可由光线方程得到三个表面上的出口位置：

$$\mathbf{P}_{\text{out},k} = \mathbf{P}_{\text{in}} + s_k \mathbf{u} \tag{10.79}$$

其中，s_k 为沿光线从 $\mathbf{P}_{in}$ 至 $\mathbf{P}_{out,k}$ 的距离。对于各个表面，将光线方程代入平面方程：

$$\mathbf{N}_k \cdot \mathbf{P}_{\text{out},k} = -D_k \tag{10.80}$$

可以对候选出口表面求解光线距离：

$$s_k = \frac{-D_k - \mathbf{N}_k \cdot \mathbf{P}_{\text{in}}}{\mathbf{N}_k \cdot \mathbf{u}} \tag{10.81}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

计算出的最小的 s_k 标识该单元的出口表面。若该单元的表面与笛卡儿坐标平面平行，法向量 $\mathbf{N}_k$ 为单位坐标轴向量（10.78），则可以简化等式（10.81）的计算。例如，如果一个候选表面法向量为(1，0，0)，则对该表面有

$$s_k = \frac{x_k - x_0}{u_x} \tag{10.82}$$

其中 $\mathbf{u} = (u_x, u_y, u_z)$，且 $x_k = -D_k$ 为候选出口表面的坐标位置，而 x_0 是该单元入口表面的坐标位置。

我们可以对单元遍历过程进行修改以加速处理。一种方法是将与 $\mathbf{u}$ 中最大分量相垂直的表面作为待定出口表面 k。该待定出口表面随后分成多个部分，如图 10.67 中的例子所示。根据表面上包含 $\mathbf{P}_{\text{out},k}$ 的分区，可以确定出真正的出口表面。例如，若交点 $\mathbf{P}_{\text{out},k}$ 在图 10.67 中示例的区域 0 内，则待定表面即为真正的出口表面且我们的工作结束。若交点在区域 1 内，则真正的出口表面为上表面，只需在该单元的上表面计算出口点。同样，区域 3 表示下表面为真正的出口表面，区域 4 和 2 分别表示真正的出口表面为左或右边界表面；当待定出口表面落在区域 5、6、7、8 时，则还需执行两个附加求交计算，以确定出口表面。如果将这种方法实现于并行向量机之上，则可以进一步提高处理速度。

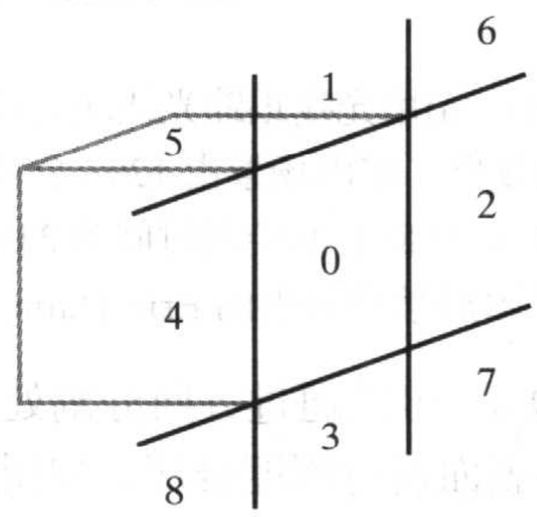


图 10.67　待定出口表面的分区

图 10.68 中的场景是使用空间细分算法进行光线跟踪的结果。如果不使用空间分割技术，则光线跟踪将花费 10 倍的时间。消除这些多边形亦可提高速度。对于包含 2048 个球面但无多边形表面的场景，同样的算法比基本的光线跟踪算法快 46 倍。

图 10.69 表示另一个使用空间分割技术和并行处理技术来对场景进行光线跟踪的场景，这幅 Rodin 的雕塑 Thinker 在 24 秒内生成，它跟踪了大约 150 万条光线。

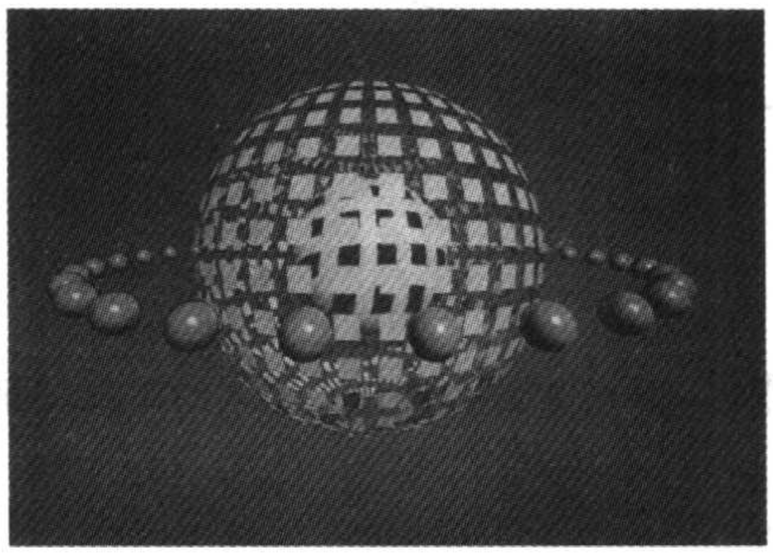


图 10.68　一个包含 37 个球体和 720 个多边形面的并行光线跟踪处理的场景。光线跟踪算法中每像素采用 9 束光线，树深度为 5，在 Alliant FX/8 上用空间分割技术处理该场景的速度为基本光线跟踪的 10 倍（新加坡信息技术研究所的 Lee-Hian Quek 提供）

图 10.69　在一个带有 32 个处理器的 Kendall Square Research KSR1 并行计算机上，光线跟踪该场景共需 24 秒，Rodin 的雕塑 Thinker 由 3036 个图元建模而成，使用两个光源及每像素一束光线来处理 1 675 776 条光线以获得全局照明效果（曼彻斯特大学计算机系的 M. J. Keates 和 R. J. Hubbold 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图10.70所示的场景使用一种空间分割形式的光线缓存（light-buffer）技术进行绘制。其中，每个立方体中心定位在一个点光源上，立方体的每个表面均由一正方形网格进行分割。光线跟踪程序保存一张透过每个正方形的光线所能“看见”的对象的排序列表，以加速阴影光线的处理。为了确定对象表面的光照效果，需要逐个计算每束阴影光线所对应的正方形，并根据它所对应的对象列表进行处理。

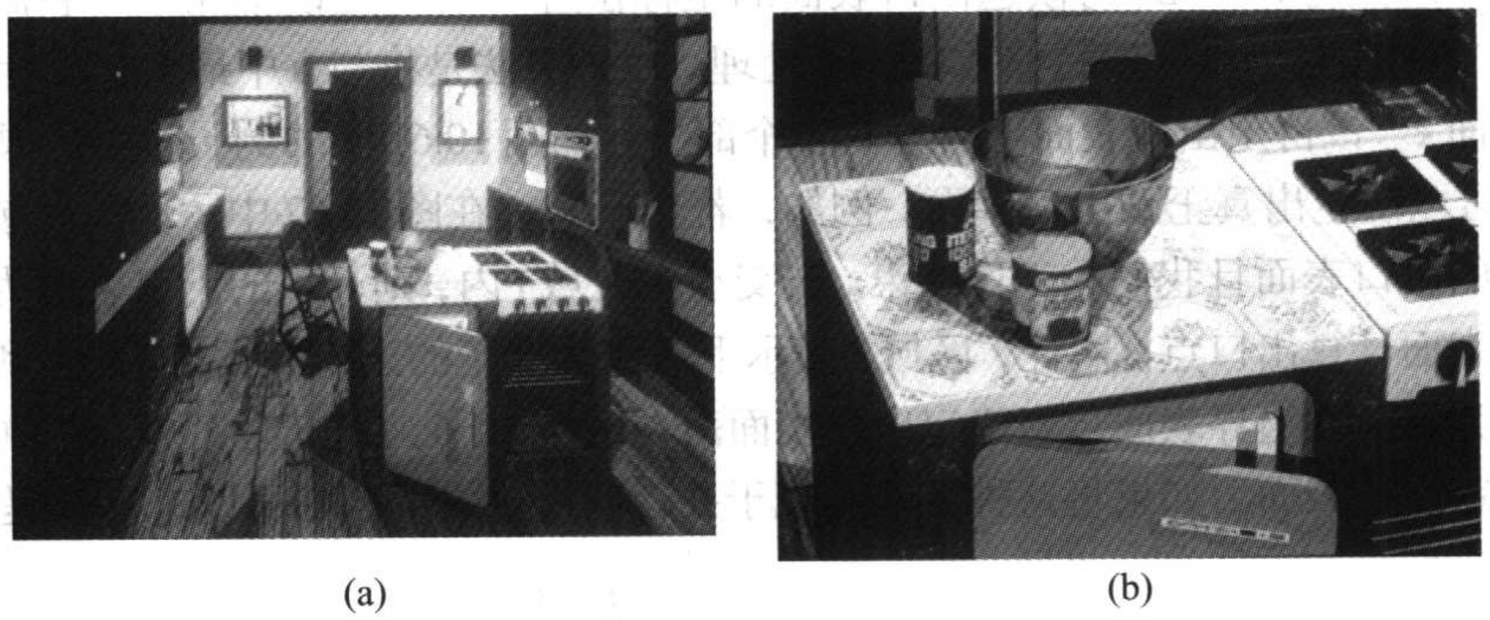


图 10.70　一个由 5 个光源照明的室内场景。(a)用光线跟踪光线缓存技术来处理阴影光线，(b)为(a)中部分室内场景以表示全局光照效果。室内场景由 1298 个多边形、4 个球、76 个圆柱体和 35 个四面体建模生成，在 VAX 11/780 上的绘制时间为 246 分钟，而不使用光线缓存则需 602 分钟（Cornell 大学计算机图形学项目组的 Eric Haines 和 Donald P.Greenberg 提供）

光线跟踪程序中的求交测试计算量，可以通过方向分割处理来降低。该方法考察包含一组光线的夹角区域。在每个区域中，将对象表面进行深度排序，如图 10.71 所示。每束光线仅需在包含它的区域内对对象进行测试。

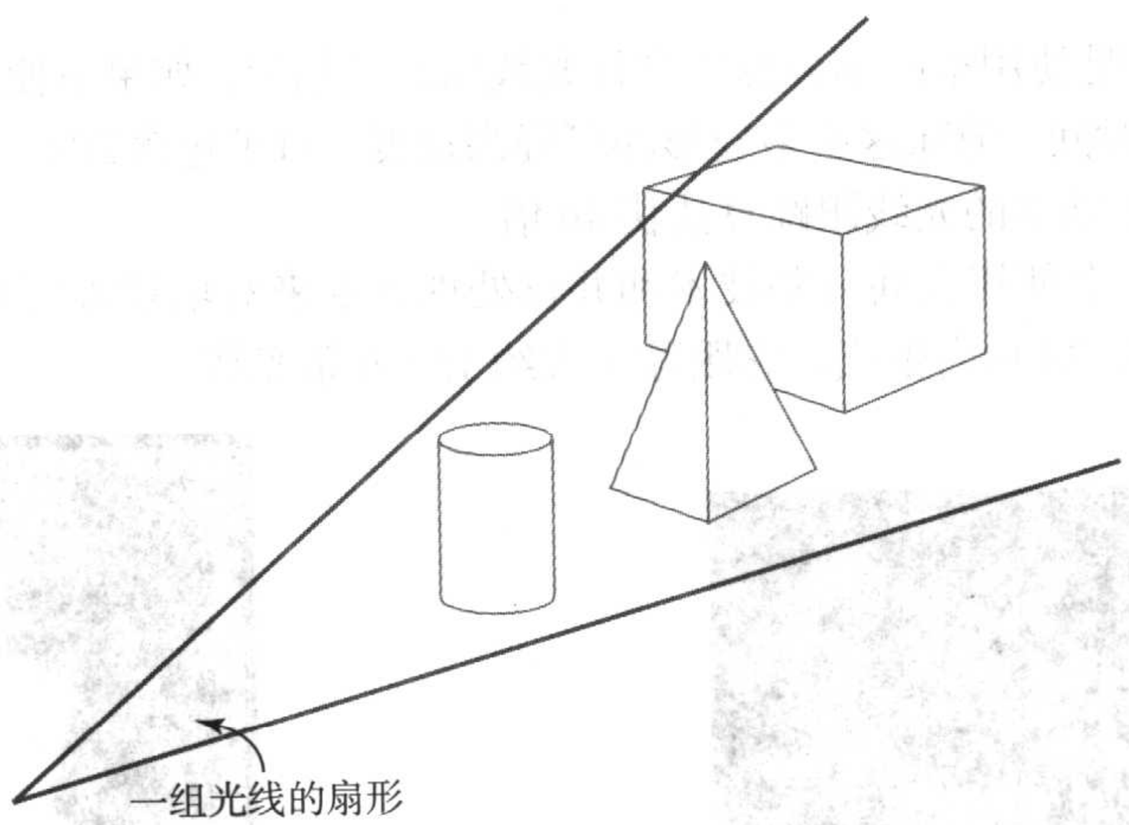


图 10.71　空间的方向分割技术，在该夹角中所有光束仅需在深度顺序上测试夹角内的对象表面

10.11.7　模拟照相机的聚焦效果

为了模拟场景中的照相机效果，要为安排在投影平面前面的凸透镜（或照相机的光圈）指定聚焦长度和其他参数。透镜参数设置成使场景中有些对象能聚焦而另一些对象在焦点之外。透镜的聚焦长度是从透镜中心到聚焦点 F 之间的距离，F 是一组经过该透镜的平行线的会聚点，如图 10.72 所示。35 mm 照相机聚焦长度一般是 50 mm。照相机光圈常用称为 f 数或 f 刻度的参数 n 描述，它是聚焦长度与光圈直径的比：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$n = \frac{f}{2r} \tag{10.83}$$

因此，我们可使用半径 r 或 f 数 n 以及聚焦长度 f 来指定照相机参数。对于一个更精确的聚焦模型，可使用胶片尺寸（宽和高）和聚焦长度来模拟照相机效果。

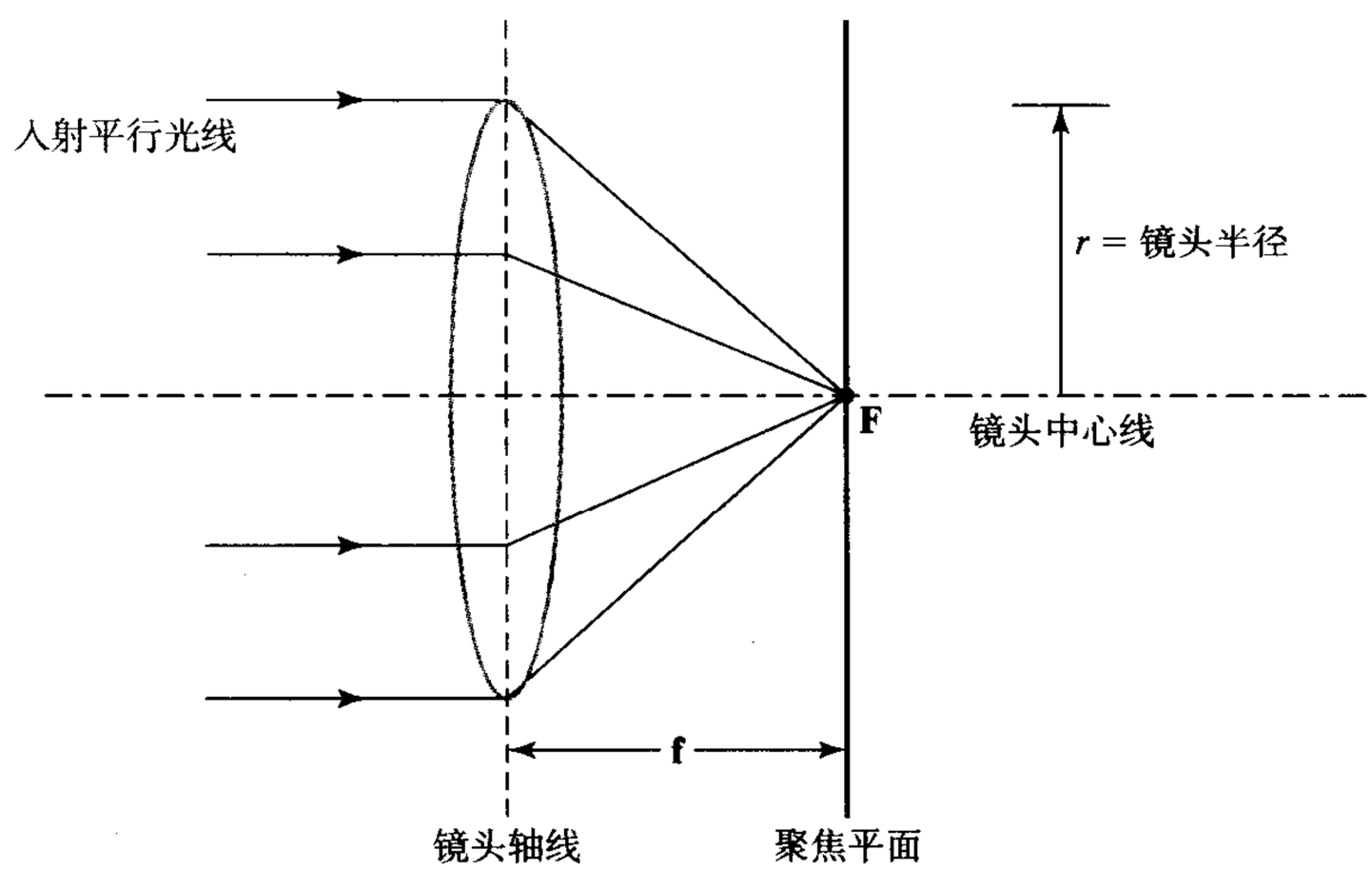


图 10.72　一片薄的凸透镜的侧视图。平行光线被该透镜聚焦到焦点平面上距透镜中心 f 的一个位置

光线跟踪算法一般使用几何光学中的**薄透镜公式**（thin-lens equation）：

$$\frac{1}{d} + \frac{1}{d_i} = \frac{1}{f} \tag{10.84}$$

参数 d 是透镜中心到一个对象位置的距离，而 d_i 是该透镜中心到聚焦该对象的图像平面的距离。该对象和它的图像位于沿经过透镜中心的一条直线上该透镜的两侧且 $d > f$（参见图 10.73）。因此，要将离透镜距离为 d 的特定对象聚焦，可将像素平面置于透镜后 d_i 处。

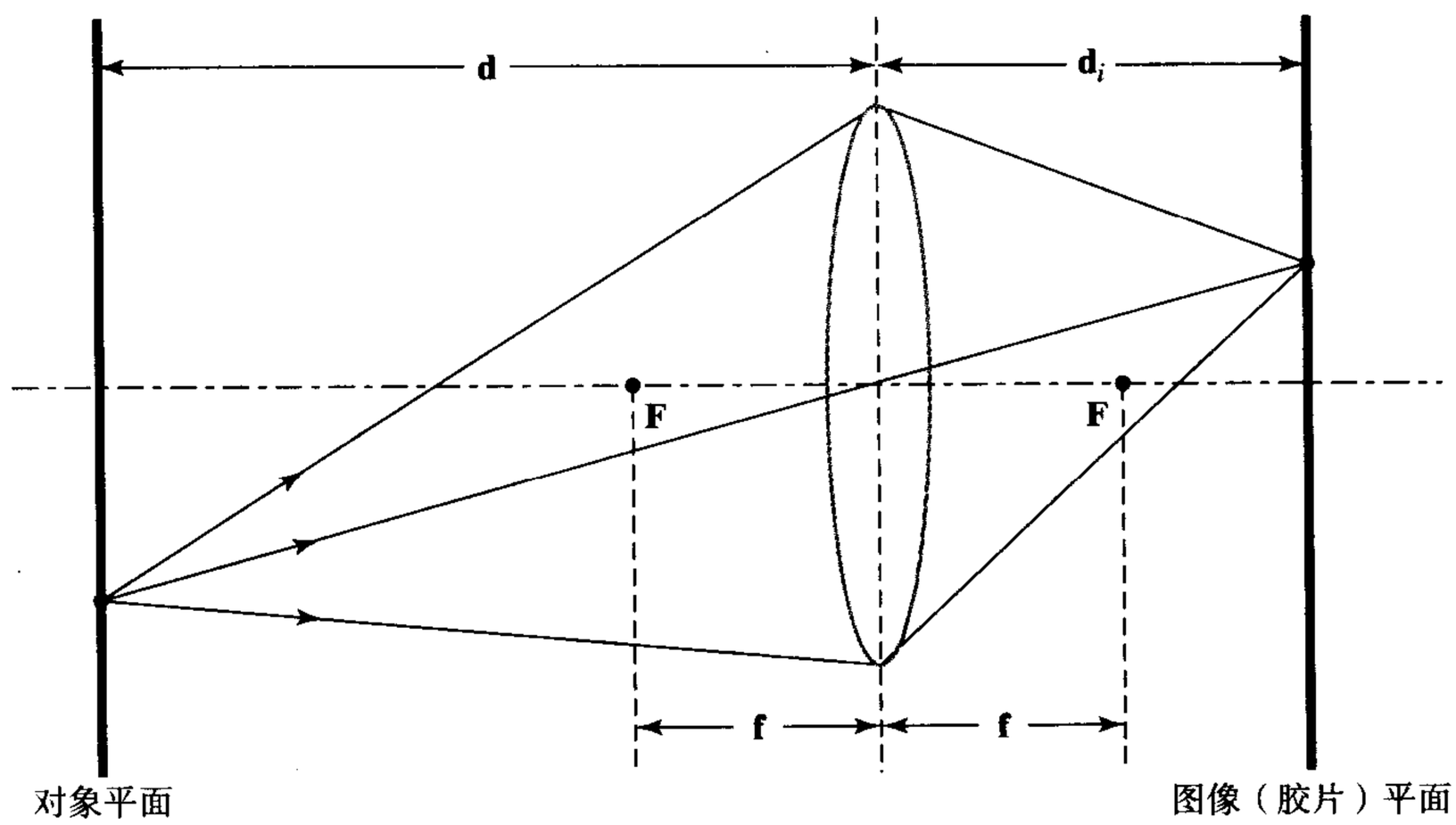


图 10.73　薄透镜参数。离透镜距离为 d 的对象聚焦到离透镜距离为 d_i 的图像平面上

对距离 $d' \neq d$ 的场景位置，投影点将不会聚焦在图像平面上。如果 $d' > d$，则该点聚焦到图像平面之前；而如果 $d' < d$，则该点聚焦到图像平面之后。位于 d' 的一个点在图像平面上的投影近似于一个小圆圈，称为**模糊圆**（the circle of confusion），而其直径计算如下：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$2r_c = \frac{|d' - d|f}{nd} \tag{10.85}$$

我们可以选择照相机参数使一定距离范围内的模糊圆的尺寸最小，该距离范围称为照相机的**景深**（depth of field）。另外，可为每一像素跟踪多条光线来获得通过透镜区域的取样位置，而**分布式光线方法**（distributed-ray method）将在后面讨论。

10.11.8 光线跟踪反走样

两种最基本的光线跟踪反走样技术为过采样（super-sampling）和适应性采样（adaptive sampling）。光线跟踪中的采样是4.17节讨论的采样方法的延伸。在过采样和适应性采样中，将像素看成一个有限的正方形区域，而非单独的点。过采样在每个像素区域内采用多束均匀排列的光线（采样点）。适应性采样则在像素区域的一些部分采用不均匀排列的光线。例如，可以在接近对象边缘处采用较多的光线以获得该处像素强度较好的估计值。（另一种采样方法是在像素区域中采用随机分布的光线。我们将在下一节中讨论该算法。）当对每个像素采用多束光线时，像素的光强度通过将各束光线强度取平均值而得到。

图10.74表示一个简单的过采样过程。其中，在每个像素的四角各生成一束光线。如果四束光线强度的差异较大，或在四束光线之间有小对象存在，则需将该像素区域进一步分割，并重复以上过程。例如，图10.75中的像素被16束光线分割为9个子区域，每束位于子像素的角点处。适应性采样是对那些四角光束强度不同或遇到小对象的子像素进行细分，细分工作一直持续到每个子像素的光线强度近似相等或每个像素中的光束数目达到上限，如256。

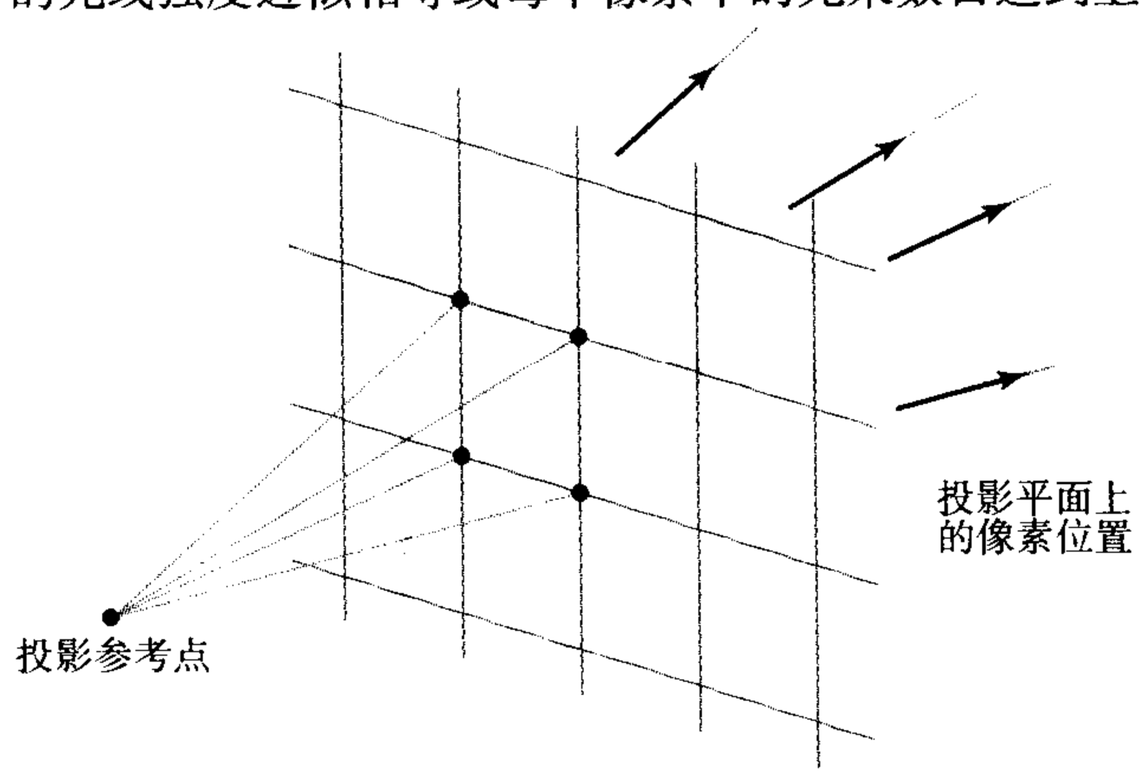


图10.74 在每个像素的四角处各取一束光线的过采样

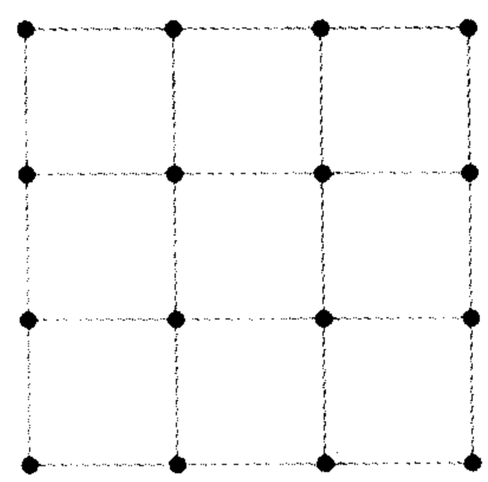


图10.75 将一个像素细分为9个子像素，每个子像素的角点处发出一束光线

图10.76是一个利用适应性细分的光线跟踪技术绘制出来的一个场景。其中使用了一个附加光源以生成具有真实感的柔和的阴影效果。场景中共生成了大约2600万条原始光线、3350万条阴影光线和6730万条反射光线，使用了关节链形体技术（参见13.8节）来生成棋子形象。并采用噪音函数的实体纹理技术生成木纹和大理石的表面图案（参见10.16节）。

我们也可以设置每束光线穿过子像素的中心而非像素角点，如图10.77所示。使用该方法，即可根据4.17节中的一个采样模式来对光线进行加权平均。

反走样显示场景的另一种方法是将像素光线看成一个锥体，如图10.78所示。对于每个像素只生成一束光线，但光线有一个有限的相交部分。为了确定被对象覆盖的像素区域的面积百分比，可以计算像素锥体与对象表面的交点。对于一个球体，这需要计算出两个圆周的交点；而对于多面体，则需求出圆周与多边形的交点。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

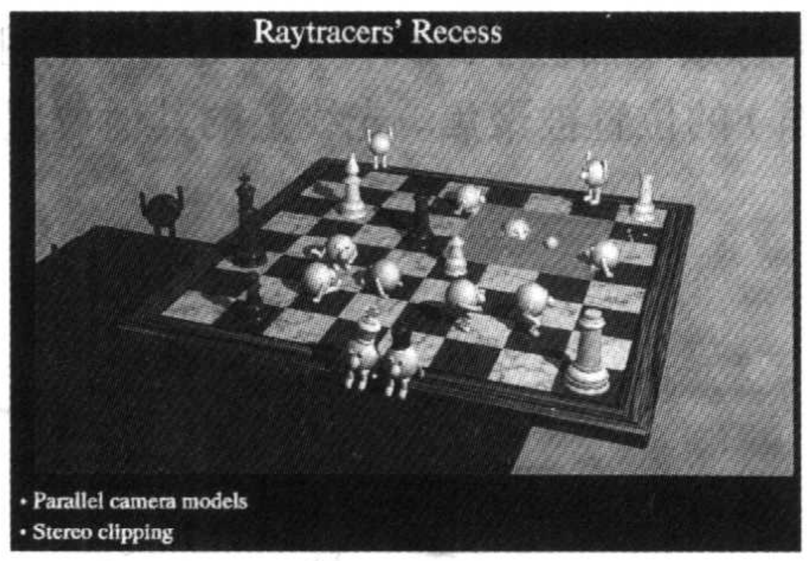


图 10.76　使用适应性细分光线跟踪方法生成的一个场景（Jerry Farm 提供）

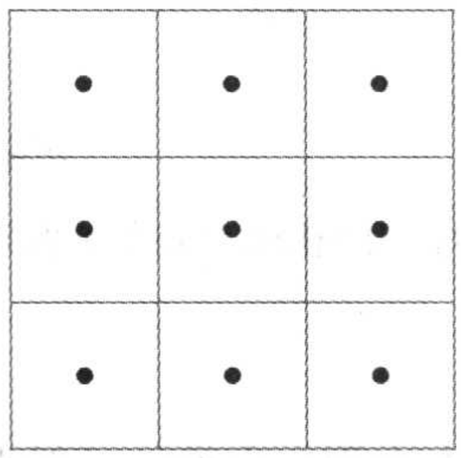


图 10.77　子像素区域中心的光束位置

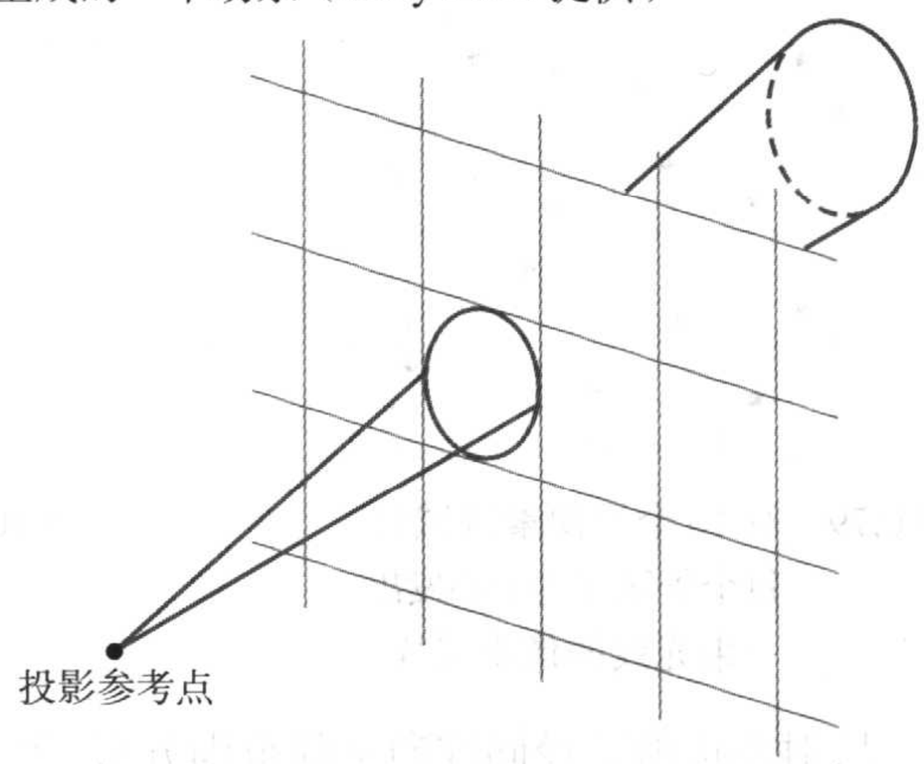


图 10.78　一个锥形的像素光束

10.11.9　分布式光线跟踪

这是一种根据光照模型中多种参数来随机分布光线的采样方法。光照参数包括：像素区域、反射与折射方向、照相机镜头区域及时间等。走样效果可由低级“噪音”来替代，这将改善图像质量，并能更好地模拟对象表面的光滑度和透明度、有限的照相机光圈、有限的光源以及移动对象的运动模糊显示。**分布式光线跟踪**（distributed ray tracing）必须提供一个在对对象表面光照进行准确描述时出现的多重积分的 Monte Carlo 估计值。

在像素平面上随机分布一些光线可以实现像素采样。但是，完全随机地选择光线位置可能导致像素内的部分区域出现光束密集，而其他部分却未经采样。在规则子像素网格上采用的一种称为抖动（jittering）的技术，可以获得像素区域内光束的较好的近似分布。这通常是将像素区域（一个单位正方形）划分为 16 个子区域（如图 10.79 所示），并在每个子区域内生成随机的抖动位置（jitter position）。通过将每一子区域的中心坐标抖动一个小分量 δ_x 和 δ_y 来获得随机光线位置，这里 $-0.5 < \delta_x$，$\delta_y < 0.5$。这样，就可以将抖动位置$(x + \delta_x, y + \delta_y)$作为中心坐标为$(x, y)$的单元内的光线位置。

对于 16 束光线可随机分配整数 1 到 16，并使用一张查找表来得到其他参数的值，如反射角和时间。每个子像素光线将穿过场景以确定其光强度的作用。然后，平均 16 束光线的强度，可以得到该像素的光强度值。如果子像素强度之间的差异过大，则还要进一步细分该像素。

为了获得照相机镜头的效果，要处理通过放在投影平面前的镜头的像素光线。如前所述，一个照相机用聚焦长度和其他有关参数来模拟，因此所选对象将被聚焦。然后我们将子像素光线分布到光圈中。假定每像素有 16 束光线穿过，则可将光圈区域分为 16 块。每块的中心位置赋给一个子像素，且使用下列过程来确定像素的分布式取样。从每块中心计算出一个抖动位置，从该抖动的块位置经过镜头焦点投影到场景中。我们将一光线的焦点定位到从子像素中心到镜头中心的线上距镜头

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

f的位置，如图10.80所示。如果像素平面位于距离镜头d_i处（参见图10.73），则沿光线接近镜头前距离d处的对象平面（取焦平面）的位置被聚焦。光线上的其他位置则模糊。增加子像素光束数目可改善焦点外对象的显示。

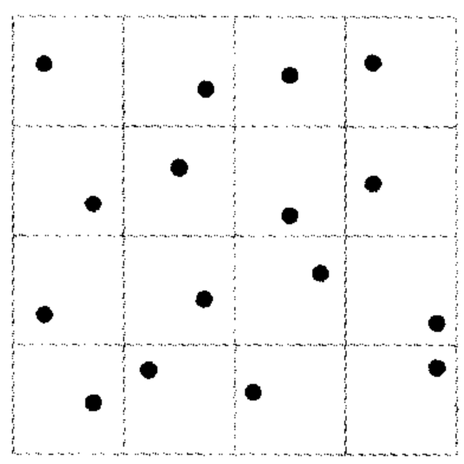


图 10.79　有16个子像素区域且每个子区域的中心发出一束光线的像素采样

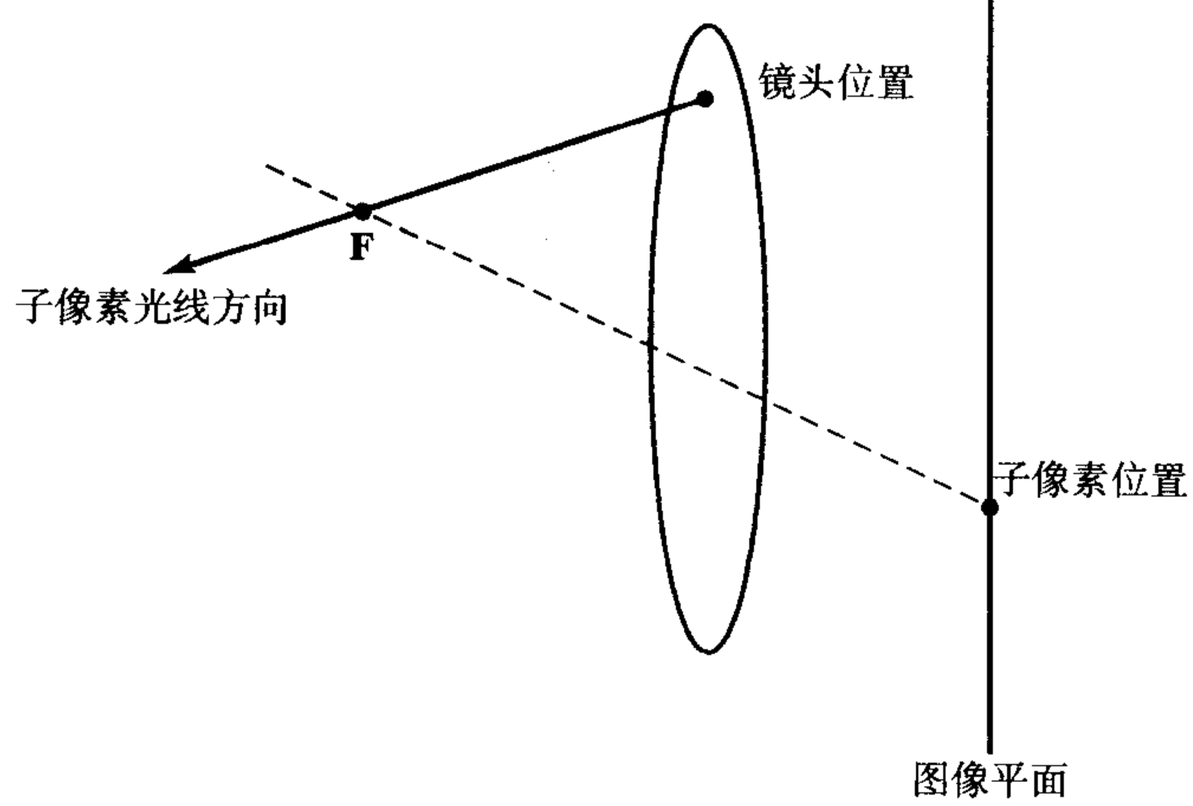


图 10.80　在镜头焦距为f的照相机上分布子像素光线

反射和透射路径同样向空间范围分布。为了模拟表面光泽，在表面交点处的反射光线将根据光束代码而分布于镜面反射方向**R**邻近的区域（参见图10.81）。距**R**最大的扩展区域将分割为16个角域，每束光线根据它的整数代码在区域中心附近的抖动位置被反射。我们可以使用Phong模型中的$\cos^{n_s}\phi$来确定反射范围的最大值。若材质是透明的，折射光线将沿着透视方向**T**按模拟透明性同样的方式进行分布（参见10.4节）。

如图10.82所示，可以在附加光源上分布一些阴影光线来对其进行处理。这样，将光源分割为一些小区域，阴影光线则被赋以指向不同区域的抖动方向。另外，可以根据其中光源的强度和该区域投影到对象表面的大小来对区域进行加权，加权系数较高的区域应有较多的阴影光线。如果一部分阴影光线被对象表面和光源之间的不透明对象挡住，则需在此表面点上生成半影（部分照明的区域）。但如果所有阴影光线均被挡住，则表面在该光源的本影（完全黑的）中。图10.83表示部分遮挡光源时，对象表面上所产生的本影和半影区域。

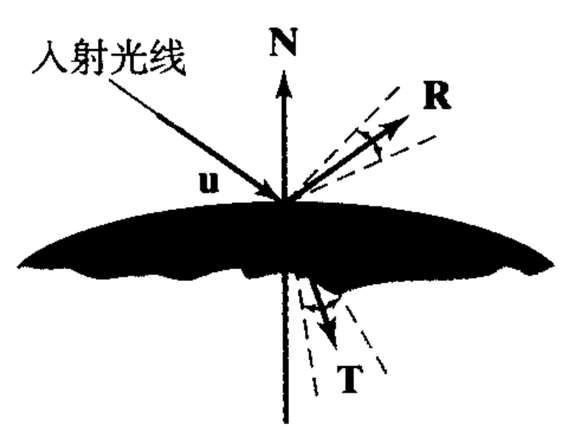


图 10.81　在反射方向**R**和透射方向**T**周围分布的子像素光线

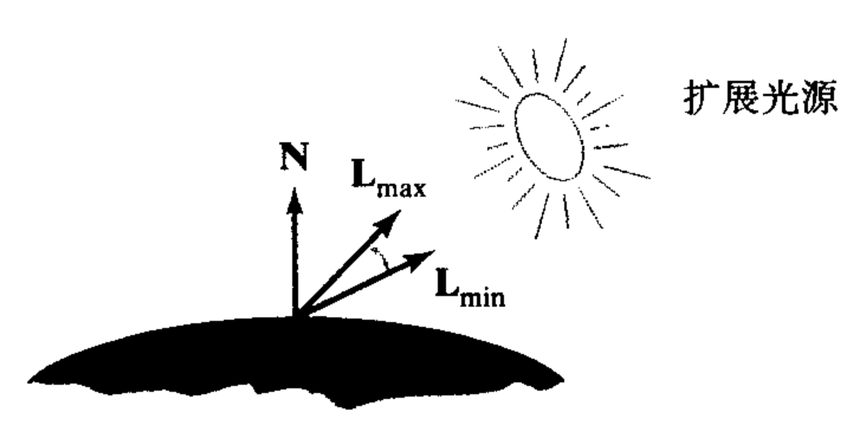


图 10.82　在一个有限大小的光源上分布阴影光线

通过在时间上分布光线，可以生成动感模糊。所有帧的总时间和各帧时间的细分根据场景所需的运动程度来确定。时间间隔用整数代码来标识，每束光线赋以一个光线代码所对应的时间间隔内的抖动时间。然后，对象运动到它们的位置，光线穿过场景进行跟踪。要绘制高度模糊的对象还需

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

要更多的光线。为了减少计算量，可以使用包围盒（球）来进行光线的初始求交测试。即根据运动的要求来移动包围对象，并进行求交测试。若光线与包围体不相交，则无需对包围体中的表面进行处理。图 10.84 给出了一个采用动感模糊技术显示的场景，该图像采用 4096 × 3550 个像素及每像素 16 束光线并根据分布式光线跟踪进行绘制。除了动感模糊反射之外，阴影由室内四周用于照明桌球台的附加光源所生成的半影区域来表示。

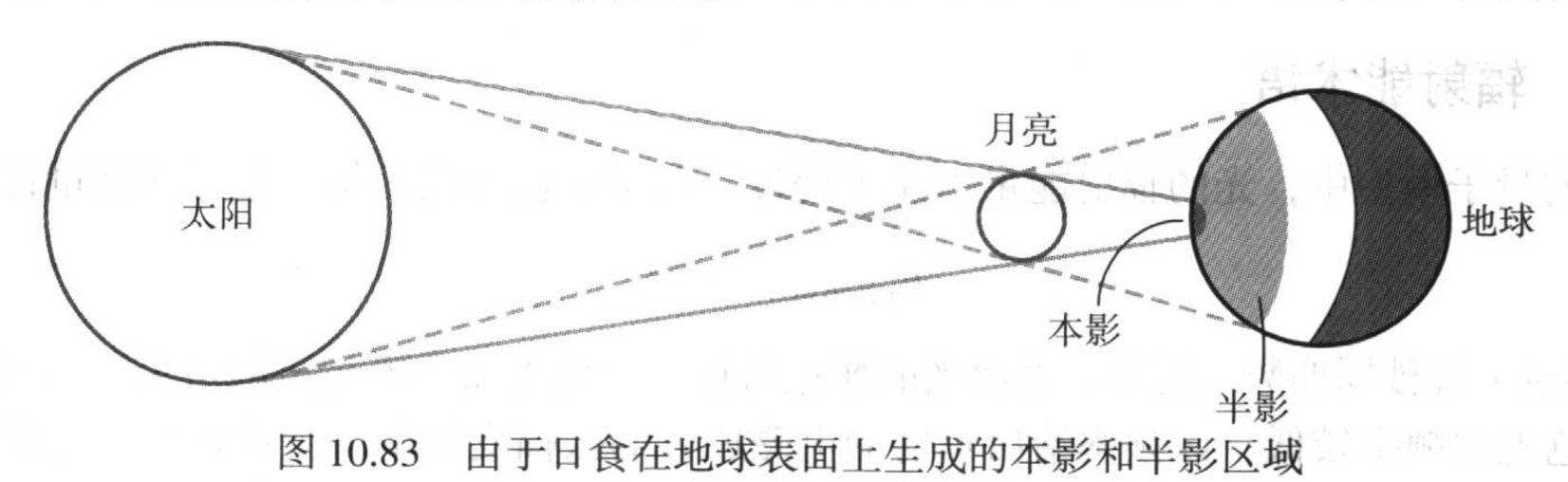


图 10.83　由于日食在地球表面上生成的本影和半影区域

图 10.85 和图 10.86 给出了使用分布式光线跟踪绘制的例子。图 10.87 表示分布式光线跟踪的聚焦折射和反走样效果。

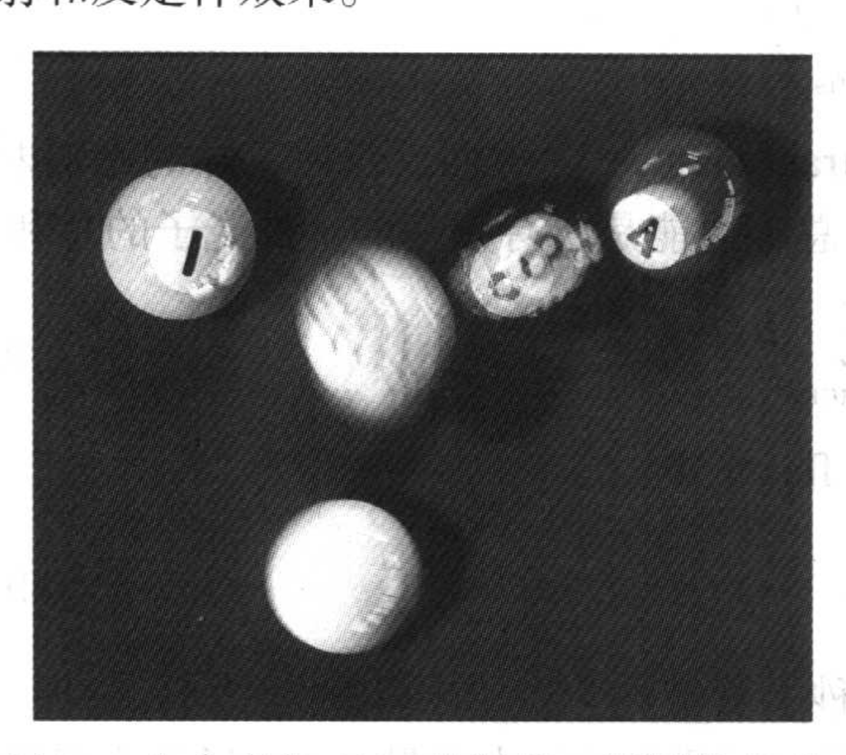


图 10.84　一个命名为 1984 的场景，使用分布式光线跟踪来表现动感模糊和半影效果（Pixar 提供）

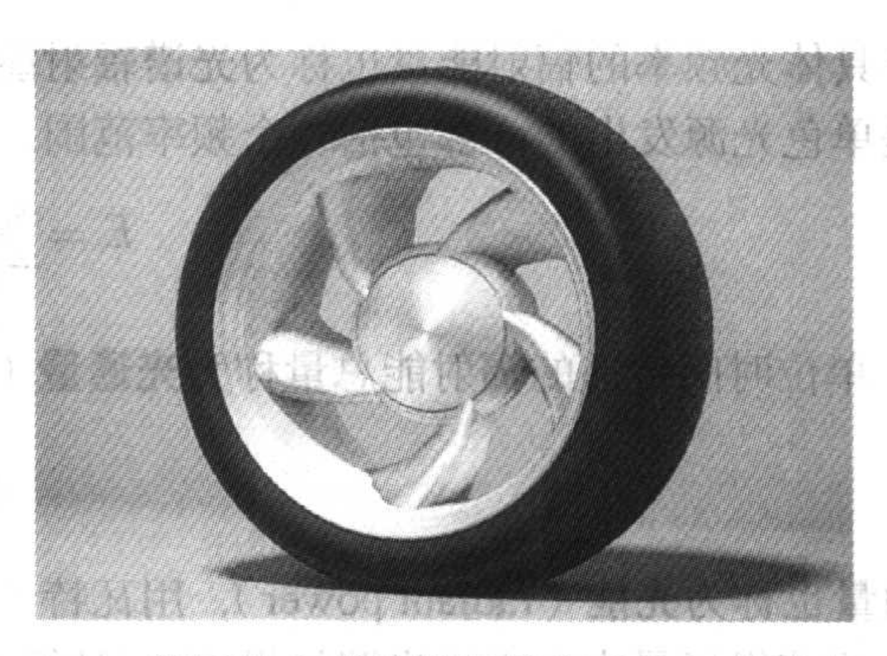


图 10.85　使用分布式光线跟踪生成的一个磨光的铝轮轴的反射和阴影效果（Cornell 大学的 Stephen H. Westin 提供）

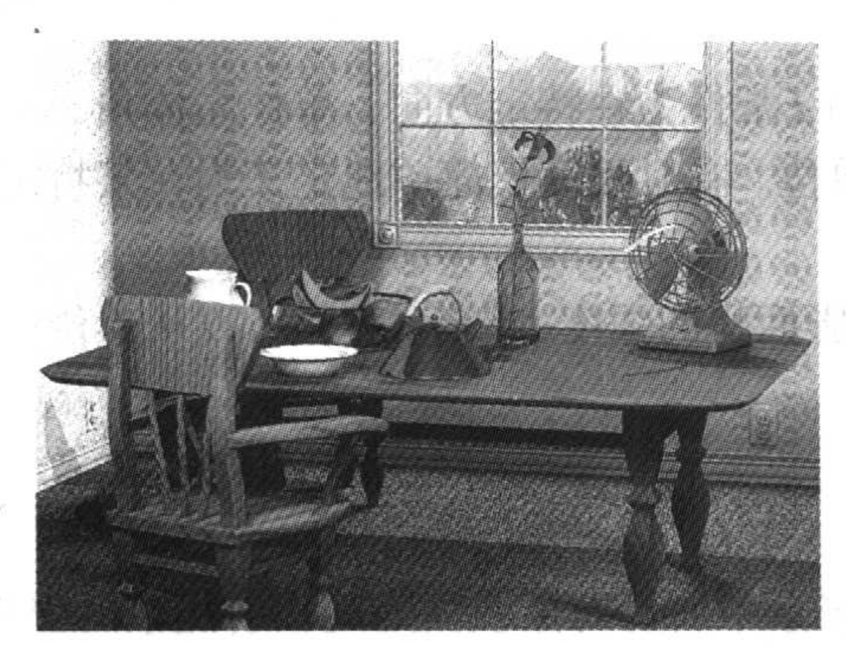


图 10.86　使用分布式光线跟踪方法绘制而成的室内场景（加州理工学院计算机图形学实验室的 John Snyder、Jed Lengyel、Devendra Kalra 和 Al Barr 提供）

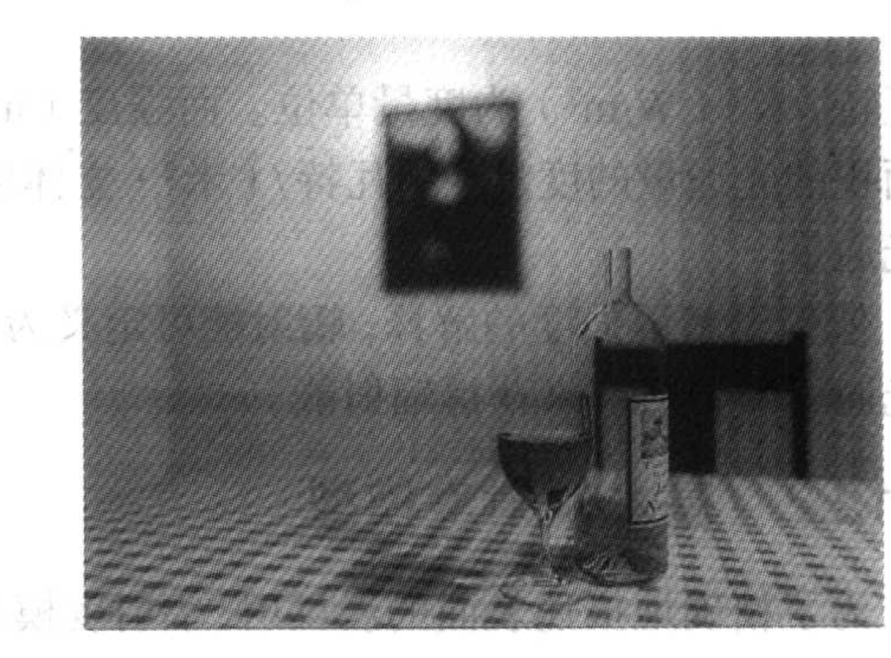


图 10.87　结合光线跟踪和辐射度方法生成的场景的聚焦、反走样及光照效果，真实感的物理光照模型用于生成包括酒瓶阴影中的液体的折射效果（印第安纳大学 Peter Shirley 提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.12 辐射度光照模型

尽管基本光照模型能为许多应用生成相当不错的结果，该模型中的简单近似方法还不能对一些光照效果进行精确描述。我们可以考察辐射能在对象表面之间的转移和能量守恒定律，从而准确地建立对象表面的漫反射模型。这种描述漫反射的方法通常称为**辐射度模型**（radiosity model）。

10.12.1 辐射能术语

在光的量子模型中，光的辐射能量由光子携带。对于单色光线，每一光子的能量计算如下：

$$E_{\text{photon},f} = hf \tag{10.86}$$

频率f用赫兹（每秒周期数）度量，表示光的颜色特征。蓝色光在电磁光谱的可见段中有较高的频率，而红色光的频率较低。该频率给出辐射的电磁成分的振幅变化率。参数h是普朗克（Planck）常数，其值为6.6262×10^{-34}焦耳·秒（J·S），与光的频率无关。

单色光辐射的总能量是

$$E_f = \sum_{\text{all photons}} hf \tag{10.87}$$

一个具体光频率的辐射能量也称为**光谱辐射能**（spectral radiance）。然而，任何实际的光辐射，甚至是单色光源发出的，都包括一个频率范围。因此，总辐射能是所有频率的全部光子的总和：

$$E = \sum_{f} \sum_{\text{all photons}} hf \tag{10.88}$$

单位时间传递的辐射能总量称为**光通量**（radiant flux）Φ：

$$\Phi = \frac{dE}{dt} \tag{10.89}$$

光通量也称为**光能**（radiant power），用瓦特（焦耳/秒）度量。

要获得场景中表面的光照效果，我们计算单位面积上离开表面的光通量。这个量称为**辐射度B**，或**辐出率**（radiant exitance），

$$B = \frac{d\Phi}{dA} \tag{10.90}$$

以瓦特/米^2（W/m^2）为度量单位。而**强度**（intensity）I常作为在特定方向、单位立体角、单位投影面积的光通量的度量，用瓦特/（米^2·立体度）作单位。有时强度仅简单地定义为在某特定方向的光通量。

依赖于术语强度的解释，**辐射能**可定义为单位投影面积的强度。或者，我们可以从单位立体角的光通量或辐射度来获得辐射能。

10.12.2 基本辐射度模型

为了精确地描述表面的漫反射，辐射度模型计算场景中所有对象表面之间辐射能量的交换。由于涉及的一组方程很难求解，基本辐射度模型假设所有表面都较小，并且是不透明、理想的漫反射器（朗伯漫反射体）。

通过确定离开场景中各表面点的辐射能的微分dB，并加上在所有表面上贡献的能量总和来得到对象表面之间的能量转移总量。在展示从一个表面传递的辐射能的图10.88中，dB是由角度θ与ϕ所确定的方向上，单位时间、单位表面面积上立体角度的微分$d\omega$的范围内表面点所发出的可见光通量。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

方向(θ, ϕ)上漫辐射的光强I，可描述为单位时间、单位投影面积、单位立体角度上的辐射能量，或

$$I = \frac{dB}{d\omega \cos\phi} \tag{10.91}$$

假定对象表面为一理想的漫反射器（参见10.3节），我们可以将所有观察方向上的光强I置为常量。这样，$dB/d\omega$与投影面的面积成正比（参见图10.89）。为了从表面点上获得能量辐射率的总和，必须将各方向上的辐射进行累计。即必须求出从以表面点为中心的半球所发散出来的全部能量，如图10.90所示，即

$$B = \int_{\text{hemi}} dB \tag{10.92}$$

对于一个理想的漫反射器，I为常量。因此我们可以将辐射能量B表示如下：

$$B = I \int_{\text{hemi}} \cos\phi \, d\omega \tag{10.93}$$

立体角的微分$d\omega$也可以表示为（参见附录A）

$$d\omega = \frac{dS}{r^2} = \sin\phi \, d\phi \, d\theta$$

因而

$$\begin{aligned} B &= I \int_0^{2\pi} \int_0^{\pi/2} \cos\phi \sin\phi \, d\phi \, d\theta \\ &= I\pi \end{aligned} \tag{10.94}$$

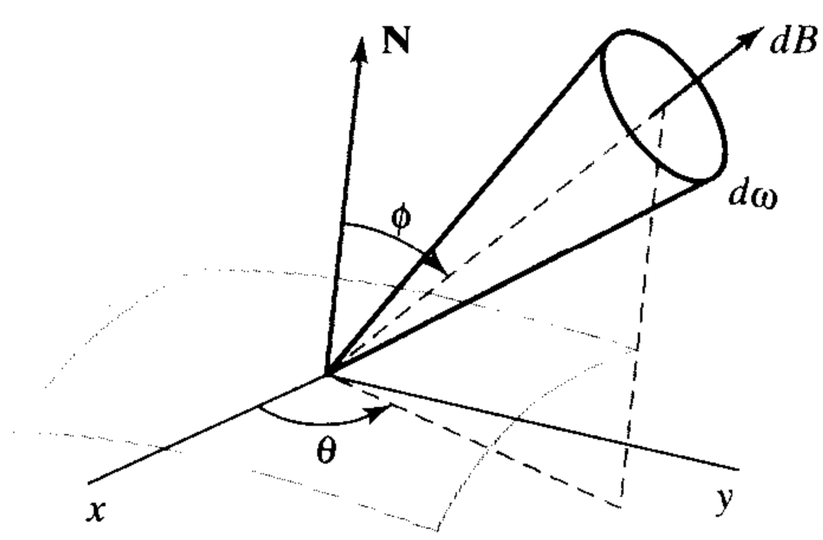


图 10.88　某表面点所发出的、在方向(θ, ϕ)上、立体角$d\omega$内的可见辐射能

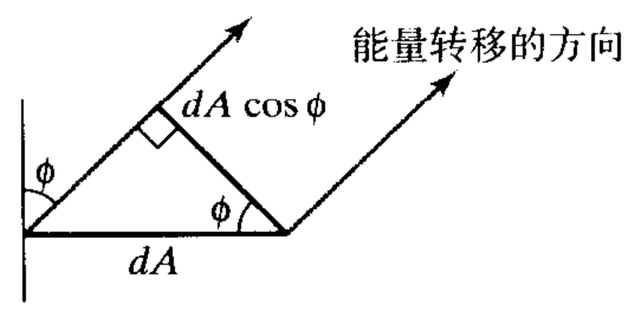


图 10.89　对于一个单位表面元素，垂直于能量转移方向的投影面积等于$\cos\phi$

可以通过设立表面的“闭包”来形成多个表面上光反射的模型（参见图10.91）。闭包中的每个表面要么是一个反射体、一个发射体（光源），或是一个反射与发射的结合体。令辐射度参数B_k为单位面积离开表面k的辐射能总速率，入射能量参数H_k为单位时间、单位面积上到达表面k的闭包内所有表面的辐射能总和，即

$$H_k = \sum_j B_j F_{jk} \tag{10.95}$$

其中，参数F_{jk}为表面j和k的形状因子，它是表面j到达表面k的辐射能与离开表面j的辐射能的比率。

对于一个在闭包中有n张表面的场景，由表面k所发出的辐射能可由**辐射度方程**（radiosity equation）描述：

$$\begin{aligned} B_k &= E_k + \rho_k H_k \\ &= E_k + \rho_k \sum_{j=1}^{n} B_j F_{jk} \end{aligned} \tag{10.96}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

若表面 k 不是光源，则 $E_k = 0$；否则，E_k 为单位面积（W/m²）上由表面 k 发出的能量的速率。参数 ρ_k 为表面 k 的反射因子（向各方向反射的入射光线的百分比）。漫反射因子与在经验光照模型中使用的漫反射系数相关。平面和凸面均无法“看见”其自身，因此不会发生自入射情况，并且每个表面的形状因子 F_{kk} 均为 0。

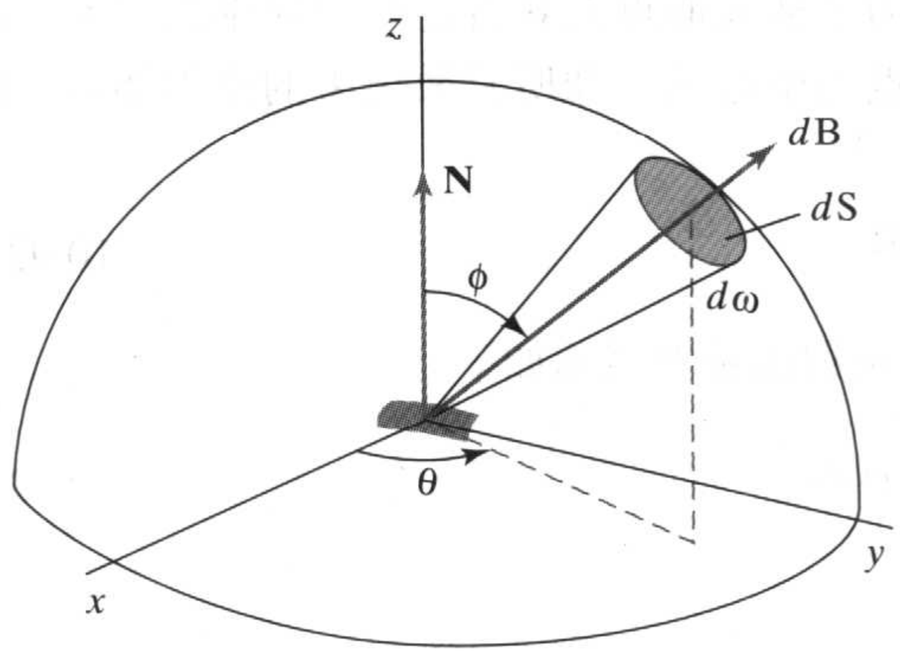


图 10.90　由一个表面点发出的总辐射能是以表面点为中心的半球上所有方向辐射能的总和

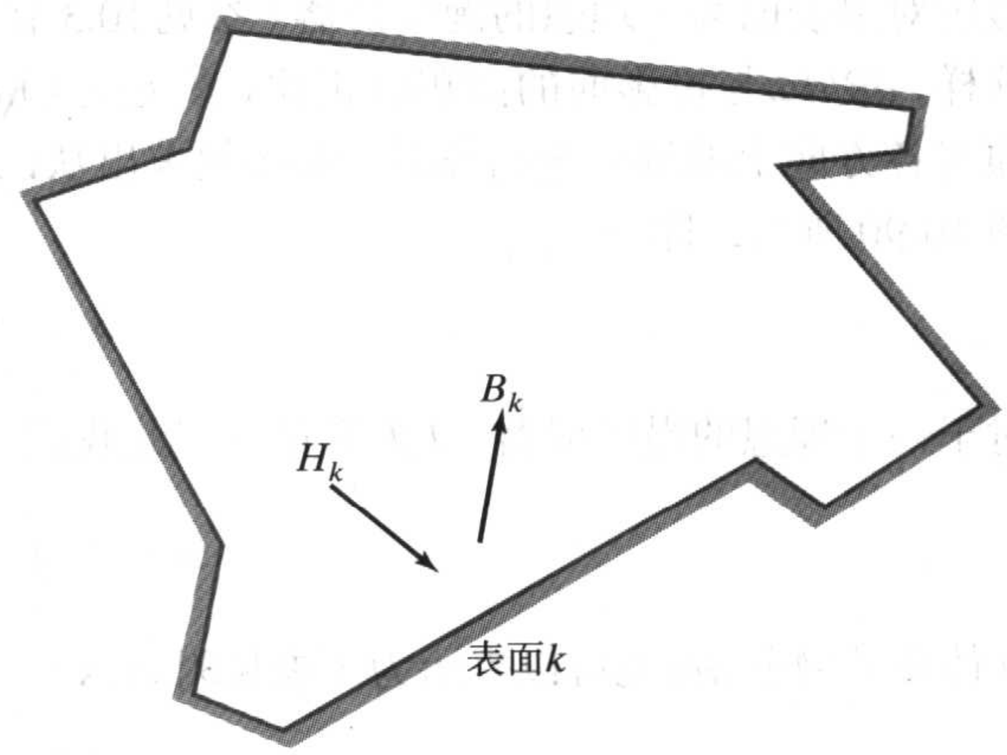


图 10.91　辐射度模型中表面的闭包

若给定 E_k、ρ_k 与 F_{jk} 的数组值，为了得到闭包内各表面的光照效果，必须对 n 张表面联立求解辐射方程，即求解：

$$(1-\rho_k F_{kk})B_k - \rho_k \sum_{j \neq k} B_j F_{jk} = E_k \qquad k = 1, 2, 3, \ldots, n \tag{10.97}$$

或

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_2 F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \tag{10.98}$$

然后，将辐射值 B_k 除以 π 从而将其转换为光强度 I_k。对于彩色场景，可以从 ρ_k 与 E_k 的颜色分量计算出辐射度（B_{kR}，B_{kG}，B_{kB}）的 RGB 分量。

在求解方程（10.97）之前，须确定出形状因子 F_{jk} 的值，可以考察由表面 j 转移至表面 k 的能量（参见图 10.92）。由表面 dA_j 向表面 dA_k 所转移的辐射能的速率为

$$dB_j\, dA_j = (I_j \cos\phi_j\, d\omega)\, dA_j \tag{10.99}$$

但立体角 $d\omega$ 可根据垂直于 dB_j 的方向上表面 dA_k 的投影而表示为

$$d\omega = \frac{dA}{r^2} = \frac{\cos\phi_k\, dA_k}{r^2} \tag{10.100}$$

因而，等式（10.99）可表示为

$$dB_j\, dA_j = \frac{I_j \cos\phi_j \cos\phi_k\, dA_j\, dA_k}{r^2} \tag{10.101}$$

两表面之间的形状因子是由面积 dA_j 发散出来的能量中投射至 dA_k 部分所占的百分比：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$F_{dA_j,dA_k} = \frac{\text{投射到}dA_k\text{上的能量}}{\text{离开}dA_j\text{的总能量}}$$

$$= \frac{I_j \cos\phi_j \cos\phi_k \, dA_j \, dA_k}{r^2} \cdot \frac{1}{B_j \, dA_j} \tag{10.102}$$

由于 $B_j = \pi I_j$，因此

$$F_{dA_j,dA_k} = \frac{\cos\phi_j \cos\phi_k \, dA_k}{\pi r^2} \tag{10.103}$$

由面积 dA_j 发出的投射在整个表面 k 上的能量为

$$F_{dA_j,A_k} = \int_{\text{surf}_j} \frac{\cos\phi_j \cos\phi_k}{\pi r^2} dA_k \tag{10.104}$$

其中，A_k 为表面 k 的面积。现在，可以将两表面间的形状因子定义为前面表达式的平均面积，即

$$F_{jk} = \frac{1}{A_j} \int_{\text{surf}_j} \int_{\text{surf}_k} \frac{\cos\phi_j \cos\phi_k}{\pi r^2} dA_k \, dA_j \tag{10.105}$$

可以使用数值积分技术，并规定以下条件来对等式（10.105）进行积分：

- $\sum_{k=1}^{n} F_{jk} = 1$，对所有 k（能量守恒）
- $A_j F_{jk} = A_k F_{kj}$（对等光反射）
- $F_{jj} = 0$，对所有 j（假设仅含平面片和凸面片）

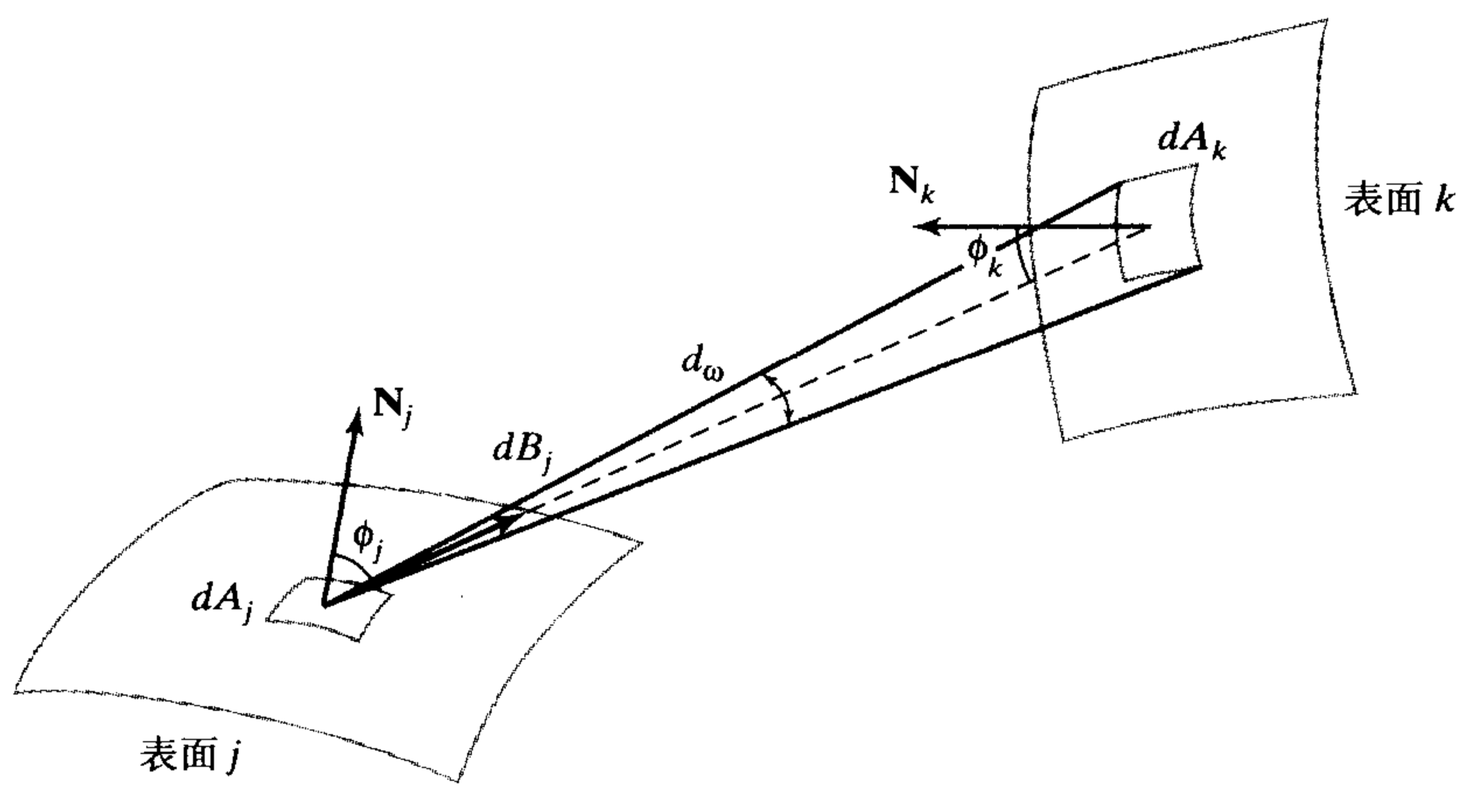


图 10.92　由面片 dA_j 向 dA_k 辐射能微分量 dB_j 的转移

为了应用辐射度模型，需要将场景中的每一表面分割为许多小多边形。分割越细，显示的真实感越强，但需花费更多的场景绘制时间。可以使用半立方体近似半球体以加速形状因子的计算，即使用一组线性（平面）表面替代球面。一旦计算出形状因子，即可用高斯消解或 LU 分解方法（参见附录 A）来联立求解线性方程（10.97）式。也可由 B_j 的近似值开始，使用 Gauss-Seidel 方法来迭代计算线性方程组。每次迭代，可以使用先前得到的辐射度值，并根据以下辐射度方程来为表面 k 计算辐射度的估计值：

$$B_k = E_k + \rho_k \sum_{j=1}^{n} B_j F_{jk}$$

我们可以逐步显示场景，每次迭代均能得到一个改善的对象表面的绘制结果，直至每次计算出的辐射度值近似相等。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.12.3 逐步求精的辐射度方法

虽然辐射度方法可以生成高度真实感的对象表面绘制结果，但同时带来了巨大的存储需求和计算形状因子所需的大量处理时间。采用逐步求精方法（progressive refinement），可以重新组织迭代辐射度算法以加速计算，并降低每次迭代的存储需求。

根据辐射度方程，两表面之间的辐射能量可以由下列等式计算：

$$\text{由 } B_j \text{ 传给 } B_k \text{ 的能量} = \rho_k B_j F_{jk} \tag{10.106}$$

相反：

$$\text{由 } B_k \text{ 传给 } B_j \text{ 的能量} = \rho_j B_k F_{kj}, \qquad \text{对所有 } j \tag{10.107}$$

还可以改写为

$$\text{由 } B_k \text{ 传给 } B_j \text{ 的能量} = \rho_j B_k F_{jk} \frac{A_j}{A_k}, \qquad \text{对所有 } j \tag{10.108}$$

这种关系是逐步求精辐射度方法的计算基础。使用单个表面k，可以计算出所有形状因子F_{jk}和由该表面向场景中所有其他表面的光转移。这样，每次仅需计算和保存一个半立方体和相关的形状因子。在下一次迭代中，用所选择的另一个表面的值来取代这些参数值。在逐个选择面片的过程中可逐步改善表面绘制效果。

首先，对所有表面设置$B_k = E_k$。然后选择辐射度最高的表面，也就是将成为最亮的发射体的表面，对其他所有表面计算近似辐射度。每次重复该步骤，即首先选择辐射能最高的光源，然后选择从光源接收到的光能量最多的表面。以下算法表示了简单的逐步求精方法的步骤。

```
for each patch k
   \*  Set up hemicube and calculate form factors F [j][k].  */

for each patch j {
   dRad   = rho [j] * B [k] * F [j][k] * A [j] / A [k];
   dB [j] = dB [j] + dRad;
   B [j]  = B [j] + dRad;
}

dB [k] = 0;
```

在算法的每一步，$\Delta B_k A_k$值最大的表面都将被选为发射表面，因为辐射度是对单位面积上辐射能的计算。我们将所有表面的初始值均设为$\Delta B_k = B_k = E_k$，逐步求精算法近似模拟了穿过场景的实际的光传播。

逐步显示对象表面的绘制结果可以生成由黑暗到全部照亮的一系列场景。在第一步结束时，照亮的表面仅为光源和从发射体位置可见的非发光表面。为了产生更有用的场景初始视图，我们可以设置一个环境光等级，以使所有表面获得一定的照明度。在每次迭代时，可以根据射入场景的辐射光能来减弱环境光强度。

图10.93表示一个利用逐步求精辐射模型绘制的场景。不同光照条件下，场景的辐射模型绘制如图10.94、图10.95和图10.96所示。通常，人们将光线跟踪技术与辐射度算法相结合，以生成高度真实感的漫反射和镜面反射效果，如图10.87所示。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

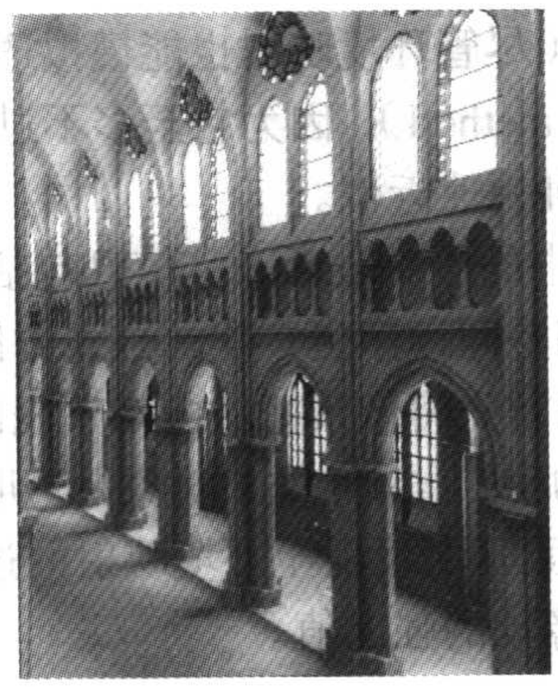


图 10.93　John Wallace 与 John Lin 使用 Hewlett-Packard Starbase Radiosity 和 Ray Tracing 软件、采用逐步求精辐射度方法绘制出来的Chartres大教堂的正厅，其中，辐射度形状因子由光线跟踪技术计算出来（Hewlett-Packard公司的 Eric Haines提供）

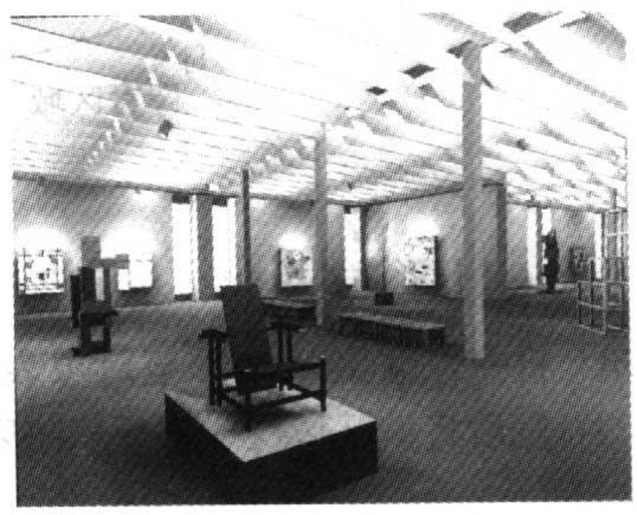
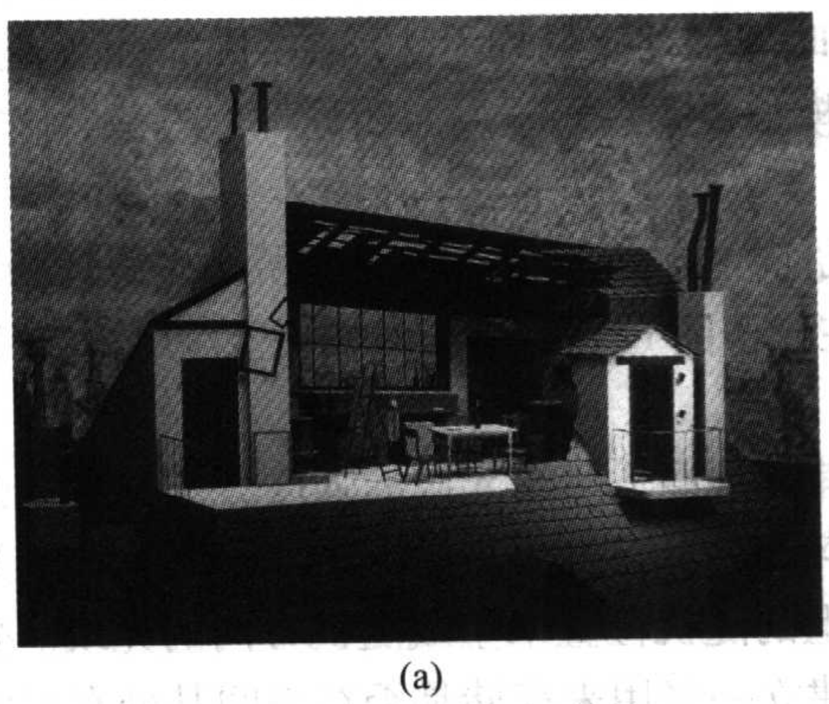
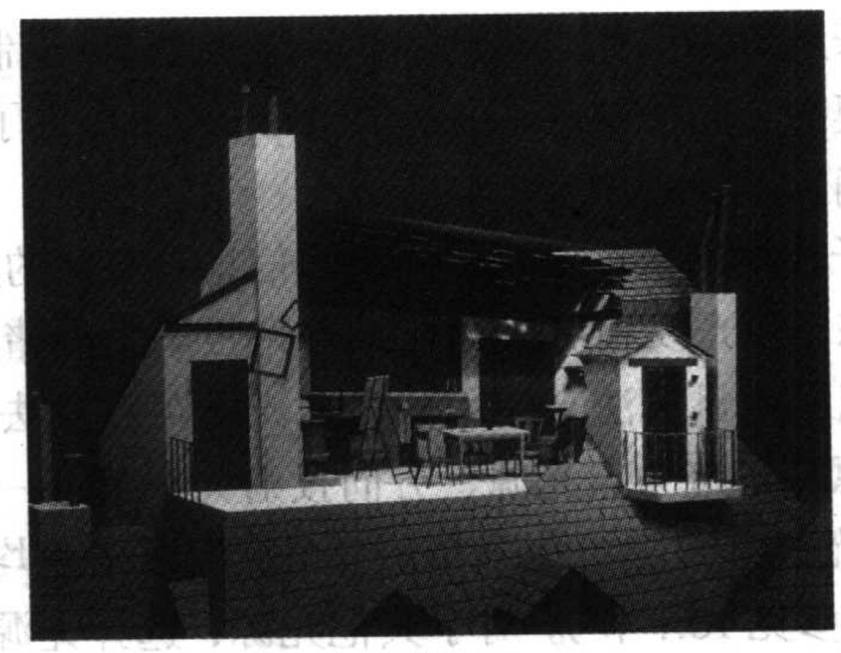


图 10.94　使用逐步求精辐射度方法绘制的博物馆场景（Cornell 大学提供）

图 10.95　使用逐步求精辐射度方法模拟的楼梯（Cornell 大学提供）



(a)



(b)

图 10.96　在不同光照环境下对 Parisian 顶楼的模拟：(a)白天的效果；(b)夜晚的效果（Cornell 大学提供）

10.13　环境映射

另一种模拟全局反射效果的方法是，定义一个描述单个或一组对象周围环境的光强度数组。这样，我们只需根据观察方向将环境数组（environment array）简单地映射至一个相关对象表面，而无需进行光线跟踪或辐射度计算来实现全局镜面反射和漫反射效果。该过程称为**环境映射**

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

（environment mapping）或**反射映射**（尽管环境映射也能模拟透明效果）。也将其称为“穷人的光线跟踪”（the poor person' s ray-tracing method），因为它至多不过是10.11节和10.12节两节所讨论的较精确的全局光照明绘制技术的一种快速近似。

环境映射定义在一个封闭环境中的对象表面，其信息包括光源的强度值、天空和其他背景物体。图10.97表示了一个球形的封闭空间，但更常用的是立方体或圆柱体形状的封闭空间。

为了绘制一个对象的表面，可以首先将像素区域向对象表面投影，然后将投影像素区域反射至环境映射中，为每个像素选取表面的明暗强度。如果对象是透明的，还可将投影像素区域折射至环境映射中。图10.98表示了将投影像素区域反射至环境映射的过程。在环境映射中的相交区域，可以通过光强值取平均的方法来得到像素的强度值。

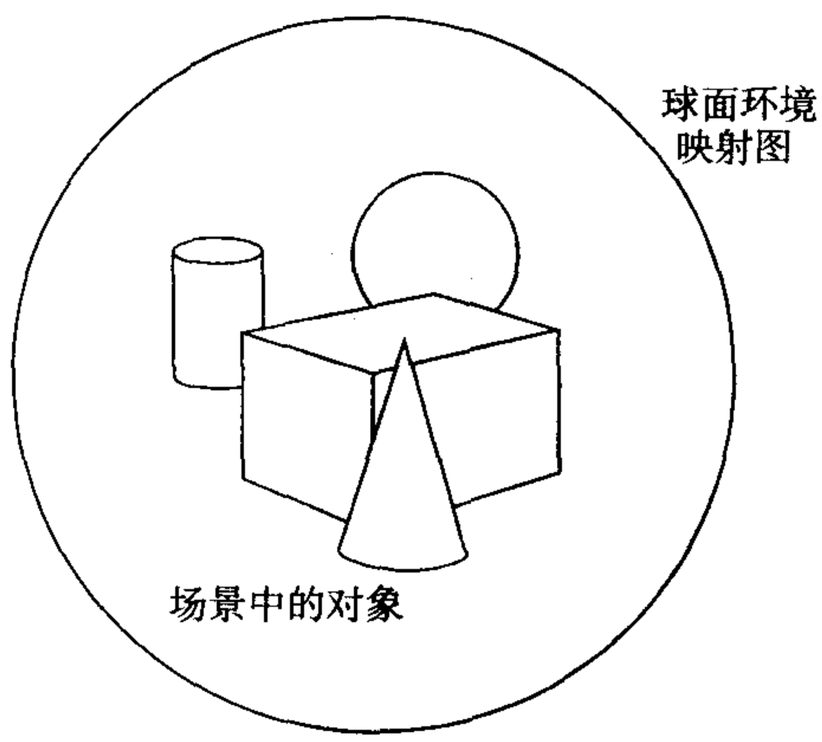


图10.97 一个包括环境映射的球形封闭空间

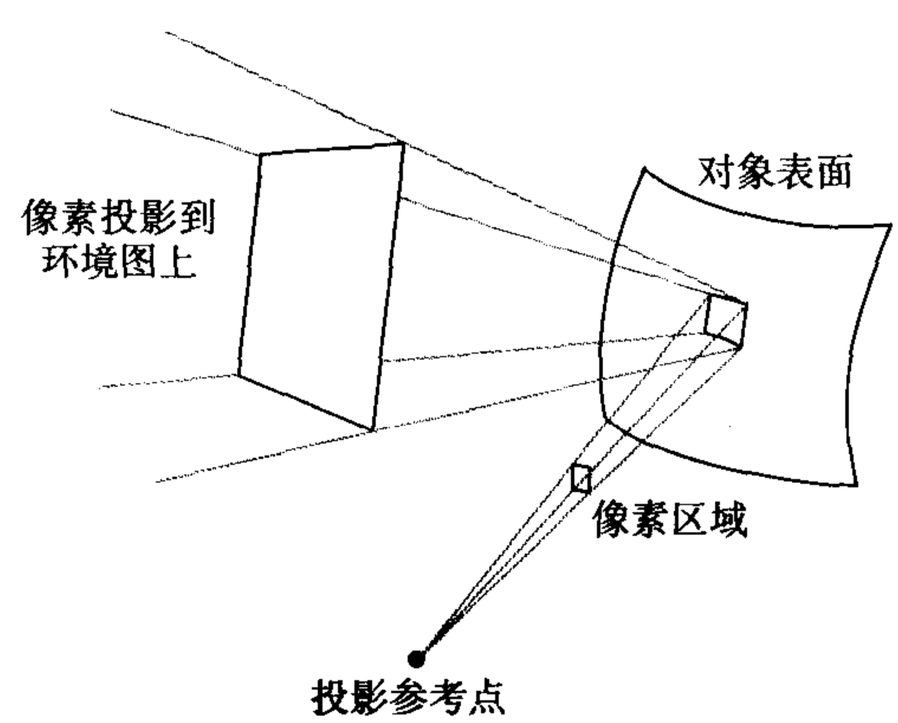


图10.98 将一像素区域投影至对象表面，然后将该区域反射至环境映射中

10.14 光子映射

尽管辐射度方法可以为简单场景生成全局照明效果的精确显示，但随着场景复杂性的增加，该方法越来越难以应用。对于非常复杂的场景，绘制时间和存储量均无法承受，因而许多光照效果很难正确模拟。**光子映射**（photon mapping）提供了模拟复杂场景全局照明的一般方法，该方法既高效又精确。

光子映射的基本概念是将照明信息与场景的几何数据分开。光线路径从所有光源到场景的方向进行跟踪，而光线和对象交点处的光照信息存储在**光子图**（photon map）中。然后通过和辐射度绘制中类似的增量算法来使用分布式光线跟踪方法。

光源可指定为点光源、方向投射光源或任一其他类型。赋给光源的强度分配给所有光线（光子），而光线方向随机地分布。一个点光源生成均匀地向各方向发散的光线路径，除非该光源是方向光源（参见10.1节）。对于其他光源，选择光源上的随机位置来生成随机方向的光线。亮的光源比能量少的光源生成更多的光线。另外，可为光源建立一张用来存储是否在空间某处有对象的二值信息的投影图（projection map）。也可以使用球体来提供大空间范围的对象信息。为一个场景可生成任意多的光线，产生的光线路径越多，则照明效果越精确。

10.15 添加表面细节

我们已经讨论过光滑对象表面的绘制技术。然而，大多数对象的表面并不光滑，甚至有些曲面也不光滑。我们需要在对象表面添加纹理以准确地模拟砖墙、砂砾小路、粗线地毯、木头和人的皮

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

肤等对象。另外，在绘制过程中必须考虑一些对象表面的图案花纹。例如，花瓶的表面可能有一幅图画，水杯上可能刻有饰纹，网球场上可能会有球道、服务区域和基准线的标记，四车道的高速公路上有分道线和溢出的油、刹车留下的轮胎印等标记。

图 10.99 给出了模拟和绘制一个带有表面细节的对象。首先，显示了该对象的线框图用于调整设计。接着，在对象轮廓上拟合各层表面来生成绘制的光滑曲面结构视图。然后，在外层添加表面细节。对于图 10.99 的例子，表面细节包括茄克衫图案，如线缝、印字、服装纹理，以及皮肤特征如毛孔、痣、雀斑等。计算机生成的一个人物的皮肤特征的放大视图在图 10.100 中给出，而图 10.101 给出了对一个老人皮肤特征的模拟。图 10.102 则给出了另外一些用表面细节绘制的场景示例。

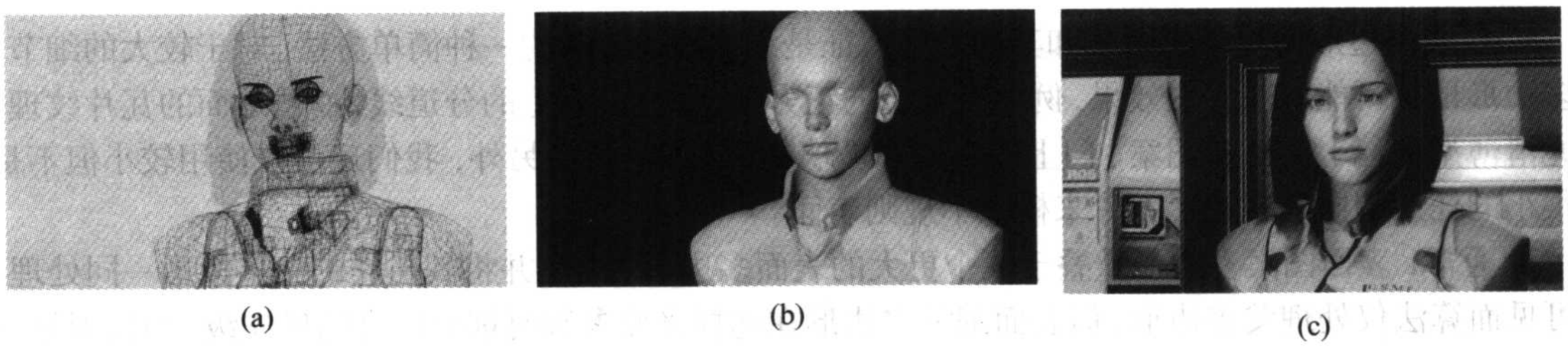


图 10.99　电影 *Final Fantasy: The Spirits Within* 的动画人物 Aki Ross 博士开发中的建模和绘制阶段：(a) Aki 的线框模型；(b) 皮肤和衣服的表面结构；(c) 最终的绘制图，包括头发和皮肤特征及服装的细节（Square Pictures 公司提供）

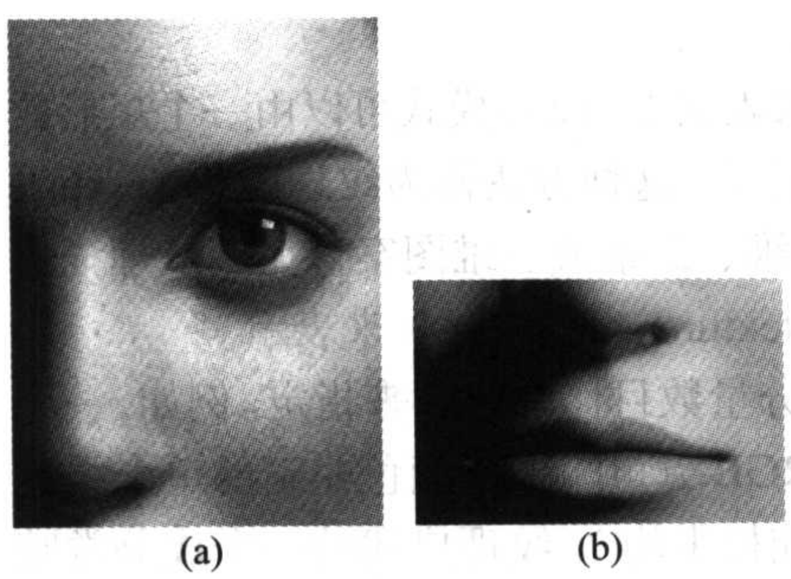


图 10.100　电影 *Final Fantasy: The Spirits Within* 中动画人物 Aki Ross 博士的皮肤细节（Square Pictures 公司提供）

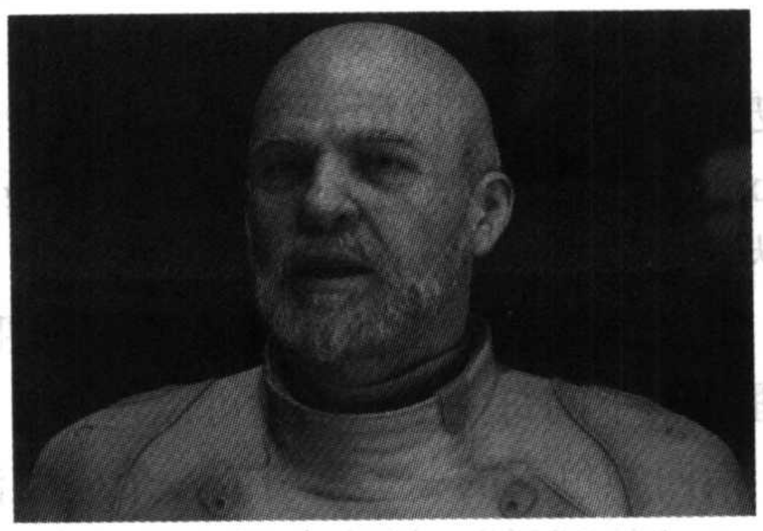


图 10.101　电影 *Final Fantasy: The Spirits Within* 中动画人物 Sid 博士的最终特征和表面皮肤纹理，表现了一个 70 岁的老人（Square Pictures 公司提供）

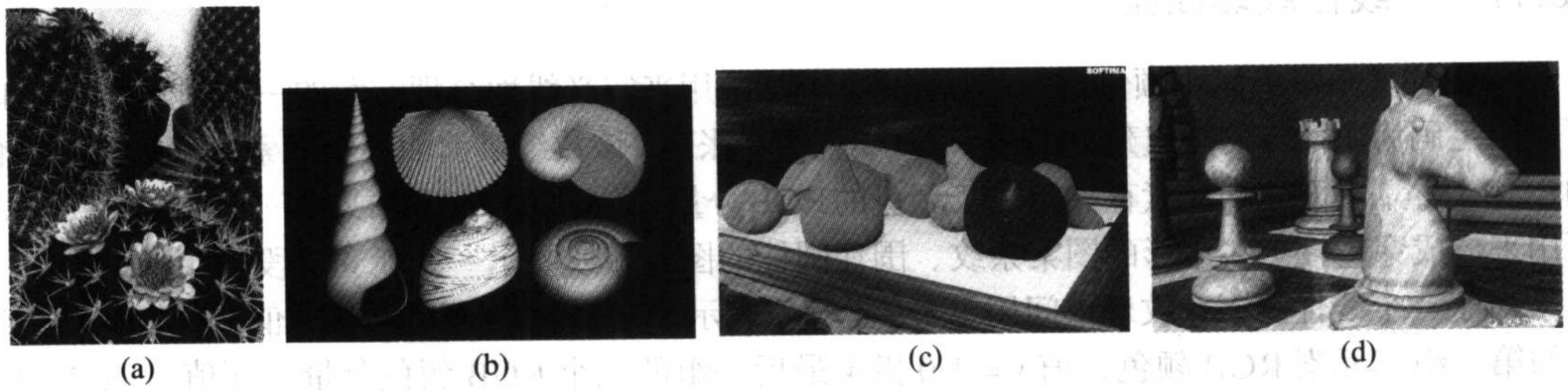


图 10.102　展示用计算机图形学为各种对象生成表面细节的场景：(a)带有刺和花的仙人掌（Calgary 大学的 Deborah R. Fowler、Przemyslaw Prusinkiewicz 和 Johannes Battjes 提供）；(b)有各种图案和凹槽表面的海洋贝壳（Calgary 大学的 Deborah R. Fowler. Hans Meinhardt 和 Przemyslaw Prusinkiewicz 提供）；(c)满桌水果（SOFTIMAGE 公司提供）；(d)用纹理映射方法生成的棋子和棋盘的表面图案（SOFTIMAGE 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们可以使用多种方法添加表面细节，包括

- 在较大的表面上粘贴蓓蕾、花朵或荆棘等小东西。
- 用小的多边形区域建立表面图案的模型。
- 将纹理数组或强度修改过程映射到表面。
- 通过修改表面法向量来生成局部凹凸。
- 通过同时修改表面法向量和表面切向量来显示木头和其他材料上的有向图案。

10.16 用多边形模拟表面细节

使用多边形面片模拟图案和其他表面特征是添加表面细节的一种简单方法。对于较大的细节，多边形模拟可能有较好的效果，例如棋盘上的正方形、高速公路上的分道线、油毡地面的瓦片纹理、光滑薄型地毯上的鲜花图案、门上的网格和玻璃货车表面的字。另外，我们还可以使用较小但不是很小的任意朝向的多边形面片来模拟不规则表面。

通常，表面图案多边形重叠于一个更大的表面多边形之上，并将其与它的父多边形一同处理。可见面算法仅处理父多边形，但表面细节多边形在选择光照参数时都有较高的优先级。当模拟复杂的或精致的表面细节时，多边形方法则不可行。例如，使用多边形表面来准确模拟葡萄干的表面结构是很困难的。

10.17 纹理映射

一个常用的添加表面细节的方法是将纹理模式映射到对象表面上。纹理模式可以由一个矩形数组进行定义，也可以用一个修改对象表面光强度值的过程来定义，这种方法称为**纹理映射**（texture mapping）或**图案映射**（pattern mapping），而纹理可定义成一维、二维或三维图案。任一纹理描述称为**纹理空间**（texture space），用0到1.0范围的**纹理坐标**（texture coordinate）来表示。

图形软件包中的纹理函数经常将图案中每一位置的颜色分量数目作为选项来指定。例如，一个纹理图案中的每一颜色描述可以包含4个RGBA分量、3个RGB分量、用于蓝色明暗的单个强度值、指向颜色表的一个索引或一个亮度值（一个颜色的RGB加权平均）。纹理描述的一个成分常暗指一个“纹理元”，但该术语使用时会有一定混淆。有时与一组颜色分量对应的纹理空间的一个位置（如一个RGB三角形）称为一个纹理元；而有时单个纹理数组元素，如RGB颜色中红色分量的值，也称为一个纹理元。

10.17.1 线性纹理图案

一维纹理图案可以用颜色值的单足码数组指定，用来定义线性纹理空间的一系列颜色。例如，可以用足码为0到95的数组建立32个RGB颜色的一张表。该数组最前面三个元素存储第一个颜色的RGB分量，下面三个元素存储第二个颜色的RGB分量，等等。这组颜色或任一连续的颜色子集，可用来形成穿越一个多边形的图案条纹、围住圆柱的图案带或显示一条孤立线段的颜色图案。

对于一个线性图案，纹理空间用单个s坐标值表示。对于RGB颜色描述，值$s = 0.0$指定数组中的第一组三元素RGB颜色，值$s = 1.0$指定最后一组的三个RGB颜色分量，而值$s = 0.5$指定该数组的中间的三个RGB颜色分量。作为一个例子，如果该纹理数组的名字是colorArray，则值$s = 0.0$指向三个数组元素colorArray[0]、colorArray[1]和colorArray[2]。

要将一个线性纹理图案映射到场景中，需要将一个s坐标值赋以一个空间位置而另一个s坐标值赋以另一空间位置。对应于指定s坐标范围的颜色数组部分用来生成这两个空间位置间的多色线。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

纹理映射过程一般使用线性函数来计算沿线段赋给像素的数组位置。当为该线段指定的纹理颜色数目较小时，每一个颜色赋给一个大的像素块，其大小根据线段长度而定。例如，当指定的 s 坐标范围跨越纹理数组中的单个RGB颜色（三个RGB颜色分量）时，则线段上所有像素均用该颜色显示。但当许多颜色映射到线段上时，每一颜色赋给若干个像素。同样，由于有些像素可能映射到RGB颜色间的数组位置，因此有多种方案可用来确定赋给每一像素的颜色。一种简单的颜色映射方法是将最近的数组颜色赋给每一像素。另一方法是，当一个像素映射到两个颜色的开始数组位置时，可用数组中两相邻颜色的线性混合来计算该像素颜色。

有些纹理映射过程允许其纹理坐标值超出0到1.0的范围。这些情况可能发生在需要将多份纹理映射到一个对象上或 s 坐标的计算值超出单位区间。如果允许纹理坐标值超出0到1.0的范围，则可忽略任一 s 值的整数部分。例如，值 -3.6 表示的纹理空间与0.6或12.6相同。但如果不允许超出0到1.0的范围，则将小于0的值设定为0而将大于1.0的值设定为1.0。

10.17.2　表面纹理图案

用于表面区域的纹理通常用矩形颜色图案定义，而在这一纹理空间的位置用二维 (s, t) 坐标值来指定。纹理图案中每一颜色的描述可存储在一个三足码数组中。例如，当一个纹理图案用 16×16 个RGB颜色定义时，该图案的数组包含 $16 \times 16 \times 3 = 768$ 个元素。

图10.103给出了一个二维纹理空间。s 和 t 的值在0到1.0之间变化。数组的第一行列出该矩形图案底部的颜色值，而数组的最后一行列出该图案顶部的颜色值。纹理空间坐标位置(0, 0)指向第一行第一个位置的第一组颜色分量，而位置(1.0, 1.0)则指向该数组最后一行最后一个位置的最后一组颜色分量。当然，该纹理数组还可以用另外的方法列出颜色。如果按自顶向下的次序列出颜色，则二维纹理空间的原点将在矩形图案的左上角。但是将纹理空间原点安排在左下角一般会简化场景中空间坐标系的映射过程。

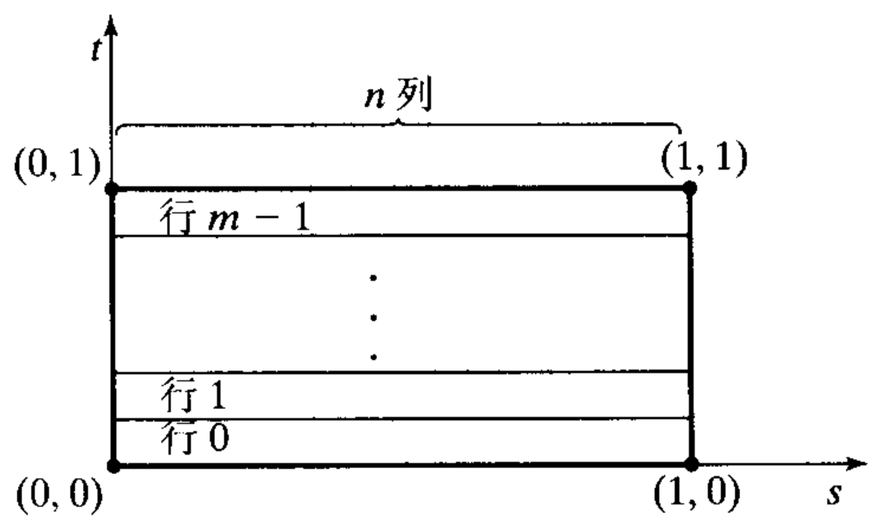


图 10.103　指向包含 m 行 n 列颜色值数组位置的二维纹理坐标。数组中每一位置可指定多个颜色分量

描述一对象表面纹理映射的过程和指定场景线性纹理映射的过程相同。纹理图案（参见图10.103）四角的 (s, t) 纹理空间坐标可赋给场景的四个空间位置，且使用线性变换将颜色值赋给指定空间区域的投影位置。也可以进行另外的映射。例如，可将三个纹理空间坐标赋给一个三角形。

三次样条面片或球面部分等对象的表面位置可用 uv 对象空间坐标来描述，而投影像素位置用笛卡儿坐标 xy 来指定。表面纹理映射可用两种方法之一来实现：将纹理映射到对象表面上，再到投影平面；或将像素区间映射到对象表面，再将该表面区域映射到纹理空间。将一个纹理图案映射到像素坐标有时也称为纹理扫描（texture scanning），而将像素坐标到纹理空间的映射称为像素次序扫描（pixel-order scanning）、逆扫描（inverse scanning）或图像次序扫描（image-order scanning）。图10.104显示了三个空间之间的两种可能的变换序列。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

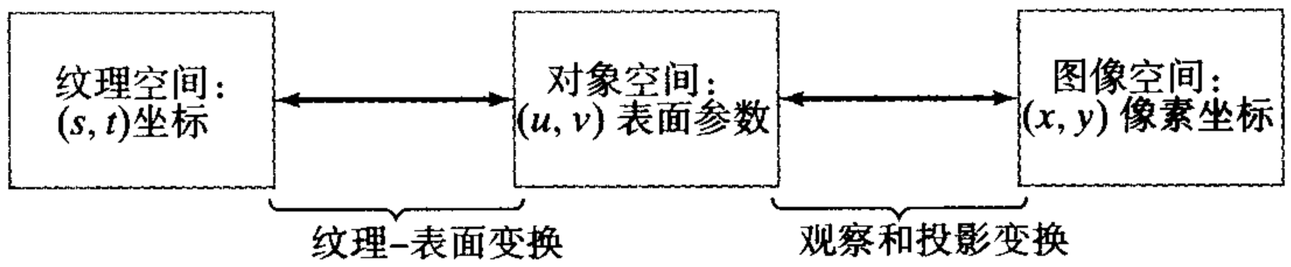


图 10.104　二维纹理空间、对象空间和图像空间的坐标系统

参数线性变换为从纹理空间到对象空间的映射提供了一个简单方法：

$$\begin{aligned} u &= u(s,t) = a_u s + b_u t + c_u \\ v &= v(s,t) = a_v s + b_v t + c_v \end{aligned} \tag{10.109}$$

对象到图像空间的变换通过合并观察和投影变换来实现。从纹理空间到像素空间映射的缺点是选定的纹理片通常不能与需要计算来确定像素覆盖部分的像素边界完全匹配。因此，从像素空间到纹理空间（参见图 10.105）的映射是常用的纹理映射方法。这避免了像素分割的计算，并且容易使用反走样（滤波）处理。一种有效的反走样处理是将图中所示覆盖到邻近像素中心的稍大一些的像素投影，并应用一个四棱锥函数给纹理图案中的强度值加权。但是从图10.106像空间到纹理空间的映射不需要计算观察和投影变换的逆变换$\mathbf{M}_{VP}^{-1}$和纹理映射变换的逆变换$\mathbf{M}_T^{-1}$。下面的例子通过将一个定义的图案映射到圆柱表面来表示这一方法。

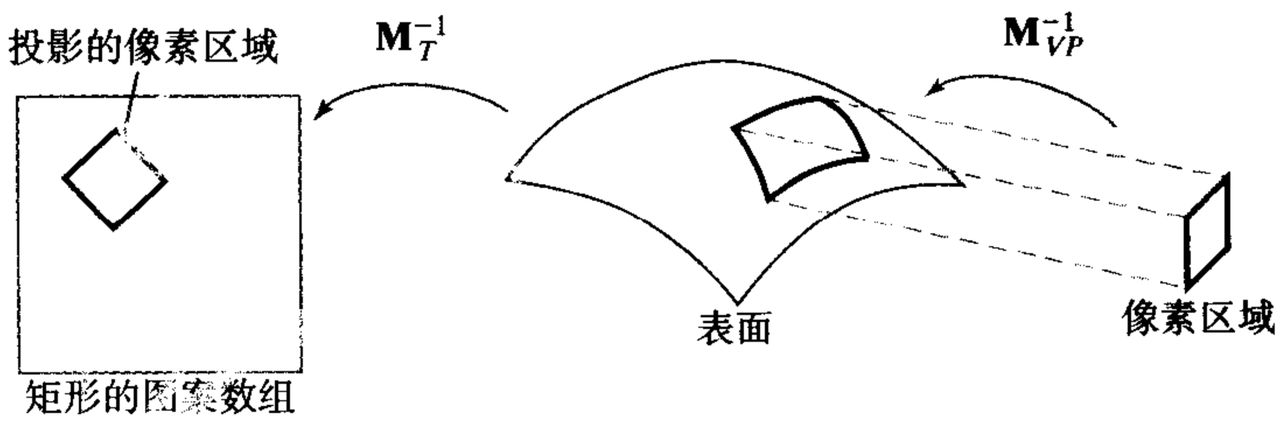


图 10.105　将像素区域投影到纹理空间的纹理映射

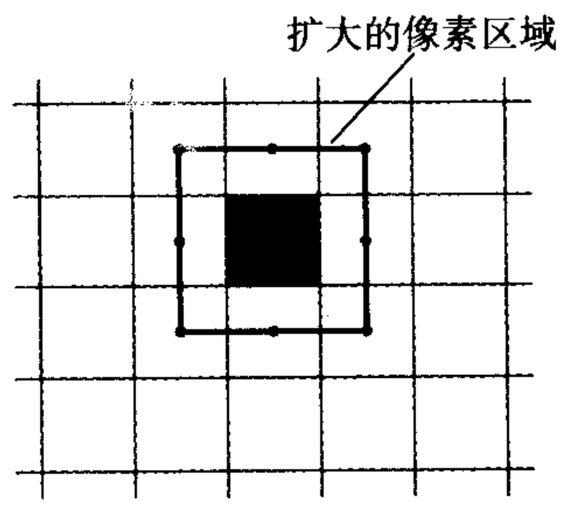


图 10.106　包含相邻像素中心位置的扩大的像素

例 10.1　表面纹理映射

为了展示表面纹理映射的步骤，考虑将图10.107(a)中的图案到圆柱表面上的变换。表面参数是圆柱坐标：

$$u = \theta, \qquad v = z$$

及

$$0 \leqslant \theta \leqslant \pi/2, \qquad 0 \leqslant z \leqslant 1$$

在笛卡儿坐标系中的参数表示为

$$x = r\cos u, \qquad y = r\sin u, \qquad z = v$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

可以由以下线性变换将图案数组映射至对象表面，其中，图案的原点$(s, t) = (0, 0)$将映射至表面元素$(x, y, z) = (r, 0, 0)$的左下角。

$$u = s\pi/2, \qquad v = t$$

接下来，选择一个观察位置并执行从像素坐标到圆柱笛卡儿坐标系的逆观察变换。然后，用下列计算将笛卡儿表面坐标变换到 uv 表面参数：

$$u = \tan^{-1}(y/x), \qquad v = z$$

而投影像素位置用下列逆变换映射到纹理空间：

$$s = 2u/\pi, \qquad t = v$$

将每一投影像素区域覆盖的图案数组颜色值平均后赋给像素。

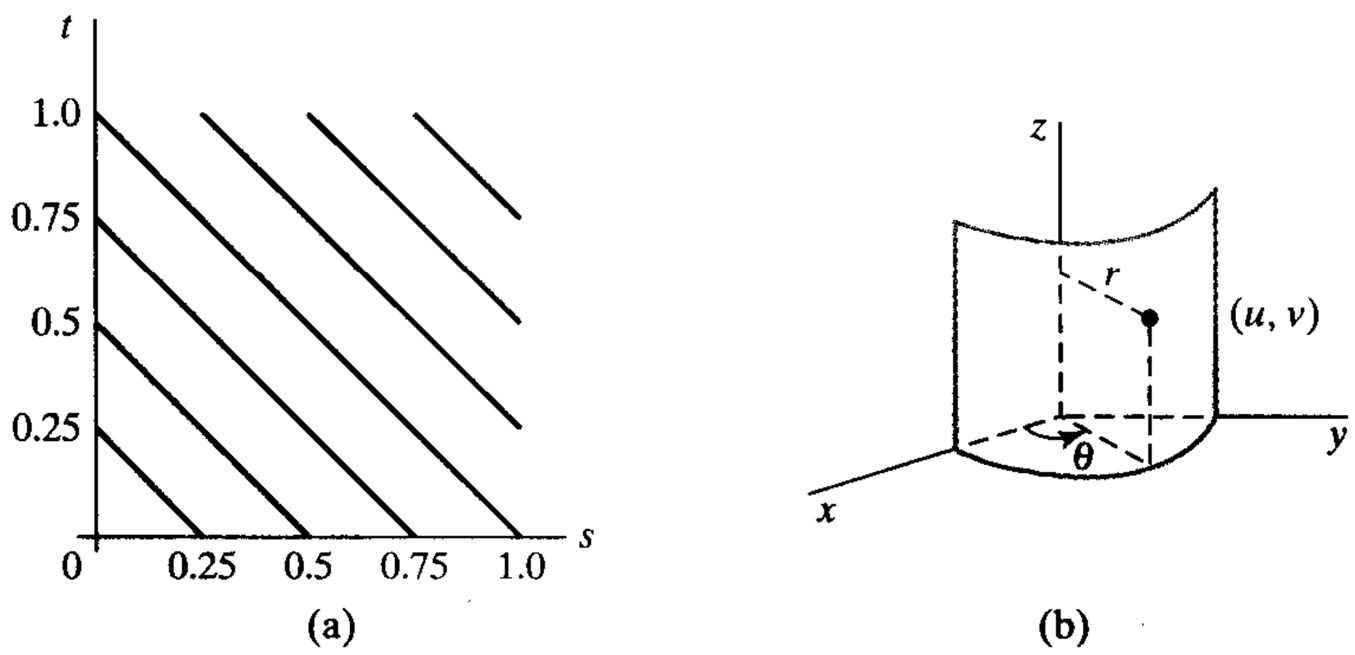


图 10.107　单位正方形(a)中定义的纹理图案映射到圆柱表面(b)

10.17.3　体纹理图案

除了线性和表面图案之外，我们还可为空间三维区域的位置指定一组颜色。这些纹理常称为**体纹理图案**（volume texture pattern）或**实体图案**（solid texture）。实体纹理通过三维纹理空间(s, t, r)来指定。而一个三维纹理空间在单位立方体内定义，其纹理坐标范围为 0 到 1.0。

一个体纹理图案可用一个四足码数组来存储，其中最前面三个足码给出行位置、列位置和深度位置。第四个足码用来指向图案中的特定颜色。例如，一个 16 行、16 列且 16 个深度平面的 RGB 实体纹理可存储在一个有 16 × 16 × 16 × 3 = 12 288 个元素的数组中。

为了将整个纹理空间映射到一个三维块，将该纹理空间的八个角的坐标赋给场景的空间位置。或者可以将纹理空间的一个平面部分（如一个深度平面或该纹理立方体的一个面）映射到场景中的一个平面区域。也可以有其他的实体纹理映射应用。

实体纹理可提供内部视图，如剖切显示和切片，使三维对象的显示中带有纹理图案。这样，砖块、炉渣或木头材料可以有相同的纹理图案应用于对象的所有空间范围中。图10.108给出了使用实体纹理获得的木纹和其他材料图案的场景显示。

10.17.4　纹理缩减图案

在动画和其他应用中，对象大小经常改变。对于用纹理图案显示的对象，在其尺寸改变时要重新使用纹理映射。当一个有纹理的对象缩小时，原纹理图案应用于较小的区域会导致纹理变形。为避免变形，可创建一系列**纹理缩减图案**（texture reduction pattern）在对象显示尺寸缩小时使用。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

一般，每一缩减图案是前一图案的一半。例如，如果已有一个 16 × 16 的图案，则可建立尺寸为 8 × 8、4 × 4、2 × 2 和 1 × 1 这样四个另外的图案。对于一个对象的任一视图，就可应用合适的缩减图案使其变形最小。这些缩减图案常称为 **MIP 图**或 **mip 图**，其中术语 mip 是拉丁语 multum in parvo 的缩略，其意可译为“小对象上的许多”。

10.17.5 过程式纹理映射方法

另一种添加表面纹理的方法是将过程式定义应用于场景中对象的颜色变量。该方法避免了将二维纹理图案映射到对象表面所需的变换计算。过程式纹理映射消除了对存储容量的需求，而这在场景中使用许多大的纹理图案、特别是实体纹理时是必需的。

我们可以通过计算表示一个对象的材质或特征的变量来生成一个过程纹理。例如，可用在三维空间区域定义的调和函数（正弦曲线）在整个对象上建立木纹或大理石图案，然后再用随机扰动添加到调和变量上来打乱对称的图案。图 10.109 中的场景用砖石建筑、抛光的金器和香蕉叶等表面图案的过程描述绘制而成。

图 10.108 一个使用实体纹理方法的对象特征建模的场景（犹他大学计算机科学系的 Peter Shirley 提供）

图 10.109 使用 VG Shaders 绘制并使用 RenderMan 建模的场景，其中使用了宝石型的多边形面片、二次曲面和双三次曲面片。除了绘制表面纹理，还使用了过程式方法模拟了丛林中的雾气和光照效果（VALIS Group 提供）

10.18 凹凸映射

虽然纹理映射可用于添加精致的表面细节，但它对于模拟粗糙的物体表面，如橘子、草莓和葡萄干等则不合适。纹理图案的光照细节的设定通常与场景中的光照方向无关。生成物体表面凹凸效果的较好的方法是使用扰动函数并在光照模型计算中使用扰动法向量，该技术称为**凹凸映射**（bump mapping）。

如果 $\mathbf{P}(u,v)$ 表示一个参数曲面上的点，可以通过如下计算得到该点处的表面法向量：

$$\mathbf{N} = \mathbf{P}_u \times \mathbf{P}_v \tag{10.110}$$

其中，$\mathbf{P}_u$ 与 $\mathbf{P}_v$ 为 $\mathbf{P}$ 关于参数 u 和 v 的偏导数，为了得到扰动法向量，我们可在表面点的向量方向上增加一个小的扰动函数，称为凹凸函数：

$$\mathbf{P}'(u, v) = \mathbf{P}(u, v) + b(u, v)\,\mathbf{n} \tag{10.111}$$

它在表面单位法向量 $\mathbf{n} = \mathbf{N}/|\mathbf{N}|$ 方向上增加凹凸效果，扰动后的表面法向量为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{N}' = \mathbf{P}'_u \times \mathbf{P}'_v \tag{10.112}$$

可以根据扰动向量 $\mathbf{P}'$ 的 u 分量来计算偏导数：

$$\begin{aligned}\mathbf{P}'_u &= \frac{\partial}{\partial u}(\mathbf{P} + b\mathbf{n}) \\ &= \mathbf{P}_u + b_u\mathbf{n} + b\mathbf{n}_u\end{aligned} \tag{10.113}$$

假定凹凸函数 b 很小，则可以忽略最后一项而得到

$$\mathbf{P}'_u \approx \mathbf{P}_u + b_u\mathbf{n} \tag{10.114}$$

同样：

$$\mathbf{P}'_v \approx \mathbf{P}_v + b_v\mathbf{n} \tag{10.115}$$

扰动的表面法向量为

$$\mathbf{N}' = \mathbf{P}_u \times \mathbf{P}_v + b_v(\mathbf{P}_u \times \mathbf{n}) + b_u(\mathbf{n} \times \mathbf{P}_v) + b_u b_v(\mathbf{n} \times \mathbf{n})$$

因为 $\mathbf{n} \times \mathbf{n} = 0$，所以

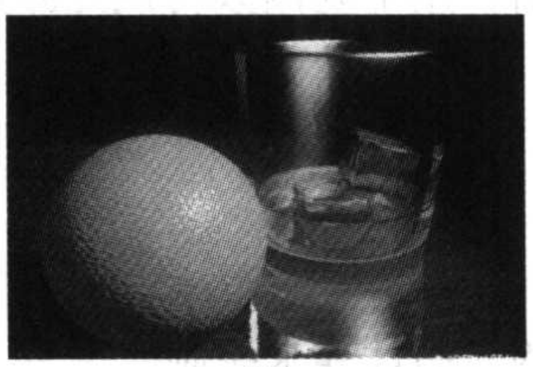
$$\mathbf{N}' = \mathbf{N} + b_v(\mathbf{P}_u \times \mathbf{n}) + b_u(\mathbf{n} \times \mathbf{P}_v) \tag{10.116}$$

最后一步是规范化 $\mathbf{N}'$，以便于在光照模型中计算。

我们可以使用几种方法来指定凹凸函数 $b(u,v)$。例如定义一个解析表达式，但如果使用查表法获得凹凸值则可减少计算。根据凹凸表，可以由线性插值和增量法很快得到 b 的值，然后，可以由有限次差分得到偏导数 b_u 和 b_v 的近似值。凹凸表则由随机图案、规则网格图案或文字形状来建立。对于不规则物体表面的模拟，例如葡萄干，随机模式是很有效的；而对于模拟橘子表面，重复模式则更有效。为了进行反走样，我们可将像素区域进行细分，并取子像素强度的平均值。

图10.110表示利用凹凸映射绘制的物体表面的例子。图10.111给出了一种结合式的表面绘制算法的例子。电影 *Young Sherlock Holmes* 中彩色玻璃骑士的盔甲混合使用了凹凸映射、环境映射和纹理映射等方法进行绘制。周围物体的环境映射与凹凸映射相结合以生成背景光照反射和粗糙的表面。该图中还添加了附加色彩和表面光照、气泡、污点、接缝处和柳钉的锈色等，从而产生出全局效果。

(a)



(b)

图 10.110　使用凹凸映射方法绘制表面粗糙的物体（印第安纳大学计算机系的 Peter Shirley 与 SOFTIMAGE 公司提供）

图 10.111　动画 *Young Sherlock Holmes* 中的彩色玻璃骑士，在绘制盔甲表面时结合凹凸映射、环境映射和纹理映射（Industrial Light & Magic 提供）

10.19　帧映射

这种增加表面细节的方法是凹凸映射的延伸。在**帧映射**（frame mapping）中，我们同时扰动表面法向量 $\mathbf{N}$ 和一个与 $\mathbf{N}$ 相关的本地坐标系统（参见图 10.112），由表面切线向量 $\mathbf{T}$ 和双法线向量 $\mathbf{B} = \mathbf{T} \times \mathbf{N}$ 可以定义本地坐标系。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

帧映射可用于建立非等值的曲面。我们沿着表面的纹理确定T的方向并进行方向扰动，同时还要在N方向上进行凹凸扰动。这样，就可以模拟木纹图案、布上的穿线图案和大理石或类似材质上的条纹。凹凸和方向扰动可以通过查找表得到。

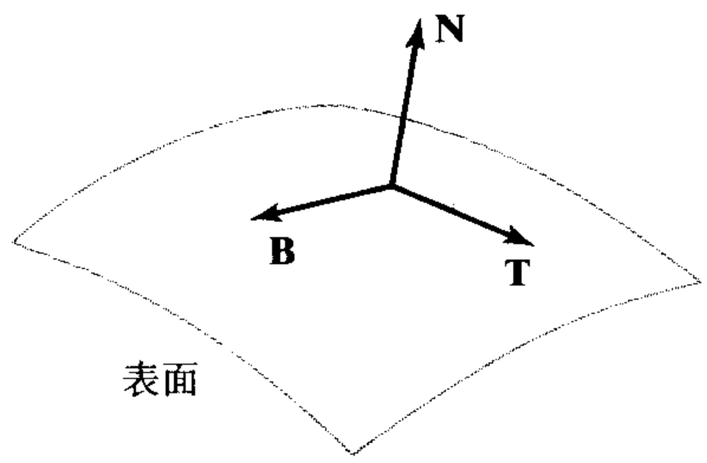


图 10.112 一个表面点的本地坐标系

10.20 OpenGL 光照和表面绘制函数

OpenGL有多个用于在基本光照模型中设定点光源、选择表面反射系数和选择其他参数值的函数。另外，我们可以模拟透明性，可用平面绘制或 Gouraud 表面绘制来显示对象。

10.20.1 OpenGL 点光源函数

OpenGL 场景描述中可包括多个点光源，而各种特性如位置、类型、颜色、衰减和投射效果等与每一个点光源结合在一起。为一个点光源设定特性的函数是

```
glLight* (lightName, lightProperty, propertyValue);
```

后缀码 i 或 f 附加在函数名之后，按特性值的数据类型来确定。对于向量数据，还要加上后缀码 v，而这时的参数propertyValue是一个指向数组的指针。每一光源用一个标识符引用，而参数lightName用 OpenGL 符号标识 GL_LIGHT0，GL_LIGHT1，GL_LIGHT2，...，GL_LIGHT7 来赋值，尽管有些 OpenGL 的实现允许使用超过 8 个光源。类似地，参数 lightProperty 必须用 10 个 OpenGL 符号特性常量之一赋值。在一个光源的所有特性均赋值后，用下列函数将该光源点亮。

```
glEnable (lightName);
```

然而，还需激活 OpenGL 光源，即调用

```
glEnable (GL_LIGHTING);
```

接着对象表面就用包括已激活的每一光源贡献的光照计算来绘制。

10.20.2 指定一个 OpenGL 光源位置和类型

用于指定光源位置的 OpenGL 符号特性常量是 GL_POSITION。实际上，这一符号常量用来同时设定两个光源特性：光源位置和光源类型（light-source type）。OpenGL 中用于照明场景的光源有两个基本类。一个点光源可能属于接近被照明对象的点光源，或属于离场景无穷远。而该分类与指定的光源位置无关。对一个近光源来说，发出的光向所有方向辐射，且该光源位置包括在光照计算中。但从远光源发出的光仅允许向一个方向发射，该方向应用于场景中的所有表面，与指定的光源位置无关。远光源发出的光线方向按从指定光源位置到坐标原点的方向来计算。

光源类型和光源位置坐标值用一个四元素浮点数向量来指定。该向量的前三个元素给出世界坐标位置，而第四个元素用来指定光源类型。如果对该位置向量的第四个元素赋值 0.0，则该光源是

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

非常远的光源（在 OpenGL 中称为方向光源），而光源位置则用来确定光线方向。否则，将该光源当做一个局部光源（在 OpenGL 中称为位置光源），而光源位置被光照子程序用来确定对场景中每一对象的光照方向。在下面的示例程序中，光源 1 按位置(2.0, 0.0, 3.0)指定为局部光源，而光源 2 是一个在负 y 方向发射光线的远光源。

```
GLfloat light1PosType [ ] = {2.0,0.0,3.0,1.0};
GLfloat light2PosType [ ] = {0.0,1.0,0.0,0.0};

glLightfv (GL_LIGHT1, GL_POSITION, light1PosType);
glEnable (GL_LIGHT1);

glLightfv (GL_LIGHT2, GL_POSITION, light2PosType);
glEnable (GL_LIGHT2);
```

如果不指定光源的位置和类型，则使用默认值(0.0, 0.0, 1.0, 0.0)，表示一个在负 z 方向发射光线的远光源。

光源的位置包含在场景描述中，并和对象位置一起用OpenGL几何变换及观察变换矩阵变换到观察坐标系中。因此，如果希望光源相对于场景中对象的位置保持不变，则在程序中设定几何和观察变换之后再设定其位置。但如果希望光源随着观察点一起移动，则在几何和观察变换之前描述其位置。我们还可以让光源相对于固定场景平移或旋转。

10.20.3　指定 OpenGL 光源颜色

与实际光源不同的是，OpenGL有三个不同的RGBA颜色特性。在此经验方法中这三种光源颜色提供了对场景中各种光照效果的选择。我们使用符号颜色特性常量 GL_AMBIENT、GL_DIFFUSE 和 GL_SPECULAR 来设定这些颜色。其中每一个通过指定一个四元素浮点值集来赋值。每一颜色的分量按次序(R, G, B, A)指定，分量 A 仅用于颜色混和函数激活以后。正如这些符号颜色特性常量的名称所表达的那样，光源颜色之一对场景的背景（环境）光有贡献，另一种颜色用于漫反射光照计算，而第三种颜色用于计算场景的镜面反射光照效果。实际上，一个光源只有一种颜色，但我们可使用三个 OpenGL 光源颜色来建立各种光照效果。在下面的示例程序中，设定一个局部光源的环境光颜色为黑色，其标号为 GL_LIGHT3，且设定该光源的漫反射和镜面反射颜色为白色。

```
GLfloat blackColor [ ] = {0.0,0.0,0.0,1.0};
GLfloat whiteColor [ ] = {1.0,1.0,1.0,1.0};

glLightfv (GL_LIGHT3, GL_AMBIENT, blackColor);
glLightfv (GL_LIGHT3, GL_DIFFUSE, whiteColor);
glLightfv (GL_LIGHT3, GL_SPECULAR, whiteColor);
```

光源0的默认颜色是环境光为黑色而漫反射和镜面反射为白色。所有其他光源的环境光、漫反射及镜面反射颜色特性均为黑色。

10.20.4　指定 OpenGL 光源的光线强度衰减系数

我们可以考虑 OpenGL 局部光源发出的光线强度衰减问题，而 OpenGL 光照子程序使用等式（10.2）在光源位置距对象为 d_l 时计算这一衰减。光线强度衰减的三个 OpenGL 特性常量是 GL_CONSTANT_ATTENUATION、GL_LINEAR_ATTENUATION和GL_QUADRATIC_ATTENUATION，分别对应于等式（10.2）中的系数 a_0、a_1 和 a_2。每一个衰减系数用正整数或正浮点数来设定。例如，我们将光线强度衰减系数设为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glLight (GL_LIGHT6, GL_CONSTANT_ATTENUATION, 1.5);
glLight (GL_LIGHT6, GL_LINEAR_ATTENUATION, 0.75);
glLight (GL_LIGHT6, GL_QUADRATIC_ATTENUATION, 0.4);
```

在衰减系数的值设定后，光线衰减函数应用于该光源的所有三种颜色（环境光、漫反射和镜面反射）。衰减系数的默认值是 $a_0 = 1.0$、$a_1 = 0.0$ 和 $a_2 = 0.0$。这样，使用默认值就是不衰减，$f_{l,\text{radatten}} = 1.0$。尽管光线衰减可生成更具真实感的显示，但其计算十分耗时。

10.20.5 OpenGL 方向光源（投射光源）

对于局部光源（不在无穷远处的光源），还可指定方向或投射效果。这将光线限制在一个圆锥形的空间中。我们用圆锥轴上的一个方向向量和一个从该轴开始的角度范围 θ_l 定义该圆锥区域，如图10.113所示。另外，可指定一个该光源的角衰减指数 a_l 来确定在从圆锥中心向其表面移动时减少多少光强度。沿光锥任一方向的角衰减因子是 $\cos^{a_l}\alpha$，参见等式（10.5），这里用该圆锥轴向量和光源到对象位置的向量的点积来计算。我们用强度分量与角衰减因子相乘来依次获得环境光、漫反射和镜面反射光颜色。如果 $\alpha > \theta_l$，则对象在光源锥体之外，该光源设有照明这一对象。对于在圆锥之内的光线，还需考虑光强度发散时的衰减。

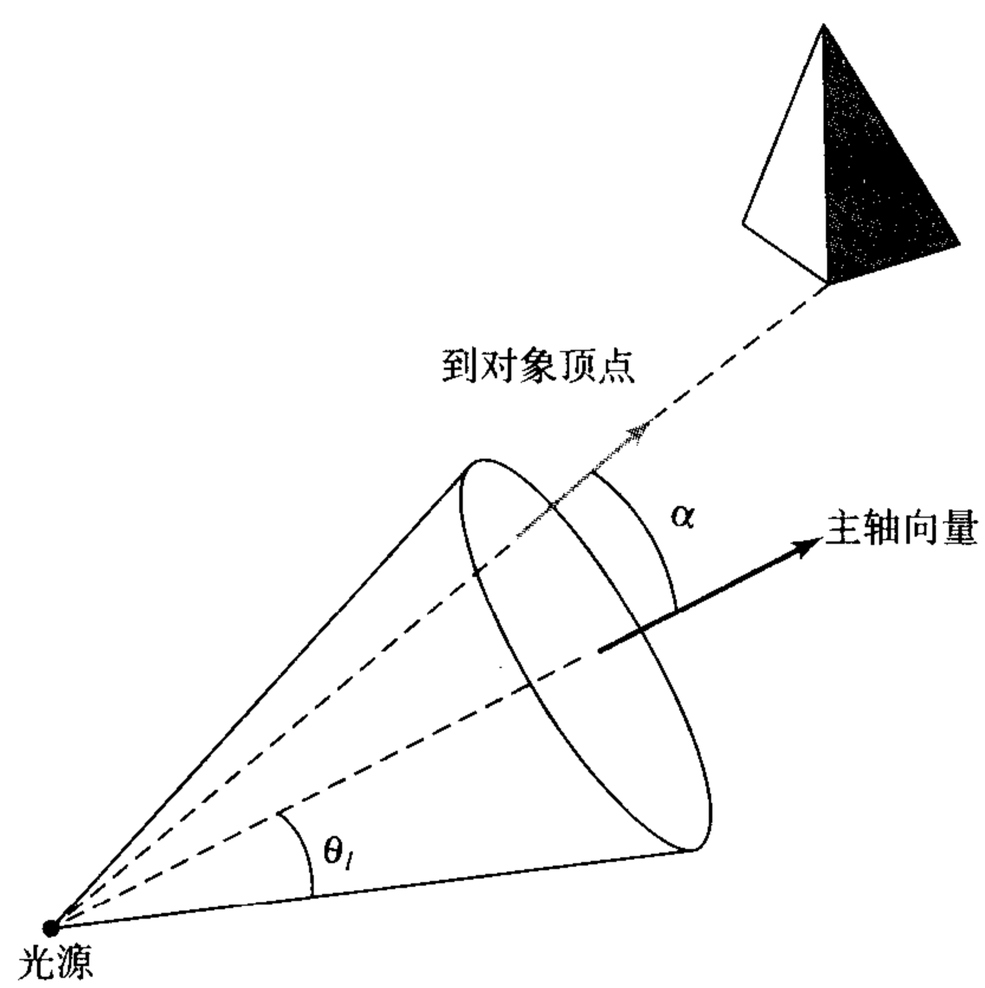


图 10.113 一个从光源发散的光锥。从圆锥轴开始测量的该光锥的角度范围是 θ_l，而从该轴到一个对象的方向向量的夹角是 a

OpenGL 方向效果的特性常量有三个：GL_SPOT_DIRECTION、GL_SPOT_CUTOFF 和 GL_SPOT_EXPONENT。我们可用整数或浮点数向量来指定光的方向。圆锥角 θ_l 用整数或浮点数的度数来给出，该角可以是 180° 或 0° 到 90° 范围内的任意值。当圆锥角设定为 180° 时，该光源向所有方向（360°）发出光线。我们为强度衰减设定从 0 到 128 的整数或浮点数指数值。下列语句为光源 3 设定方向效果，其圆锥轴在正 x 方向，圆锥角 θ_l 为 30°，而衰减因子是 2.5。

```
GLfloat dirVector [ ] = {1.0,0.0,0.0};

glLightfv (GL_LIGHT3, GL_SPOT_DIRECTION, dirVector);
glLightf (GL_LIGHT3, GL_SPOT_CUTOFF, 30.0);
glLightf (GL_LIGHT3, GL_SPOT_EXPONENT, 2.5);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如果不指定光源的方向，默认方向是平行于负 z 轴方向；即（0.0, 0.0, -1.0）。同样，默认的圆锥角是 180°，而默认的衰减指数为 0。因此，使用默认值时该光源为向各方向发射光线的点光源，没有角衰减。

10.20.6　OpenGL 全局光照参数

可在全局级指定若干个OpenGL光照参数。这些参数可用来控制某些光照计算的实现方法，且每一个全局参数用下列函数设定。

```
glLightModel* (paramName, paramValue);
```

按参数值的数据类型来确定添加后缀码 i 或 f。对向量数据，还要添加后缀码 v。参数 paramName 赋以 OpenGL 一个标识设定的全局特性的符号常量，而参数 paramValue 赋以一个或一组值。使用 glLightModel 函数，可设定全局环境光的等级、指定如何计算镜面高光及选择应用于多边形后向面的光照模型。

除了单个光源的环境光，还可将OpenGL背景光设定为全局值。这仅为光照计算经验公式增加了一个选项。该选项由符号常量 GL_LIGHT_MODEL_AMBIENT 来设定。例如，下列语句将一场景的总的背景光设定为低强度（暗）蓝色，使用的 alpha 值为 1.0。

```
globalAmbient [ ] = {0.0, 0.0, 0.3, 1.0};
glLightModelfv (GL_LIGHT_MODEL_AMBIENT, globalAmbient);
```

如果不设定全局环境光等级，则使用低强度白色(0.2, 0.2, 0.2, 1.0)（暗灰色）为默认值。

镜面反射计算需要确定几个向量，包括从表面到观察位置的向量 **V**，它指出表面位置与观察位置的关系。该常数单位向量 **V** 在正 z 方向(0.0, 0.0, 1.0)，即默认方向。但如果希望不用默认而使用位于观察坐标原点的实际观察位置来计算 **V**，则使用下列命令。

```
glLightModeli (GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

尽管该镜面反射计算使用实际的观察位置来计算 **V** 会花费较多时间，但可获得更具真实感的显示。当使用默认值 GL_FALSE（或 0、0.0）作为局部观察参数时不再计算表面的向量 **V**。

在OpenGL光照计算中加入表面纹理时，表面高光可叠加上去而镜面反射项可能使纹理图案变样。因此，作为一个选项，纹理图案可仅仅应用于对表面颜色有贡献的非镜面反射项上。这些非镜面反射项包括环境光、表面散射和漫反射。使用这一选项，OpenGL 光照子程序在对每一表面进行光照计算时生成两个颜色：镜面反射颜色和非镜面反射颜色。纹理图案先和非镜面反射颜色混合，然后再和镜面反射颜色混合。用下列函数可选择两个颜色项：

```
glLightModeli (GL_LIGHT_MODEL_COLOR_CONTROL,
               GL_SEPARATE_SPECULAR_COLOR);
```

如果不使用纹理图案则不必分开这两种颜色，而如果不用该选项则光照计算的效率较高。这一特性的默认值是 GL_SINGLE_COLOR，即不将镜面反射颜色和其他非镜面反射颜色分开。

在有些应用中，可能要显示对象的后向面。一个实体的内部剖面视图是这样的例子，其中某些后向面和前向面一起显示。但在默认时，仅对前向面使用赋予的材质进行光照计算。使用下列命令则可对前向面和后向面分别使用两种材质进行光照计算。

```
glLightModeli (GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

此时将后向面的表面法向量反向，并按赋给后向面的材质进行光照计算。要关闭两侧的光照计算，可在函数 glLightModel 中使用值 GL_FALSE（或 0、0.0）。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.20.7 OpenGL 表面特性函数

表面的反射系数和其他可选特性用下列函数设定：

```
glMaterial* (surfFace, surfProperty, propertyValue);
```

按照特性值的数据类型在函数后添加后缀 i 或 f，且当应用向量类特性时还要添加后缀 v。参数 surfFace 赋以符号常量 GL_FRONT、GL_BACK 或 GL_FRONT_AND_BACK 之一；参数 surfProperty 是一个用来标识表面参数如 I_{surf}、k_a、k_d、k_s 或 n_s 的符号常量；而参数 propertyValue 用相应的值来设定。除镜面反射指数 n_s 外所有其他特性均用向量值指定。我们需在描述对象几何数据前使用一系列 glMaterial 函数来指定该对象的所有光照特性。

表面散射颜色的 RGBA 值 I_{surf} 使用 OpenGL 符号常量 GL_EMISSION 来选择。例如，下列语句将前向面的散射颜色设定为浅灰色（白色）。

```
surfEmissionColor [ ] = {0.8, 0.8, 0.8, 1.0};
glMaterialfv (GL_FRONT, GL_EMISSION, surfEmissionColor);
```

表面的默认散射颜色是黑色，即(0.0, 0.0, 0.0, 1.0)。尽管对一个表面赋予了散射颜色，但该颜色不用来照明场景中的其他对象。如果要照明其他对象，必须用10.3节讨论的方法定义该表面为一个光源。

OpenGL 符号特性名 GL_AMBIENT、GL_DIFFUSE 和 GL_SPECULAR 用来设定表面反射系数的值。实际上，环境光和漫反射系数应赋以相同的向量值，而这可使用符号常量 GL_AMBIENT_AND_DIFFUSE 来实现。环境光系数的默认值是 (0.2, 0.2, 0.2, 1.0)，漫反射的默认值是(0.8, 0.8, 0.8, 1.0)，而镜面反射的默认系数是(1.0, 1.0, 1.0, 1.0)。常量 GL_SHININESS 用来设定镜面反射指数。可用 0 到 128 之间的任一值赋给这一特性，而默认值是 0。作为一个例子，下列语句设定三个反射系数和镜面指数。漫反射和环境光系数设定成在使用白色光照明时该表面显示为浅蓝色，镜面反射的颜色为入射光颜色，而镜面指数设定为 25.0。

```
diffuseCoeff [ ] = {0.2, 0.4, 0.9, 1.0};
specularCoeff [ ] = {1.0, 1.0, 1.0, 1.0};

glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, diffuseCoeff);
glMaterialfv (GL_FRONT_AND_BACK, GL_SPECULAR, specularCoeff);
glMaterialfv (GL_FRONT_AND_BACK, GL_SHININESS, 25.0);
```

反射系数的分量也可由颜色表中的值来设定，OpenGL符号常量GL_COLOR_INDEXES就是为此目的而设的。该颜色表索引项赋以三元素的整数或浮点数数组，而默认为(0,1,1)。

10.20.8 OpenGL 光照模型

OpenGL中使用10.19节中的基本光照模型计算表面光照效果，但在参数设定方面有所变化。环境光的等级是光源的环境光分量和设定的全局环境光的总和。漫反射计算使用光源的漫反射分量，而镜面反射计算使用每一光源的镜面反射分量。

同样，如果不使用局部观察选项，则指定从表面到观察位置的单位向量 **V** 设定为常数值(0.0, 0.0, 0.0)。对于一个“无穷远”处的光源，单位光方向向量与设定的从该光源发出的光线方向相反。

10.20.9 OpenGL 雾气效果

在使用OpenGL光照模型计算出表面颜色后，可设定场景中空气的颜色并将表面颜色和雾气颜色混合。然后可使用雾气强度衰减函数来模拟透过模糊或烟雾的雾气对场景的观察。各种雾气参数

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

用 9.14 节介绍的 glFog 函数来设定。

```
glEnable (GL_FOG);
glFog* (atmoParameter, paramValue);
```

添加后缀码 i 或 f 用来指出数据类型，对向量数据还要添加后缀码 v。

要设定一种雾气颜色，需要将OpenGL符号常量GL_FOG_COLOR赋给参数atmoParameter。例如，使用下列语句可设定雾气为蓝灰色：

```
GLfloat atmoColor [4] = {0.8, 0.8, 1.0, 1.0};
glFogfv (GL_FOG_COLOR, atmoColor);
```

雾气颜色的默认值是黑色，即(0.0, 0.0, 0.0, 0.0)。

接下来选择雾气衰减函数将对象颜色和雾气颜色混合。这由符号常量GL_FOG_MODE实现：

```
glFogi (GL_FOG_MODE, atmoAttenFunc);
```

如果参数atmoAttenFunc赋以值GL_EXP，则使用等式（10.31）作为雾气衰减函数。如赋以GL_EXP2，则使用等式（10.32）作为雾气衰减函数。对这两个指数函数中的任一个，使用下列函数选择空气强度：

```
glFog (GL_FOG_DENSITY, atmoDensity);
```

雾气衰减的第三个选项是线性的深度提示函数（9.13）。此时，参数atmoAttenFunc赋值为GL_LINEAR。参数 atmoAttenFunc 的默认值是 GL_EXP。

选定一个雾气衰减函数后，该函数被用来为对象计算调和的雾气-表面颜色。OpenGL雾气子程序使用等式（10.33）计算这一调和颜色。

10.20.10　OpenGL 透明性函数

在 OpenGL 中使用 4.3 节描述的颜色调和函数可模拟某些透明效果。但是，OpenGL 中的透明性一般不直接实现。我们可通过使用 alpha 调和值来指定透明程度，并按深度优先次序处理表面来为一个包含少量不透明和透明表面的简单场景混合对象颜色。但OpenGL颜色调和操作忽略折射效果，并且在有多种光照条件或动画的复杂场景中对透明表面的处理是比较困难的。OpenGL也不直接提供模拟一个半透明对象（如一张塑料薄膜或一块结霜的玻璃）的表面外貌的手段，这些对象将传递到半透明材料中的光散射出去。因此，要显示半透明表面或由折射形成的光照效果，必须另写子程序。要模拟通过半透明对象时的光照效果，可使用表面纹理和材料特性值的混合。对于折射效果，使用等式（10.29）计算所需位移量，将在透明对象后的表面上的像素位置做一些移动。

使用 glMaterial 和 glColor 等 OpenGL 的 RGBA 表面颜色命令中的 alpha 参数可指定场景中的对象为透明的。该对象的表面 alpha 参数可设定为透明系数（参见等式（10.30））的值。例如，如果用下列函数指定一个透明表面的颜色：

```
glColor4f (R, G, B, A);
```

则设定了 alpha 参数值为 $A = k_t$。完全透明表面的 alpha 值赋为 $A = 1.0$，而不透明表面的 alpha 值为 $A = 0.0$。

设定透明值后激活OpenGL的颜色调和功能并处理表面，次序为从离观察位置最远的对象开始到最近的对象结束。颜色调和功能激活后，使用赋予的 alpha 值将每一表面颜色和任何已存入帧缓存的重叠表面相混合。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

设定颜色调和因子使所有的当前表面（源对象）颜色分量用$(1-A)=(1-k_t)$来乘，而所有对应的帧缓存位置（目标对象）中的颜色分量用因子$A=k_t$来乘：

```
glEnable (GL_BLEND);
glBlendFunc (GL_ONE_MINUS_SRC_ALPHA, GL_SRC_ALPHA);
```

然后使用等式（10.30）及设定为k_t的alpha参数将这两个颜色调和，这里帧缓存颜色是那些在正被处理的透明对象后面的表面颜色。例如，如果$A=0.3$，则每一表面位置的新的帧缓存颜色是当前帧缓存颜色乘以30%的值和对象反射颜色乘以70%的值之和。（另一方法是，使用alpha颜色参数作为不透明因子来取代透明因子。如果设定A为一个不透明值，还需将函数glBlendFunc中的两个变量交换。）

可见性检测可使用9.14节的OpenGL深度缓存函数来实现。在处理每一个可见不透明表面时，存储了表面颜色和表面深度值。但在处理一个可见透明表面时，仅存储其颜色，因其并未遮挡背景的表面。因此，在处理每一个透明表面时，使用函数glDepthMask将深度缓存设定为只读状态。

如果按深度次序处理所有对象，则在处理每一个透明表面时关掉深度缓存的写模式并在处理完后再打开。也可将对象分为两类，如下面的程序所做的。

```
glEnable (GL_DEPTH_TEST);
\*  Process all opaque surfaces.  *\

glEnable (GL_BLEND);
glDepthMask (GL_FALSE);
glBlendFunc (GL_ONE_MINUS_SRC_ALPHA, GL_SRC_ALPHA);
\*  Process all transparent surfaces.  *\

glDepthMask (GL_TRUE);
glDisable (GL_BLEND);

glutSwapBuffers ( );
```

如果不按从后往前的次序处理这些透明对象，则该方法不能精确地累计所有情况下的表面颜色。但对于简单的场景，这是一种快速、高效地生成透明效果近似表示的方法。

10.20.11 OpenGL表面绘制函数

OpenGL子程序可用常数强度表面绘制或Gouraud表面绘制方法来显示表面。没有使用Phong表面绘制、光线跟踪法或辐射度方法的OpenGL子程序。绘制方法用下列函数来选定：

```
glShadeModel (surfRenderingMethod);
```

对参数surfRenderingMethod赋以符号常量GL_FLAT可指定常数强度表面绘制方法。使用符号常量GL_SMOOTH则指定Gouraud表面绘制方法（默认）。

在将glShadeModel函数应用于网格曲面如用多边形网格逼近的球面时，OpenGL子程序使用多边形顶点的表面法向量来计算多边形颜色。OpenGL中表面法向量的笛卡儿分量用下列命令指定：

```
glNormal3* (Nx, Ny, Nz);
```

该函数的后缀码是b（字节）、s（短整数）、i（整数）、f（浮点数）和d（双倍浮点数）。另外，在用数组表示向量分量时还要加上后缀码v。字节、短整数和整数转换成-1.0到1.0范围内的浮点数。glNormal函数将表面法向量分量设定为用于所有glVertex后继命令的状态值，且默认的法向量是正z方向(0.0, 0.0, 1.0)。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于平表面绘制而言，每一多边形仅需一个表面法向量。因此，可设定每一多边形法向量如下：

```
glNormal3fv (normalVector);
glBegin (GL_TRIANGLES);
    glVertex3fv (vertex1);
    glVertex3fv (vertex2);
    glVertex3fv (vertex3);
glEnd ( );
```

如果要对上述三角形应用 Gouraud 表面绘制方法，则需为每一顶点指定一个法向量：

```
glBegin (GL_TRIANGLES);
    glNormal3fv (normalVector1);
    glVertex3fv (vertex1);
    glNormal3fv (normalVector2);
    glVertex3fv (vertex2);
    glNormal3fv (normalVector3);
    glVertex3fv (vertex3);
glEnd ( );
```

尽管法向量并不需要指定为单位向量，但如果所有表面法向量都使用单位向量可减少计算。使用下列命令可自动将所有非单位向量转换成单位向量：

```
glEnable (GL_NORMALIZE);
```

该命令也会对那些经过缩放或错切等几何变换的表面向量进行规范化。

另一可用选项是指定一个法向量列表，与顶点数组（参见 3.17 节和 4.3 节）混合使用。创建法向量数组的语句是

```
glEnableClientState (GL_NORMAL_ARRAY);
glNormalPointer (dataType, offset, normalArray);
```

参数dataType赋以常量值GL_BYTE、GL_SHORT、GL_INT、GL_FLOAT（默认值）或GL_DOUBLE。参数 offset 给出数组 normalArray 中连续的法向量之间的字节数，其默认值为 0。

10.20.12　OpenGL 半色调操作

有些系统中使用OpenGL半色调操作可获得多种颜色和灰度效果。半色调逼近图案和操作依赖于硬件，并且对具有全彩色图形能力的系统不起作用。然而，如果一个系统每像素只有少量位时，可用半色调逼近 RGBA 颜色设定。激活半色调子程序的函数是

```
glEnable (GL_DITHER);
```

这是默认的函数，并且半色调子程序可由以下函数重新激活：

```
glDisable (GL_DITHER);
```

10.21　OpenGL 纹理函数

OpenGL 中有一个纹理函数的扩充集。我们可以为一条线段、一个表面、空间区域的一个内部体指定一个图案或指定一个插入另一纹理图案的子图案。我们可以按多种方式应用和管理纹理图案。另外，纹理图案可用来模拟环境映射。尽管有些参数可使用颜色表索引来设定，但 OpenGL 的纹理子程序只能在 RGB（RGBA）颜色模型下使用。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.21.1 OpenGL线纹理函数

用单下标颜色数组指定的一维RGBA纹理图案的参数由下列语句指定：

```
glTexImage1D (GL_TEXTURE_1D, 0, GL_RGBA, nTexColors, 0,
              dataFormat, dataType, lineTexArray);

glEnable (GL_TEXTURE_1D);
```

我们在函数glTexImage1D中将第一个参数设定为符号常量GL_TEXTURE_1D来指出正在为一个一维对象：一条线段定义一个纹理数组。如果不清楚系统是否支持该参数指定的纹理图案，则将函数glTexImage1D中的第一个参数设定为符号常量GL_PROXY_TEXTURE_1D。这使我们能在定义纹理数组的元素前先询问系统，而询问的过程在后面再讨论。

在该函数示例中的第二和第五个变量使用了值0。第一个0（第二参数）表示该数组不是某个大纹理数组的缩减。对于第五个变量，值0表示不希望在纹理周围有边界。如果对第五个变量赋值1，则该纹理图案在显示时有一个单像素宽的边界，用来将该图案与相邻图案融合。第三个变量，GL_RGBA，表示该纹理图案的每一颜色用RGBA四个值的指定。我们仅使用了RGB三个颜色值，但由于存储器边界对齐的缘故，有时使用RGBA四个值的效率更高。也可能使用各种其他颜色描述，包括单一的强度或亮度值。第四个变量nTexColors，赋以一个正整数，用来指出该线性纹理图案的颜色数量。由于第五个变量（边界参数）的值为0，该纹理图案中颜色数量必须是2的幂。如果这第五个变量指定为1，则该纹理图案的颜色数量是2加上2的幂。这两种边界颜色用来与相邻图案进行颜色调和。可提供不超过64+2种颜色的单足码纹理图案，有的OpenGL还允许更大的纹理图案。描述纹理颜色和边界颜色的参数存放在lineTexArray中。在本例中没有边界，而该数组中每一组连续的四个元素表示纹理图案的一个颜色成分。因此，在lineTexArray中的数值是4 × nTexColor。如果希望一个纹理图案的颜色数量为8，则该纹理数组必须包含4 × 8 = 32个元素。

参数dataFormat和dataType与在glDrawPixels及glReadPixels函数（参见3.19节）中的一样。我们赋予dataFormat一个OpenGL常量来指示纹理图案中怎样指定颜色值。例如，可以用符号常量GL_BGRA指出给出的颜色分量次序为蓝、绿、红、alpha。为了指定BGRA或RGBA数据类型，可将OpenGL常量GL_UNSIGNED_BYTE赋给dataType。可以赋给dataType的其他一些值取决于选定的数据格式，包括GL_INT和GL_FLOAT。

我们可以将一个纹理的多个副本或该纹理颜色的任何相邻子集映射到场景的一个对象上。当一组纹理元素映射到一个或多个像素区域时，纹理元素的边界通常与像素边界位置不对齐。一个像素区域可能包含在一个RGB（或RGBA）纹理元素中，也可能与多个纹理元素重叠。为了简化纹理映射时的计算，使用下列函数来给出每一像素的最接近纹理元素的颜色。

```
glTexParameteri (GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER,
                 GL_NEAREST);
glTexParameteri (GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER,
                 GL_NEAREST);
```

纹理子程序在必须放大纹理图案的一部分以适合指定的场景坐标范围时使用第一个函数，而在必须缩减纹理图案时使用第二个函数。（这两个纹理操作称为放大MAG和缩小MIN。）尽管将最近纹理颜色赋给一个像素可加快处理速度，但它可能导致走样效果。要按重叠纹理颜色的线性混合方式来计算像素颜色，需用符号常量GL_LINEAR代替GL_NEAREST。使用函数glTexParameter还可设定其他一些参数值，这些选项将在后面讨论。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

为一个场景指定 OpenGL 纹理图案在某种程度上和指定表面法向量、RGB 颜色及其他属性相类似。我们需将该纹理与某个对象结合，但与单一的颜色设置不同的是，现在有一组颜色值。对于一个一维的纹理空间，颜色值用跨越纹理空间的 0.0 到 1.0 之间的单个 *s* 坐标指定（参见 10.16 节）。这样，通过将纹理坐标值赋给对象位置来将纹理图案用到场景中的对象上。一维纹理空间的具体的 *s* 值用下列函数来选择。

```
glTexCoord1* (sCoord);
```

该函数允许使用的后缀码是 b（字节）、s（短整数）、i（整数）、f（浮点数）和 d（双倍浮点数），这依赖于纹理坐标参数 sCoord 的数据类型。如果 *s* 坐标值给定为一个数组，则使用后缀码 v。与颜色及其他类似的参数一样，*s* 坐标是一个状态参数，它应用于所有在其后定义的世界坐标位置。*s* 坐标的默认值是 0.0。

要将一个线性纹理图案映射到世界坐标系场景的位置上，可将 *s* 坐标赋给一条线段的端点。然后，纹理颜色可以按照多种方法映射到对象上，而 OpenGL 的默认方法是用纹理图案中的对应颜色值去乘每一像素颜色值。如果线性颜色是白色(1.0, 1.0, 1.0, 1.0)，即场景中对象的默认颜色，则该线段仅用该纹理颜色显示。

下例中，我们创建一个交替使用绿色和红色的四元素线性纹理图案。然后把从 0.0 到 1.0 的整个纹理图案赋给一条直线段。由于该线段是白色的，按默认要求，它将被显示成纹理颜色。

```
GLint k;
GLubyte texLine [16];   // 16-element texture array.

/*  Define two green elements for the texture pattern.
/*  Each texture color is specified in four array positions.
 */
for (k = 0; k <= 2; k += 2)
{
    texLine [4*k]   = 0;
    texLine [4*k+1] = 255;
    texLine [4*k+2] = 0;
    texLine [4*k+3] = 255;
}

/*  Define two red elements for the texture pattern.  */
for (k = 1; k <= 3; k += 2)
{
    texLine [4*k]   = 255;
    texLine [4*k+1] = 0;
    texLine [4*k+2] = 0;
    texLine [4*k+3] = 255;
}

glTexParameteri (GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri (GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

glTexImage1D (GL_TEXTURE_1D, 0, GL_RGBA, 4, 0, GL_RGBA, GL_UNSIGNED_BYTE,
texLine);

glEnable (GL_TEXTURE_1D);

/*  Assign the full range of texture colors to a line segment.  */
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glBegin (GL_LINES);
    glTexCoord1f (0.0);
    glVertex3fv (endPt1);
    glTexCoord1f (1.0);
    glVertex3fv (endPt2);
glEnd ( );

glDisable (GL_TEXTURE_1D);
```

该线段沿其路径交替地用绿色和红色的段显示。我们可对 s 坐标赋以任意的值。例如，用下列语句将纹理图案中间的红色和绿色映射到该线段上。

```
glBegin (GL_LINES);
    glTexCoord1f (0.25);
    glVertex3fv (wcPt1);
    glTexCoord1f (0.75);
    glVertex3fv (wcPt2);
glEnd ( );
```

这样，线段的前半段是红色而后半段是绿色。我们还可以使用 0.0 到 1.0 范围以外的 s 值。例如，如果对一个端点的 s 赋值 –2.0 而另一个端点的 s 赋值 2.0，则分四次将纹理图案映射到线段上。该线段显示成 16 个绿色段和 16 个红色段。对于超出单位区间的 s 坐标值，忽略其整数部分，除非指定了将 s 归入到 0 或 1.0 之间。图 10.114 给出了某些可能用包含两种绿色和两种红色的 RGB 值的数组显示的线图案。

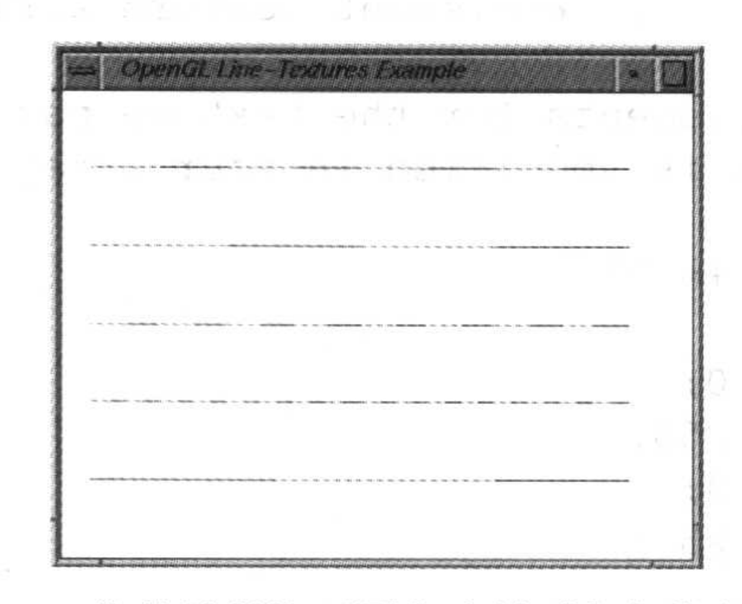


图 10.114　将单足码纹理图案映射到白色线段的例子

OpenGL纹理图案有关的参数和选项很多。但在深入了解OpenGL纹理子程序的这些特性之前，我们先讨论生成二维和三维纹理图案的基本函数。

10.21.2　OpenGL 表面纹理函数

我们可以使用与一维纹理图案例子中类似的函数来建立二维 RGBA 纹理空间的参数：

```
glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA, texWidth,
              texHeight, 0, dataFormat, dataType, surfTexArray);

glEnable (GL_TEXTURE_2D);
```

这里惟一的差别是必须指定三足码纹理数组的宽（列数）和高（行数）。宽和高在没有边界时都必须是 2 的幂，在有边界时都必须是 2 加上 2 的幂。我们再次使用 RGBA 颜色分量，并说明该图案没有边界且它不是一个大纹理图案的缩减。因此，存储在 surfTexArray 中的数组尺寸是 4 × texWidth × texHeight。对于二维纹理图案，按自下而上次序设定纹理数组元素的颜色值。从颜色图案的左下

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

角开始，将该数组第一行的元素设定为与纹理空间底行对应的RGBA值，将数组最后一行元素设定为与矩形纹理空间顶行对应的 RGBA 值（参见图 10.103）。

与使用线性纹理图案一样，场景中的表面像素可赋以最近纹理颜色或一个插值纹理颜色。我们可用在一维纹理中使用的同样两个glTexParameter函数来选择这些参数中的一个。一个函数指定在放大纹理图案以适合一个坐标范围时使用的选项，而另一个函数指定在缩减纹理时使用的选项。一个纹理图案可在一个方向拉长而在另一方向压缩。例如，下面的语句构成了使用最近颜色显示投影表面位置的纹理子程序。

```
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                    GL_NEAREST);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                    GL_NEAREST);
```

要将一个插值纹理颜色赋给表面像素，可使用符号常量 GL_LINEAR 来代替 GL_NEAREST。

二维纹理空间的一个坐标位置用下列函数来选择：

```
glTexCoord2* (sCoord, tCoord);
```

对纹理空间进行规范化，从而可用 0.0 到 1.0 范围内的坐标值来指定该图案。因而我们可以使用任一纹理坐标值来表示一个表面上的图案。纹理坐标可用多种形式来指定，并用一个后缀码 b、s、i、f 或 d 来指定数据类型。如果纹理坐标用数组给出则还要加上后缀码 v。

为展示二维纹理空间的 OpenGL 函数，下列程序段设定一个 32 × 32 的图案并将其映射到一个四边形表面。每一纹理颜色用四个 RGBA 分量指定，且该图案没有边界。

```
GLubyte texArray [32][32][4];

/*  Next: assign the texture color components to texArray.  */

/*  Select nearest-color option.  */
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA, 32, 32, 0, GL_RGBA,
              GL_UNSIGNED_BYTE, texArray);

glEnable (GL_TEXTURE_2D);

/*  Assign the full range of texture colors to a quadrilateral.  */
glBegin (GL_QUADS);
    glTexCoord2f (0.0, 0.0);     glVertex3fv (vertex1);
    glTexCoord2f (1.0, 0.0);     glVertex3fv (vertex2);
    glTexCoord2f (1.0, 1.0);     glVertex3fv (vertex3);
    glTexCoord2f (0.0, 1.0);     glVertex3fv (vertex4);
glEnd ( );

glDisable (GL_TEXTURE_2D);
```

10.21.3　OpenGL 体纹理函数

三维纹理空间函数是二维纹理空间函数的简单扩充。例如，一个没有边界的四足码 RGBA 纹理数组可用下列函数来建立：

```
glTexImage3D (GL_TEXTURE_3D, 0, GL_RGBA, texWidth, texHeight,
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
                    texDepth, 0, dataFormat, dataType, volTexArray);
glEnable (GL_TEXTURE_3D);
```

该RGBA纹理颜色存储在volTexArray中，其中包含了4 × texWidth × texHeight × texDepth个元素。数组的宽、高和深必须是2的幂或2加上2的幂。

下列语句可使用最近纹理颜色显示像素。

```
glTexParameteri (GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER,
                 GL_NEAREST);
glTexParameteri (GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER,
                 GL_NEAREST);
```

对于线性插值纹理颜色，可使用符号常量GL_LINEAR来代替GL_NEAREST。

三维纹理坐标用下列函数选择：

```
glTexCoord3* (sCoord, tCoord, rCoord);
```

然后，选定的每一个纹理空间位置与世界坐标场景中的空间坐标位置相结合。

10.21.4 OpenGL纹理图案的颜色选项

一个纹理空间的元素可以用许多不同的方法来指定。函数glTexImage1D、glTexImage2D和glTexImage3D的第三个变量用来指定图案每一元素的颜色分量的一般格式和数量。对这一描述可使用近40个符号常量。例如，每一纹理元素可以是一组RGBA值、一组RGB值、单个alpha值、单个红色强度值、单个亮度值或一对亮度值加上一个alpha值。另外，某些常量也指定位的尺寸。例如，OpenGL常量GL_R3_G3_B2指定在一个字节的RGB颜色中有3位用于红色、3位用于绿色、2位用于蓝色。

纹理函数中的参数dataFormat用来指定纹理元素的特殊格式。我们可以从该参数的11个符号常量中选择一个。从而可用指向颜色表的索引、单个alpha值、单个亮度值、亮度alpha值的组合、单个RGB分量的强度值、3个RGB分量，或者按BGRA次序的RGBA描述的4个分量来指定每一个纹理元素。参数dataType赋值为GL_BYTE、GL_INT、GL_FLOAT或一个同时指定数据类型和位尺寸的符号常量。可从20个符号常量中选择一个值用于该数据类型的参数。

10.21.5 OpenGL纹理映射选项

纹理元素映射到对象上时其值可以与当前对象颜色混合，或取代对象颜色。下列函数用来选择纹理映射方法：

```
glTexEnvi (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
           applicationMethod);
```

如果参数applicationMethod赋值为GL_REPLACE，则纹理颜色、亮度、光强度或alpha值取代对象的对应值。例如，alpha值的纹理图案取代对象的alpha值。类似的取代操作还用于用单个亮度或光强度值指定的纹理图案上。绿色强度值的图案取代对象颜色的绿色分量。

将applicationMethod赋值为GL_MODULATE导致对象颜色值的“调制”。即用纹理颜色调制当前颜色值。具体结果依赖于纹理图案中的元素格式，例如alpha值调制alpha值而强度值调制强度值。纹理图案的默认应用方法是GL_MODULATE。如果对象颜色是白色（对象的默认颜色）则调制操作生成与取代操作相同的结果，这依赖于如何指定纹理图案的元素。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们还可以将符号常量 GL_DECAL 用于纹理映射操作，然后该操作将 RGBA 的 alpha 值作为透明系数。此时，将该对象看作为对背景中的纹理是透明的。如果该纹理图案仅包含 RGB 值而没有 alpha 分量，则该纹理颜色取代对象颜色。同样，在有些情况下，如在纹理图案仅包含 alpha 值时，该贴花操作没有定义。

将常量 GL_BLEND 赋给 applicationMethod 时，纹理子程序使用下列函数指定的颜色进行颜色混合：

```
glTexEnv* (GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR,
           blendingColor);
```

按该混合颜色的数据类型决定加后缀 i 或 f。在混合颜色是一个数组时还要加后缀 v。

10.21.6　OpenGL 纹理环绕

当纹理空间的坐标超出 0 到 1.0 的范围时，可使用下列命令补充纹理空间中描述的图案：

```
glTexParameter* (texSpace, texWrapCoord, GL_REPEAT);
```

仅使用纹理空间坐标值的小数部分来补充图案。参数 texSpace 赋以符号常量 GL_TEXTURE_1D、GL_TEXTURE_2D 或 GL_TEXTURE_3D 之一，而 texWrapCoord 通过 GL_TEXTURE_WRAP_S、GL_TEXTURE_WRAP_T 或 GL_TEXTURE_WRAP_R 来指定纹理空间坐标。

为了将纹理坐标强制限于单位区间内，我们使用 GL_CLAMP 代替 GL_REPEAT。如果一个要强制的坐标值大于 1.0，则赋值为 1.0。同样，要强制的坐标值小于 0.0 时将其赋为 0.0。在一个特定的纹理空间中，我们可以任意指定复用和强制的组合。所有坐标的默认是 GL_REPEAT。

10.21.7　复制帧缓存中的 OpenGL 纹理图案

无论是原始的图案还是子图案都可以从帧缓存中的值获得。下面的函数使用一块 RGBA 像素值来为当前纹理状态建立一个二维图案。

```
glCopyTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA, x0, y0, texWidth,
                  texHeight, 0);
```

变量表中的两个0值指出这个图案不是缩减的且没有边界。以缓存左下角为坐标原点的帧缓存位置(x0, y0)给出 texWidth × texHeight 的像素颜色块。

用做纹理子图案的像素块可用下列类似的函数获得：

```
glCopyTexSubImage2D (GL_TEXTURE_2D, 0, xTexElement,
                     yTexElement, x0, y0, texSubWidth, texSubHeight);
```

这一块像素值放在纹理位置(xTexElement, yTexElement)的当前图案中。参数 texSubWidth 和 texSubHeight 给出像素块的尺寸，其左下角位于帧缓存位置(x0, y0)处。

10.21.8　OpenGL 纹理坐标数组

有了颜色数据、表面法向量和多边形的边标志，我们在可与顶点数组混合或辅助该数组的列表中指定纹理坐标（参见 3.17 节和 4.3 节）。

```
glEnableClientState (GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer (nCoords, dataType, offset, texCoordArray);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

参数 nCoords 赋值 1、2、3 或 4，指定纹理图案的尺寸。默认值 4 表示以齐次坐标形式指定纹理空间，因此纹理空间位置用前三个坐标值除以第四个而得。这种形式很有用，例如在纹理图案是一个透视照片时。参数dataType赋以常量GL_SHORT、GL_INT、GL_FLOAT（默认值）或GL_DOUBLE。数组 texCoordArray 的坐标位置间的字节位移在参数 offset 中指定，该参数的默认值是 0。

10.21.9 OpenGL 纹理图案命名

常常会在一个应用中使用几个纹理图案，因此OpenGL允许创建多个命名的纹理图案。然后我们可以在任何时间简单地通过名字指定要使用的纹理。这是一种每次引用glTexImage函数的非常有效的方法，因为每次对glTexImage的调用需要创建图案，而该图案可能来源自一个数据文件中的颜色值。我们需在图案定义之前选择一个正（无符号）整数来命名一个纹理图案。例如，下列语句用纹理 3 来命名前一例子中的绿色和红色线图案并随后激活它。

```
glBindTexture (GL_TEXTURE_1D, 3);
glTexImage1D (GL_TEXTURE_1D, 0, GL_RGBA, 4, 0, GL_RGBA,
              GL_UNSIGNED_BYTE, texLine);

glBindTexture (GL_TEXTURE_1D, 3);
```

第一个 glBindTexture 语句为图案命名，第二次调用 glBindTexture 将该图案指定为**当前纹理状态**（current texture state）。如果创建了多个纹理图案，则可通过另一个纹理名再调用 glBindTexture 来激活该纹理在场景中某些对象上的应用。对于一个二维或三维图案，将glBindTexture函数的第一个参数改为GL_TEXTURE_2D或GL_TEXTURE_3D。在一个纹理名第一次引用时，使用图案参数的默认值创建一个图案。

使用下面的命令可删除一个或多个现有的纹理图案：

```
glDeleteTextures (nTextures, texNamesArray);
```

参数 nTextures 指定在数组 texNamesArray 中列出的图案名的数量。

我们也可以让 OpenGL 为一个图案选择一个名字从而不必记住已使用的名字。例如，

```
static GLuint texName;

glGenTextures (1, texName);
glBindTexture (GL_TEXTURE_2D, texName);
```

作为一个例子，下列语句用来获得六个未使用的纹理名并使用其中之一来创建一个图案。

```
static GLuint texNamesArray [6];

glGenTextures (1, texNamesArray [3]);
glBindTexture (GL_TEXTURE_2D, texNamesArray [3]);
```

OpenGL 中有一个查询语句用来确认一个纹理名是否已用于现有图案：

```
glIsTexture (texName);
```

如果 texName 是一个已有图案的名字则该函数返回值 GL_TRUE，否则返回值 GL_FALSE。如果 texName = 0 或发生错误也返回 GL_FALSE。

10.21.10 OpenGL 纹理子图案

一旦定义了一个纹理图案，我们就可以创建称为子图案的另一个图案，从而修改原始图案中的任意部分或全部。子图案中的纹理值取代原始图案中的指定值。这常常比用新的值重新创建一个纹

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

理的效率高。例如，下列函数指定一组RGBA颜色值来取代一个二维纹理的一个部分，该部分没有边界也不是大图案的缩减。

```
glTexSubImage2D (GL_TEXTURE_2D, 0, xTexElement,
                 yTexElement, GL_RGBA, texSubWidth, texSubHeight,
                 0, dataFormat, dataType, subSurfTexArray);
```

参数xTexElement和yTexElement用来选择原始图案中纹理元素的整数坐标位置，其中(0, 0)是指图案左下角的纹理元素。该子图案通过其左下角位置与位置(xTexElement, yTexElement)对应来向原始图案传递。参数 TexSubWidth 和 TexSubHeight 给出子图案的尺寸。一个 RGBA 纹理图案的数组 subSurfTexArray 的颜色元素数量是 4 × texSubWidth × texSubHeight。其他参数与 glTexImage 函数中的一样，且可为一维或三维纹理建立类似的子图案。

10.21.11　OpenGL 纹理缩减图案

对于缩小的对象尺寸，我们可使用 OpenGL 函数建立一系列纹理缩减图案，称为 mip 图（参见 10.17 节）。建立一系列缩减图案的一种方法是反复引用glTexImage函数，引用时函数中第二个变量（级数）用较大整数。原始图案的缩减级数看成为0。原始图案缩小一半的图案的级数为 1，再缩小一半则级数为 2，按此处理其他缩减级数。当我们设定级数为 1 或以上时，copyTexImage 函数还生成一个缩减图案。

另外，我们还可以让 OpenGL 自动生成缩减图案。例如，16 × 16 表面纹理的 RGBA 缩减图案可通过下列 GLU 函数获得：

```
gluBuild2DMipmaps (GL_TEXTURE_2D, GL_RGBA, 16, 16, GL_RGBA,
                   GL_UNSIGNED_BYTE, surfTexArray);
```

该函数生成完整的一组四个图案，其缩减后的尺寸分别是 8 × 8、4 × 4、2 × 2 和 1 × 1。我们还可以用下列函数建立缩减：

```
gluBuild2DMipmapLevels (GL_TEXTURE_2D, GL_RGBA, 16, 16,
                GL_RGBA, GL_UNSIGNED_BYTE, 0, minLevel, maxLevel,
                surfTexArray);
```

该函数按由参数 minLevel 和 maxLevel 指定的级数范围生成缩减图案。在每一种情况下，都为在 0 级指定的当前纹理图案构建 mip 图 。

我们使用glTexParameter函数和GL_TEXTURE_MIN_FILTER符号常量来选择从缩减图案确定像素颜色的方法。例如，下列函数为二维纹理指定映射过程：

```
glTexParameter (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_NEAREST_MIPMAP_NEAREST);
```

该函数指出纹理子程序应该使用尽可能接近匹配像素尺寸（MIPMAP_NEAREST）的缩减图案。然后将该缩减图案中靠近的纹理元素（GL_NEAREST）的颜色赋给像素。使用符号常量 GL_LINEAR_MIPMAP_NEAREST，我们指定从最接近的缩减图案纹理中颜色的线性混合。使用 GL_NEAREST_MIPMAP_LINEAR（默认值），我们指定从每一个最接近像素尺寸的缩减图案中最接近的纹理元素计算的平均颜色。而GL_LINEAR_MIPMAP_LINEAR使用一组最接近尺寸缩减图案的纹理颜色的线性混合计算一个像素颜色。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.21.12 OpenGL 纹理边界

多个纹理或一个纹理的多个复制应用于一个对象且用纹理颜色的线性插值计算像素颜色时，可能在相邻图案边界出现走样。通过在每一纹理图案中包含边界且边界颜色与相邻图案的纹理边界颜色相匹配来避免走样现象。

我们可以使用多个方法来指定纹理边界。使用glTexSubImage函数可以将一个相邻图案的颜色值复制到另一个图案的边界，边界颜色也可直接在由glTexImage函数指定的纹理数组中赋值。另一种选择是使用glTexParameter子程序设备边界颜色。例如，使用下列函数为一个二维图案指定边界颜色：

```
glTexParameterfv (GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR,
                  borderColor);
```

这里参数borderColor赋以一个四元素的RGBA颜色分量。默认的边界颜色是黑色(0.0, 0.0, 0.0, 0.0)。

10.21.13 OpenGL 代理纹理

在任一的glTexImage函数中，可将第一个变量设定为符号常量，称为一个纹理代理。这一常量的目的是保持该纹理图案的定义直到发现是否有足够的资源来处理这一图案。对于一个二维图案，该代理常量是GL_PROXY_TEXTURE_2D，而类似的常量可用于线性或体纹理。一旦建立纹理代理，可使用glGetTexLevelFunction来确定是否可提供专门的参数值。

作为使用纹理代理的例子，下列语句询问系统以确定是否可使用为二维图案指定的高度：

```
GLint texHeight;

glTexImage2D (GL_PROXY_TEXTURE_2D, 0, GL_RGBA12, 16, 16, 0,
              GL_RGBA, GL_UNSIGNED_BYTE, NULL);
glGetTexLevelParameteriv (GL_PROXY_TEXTURE_2D, 0, GL_RGBA12,
                          GL_TEXTURE_HEIGHT, &texHeight);
```

如果系统不能提供所要的图案高度（此例中为16），则在参数texHeight中返回值0。否则，返回值等于所要的值。类似地，使用符号常量GL_TEXTURE_WIDTH、GL_TEXTURE_DEPTH、GL_TEXTURE_BORDER和GL_TEXTURE_BLUE_SIZE可以询问其他的图案参数。每一种情况下，返回值为0表示不能提供在glTexImage函数中所询问的参数值。对于浮点数据值，将后缀码i换成f。

尽管我们可以获得肯定的答案，但仍有可能无法将该图案存入存储器。这将在另一个图案正占据有效存储器时发生。

10.21.14 二次曲面的自动纹理映射

OpenGL中提供在某些应用中自动生成纹理坐标的子程序。该功能在较难直接确定一个对象表面坐标时特别有用；有一个GLU函数用于二次曲面的这些子程序。

要将一个纹理映射到一个二次曲面，必须先建立该纹理空间的参数。然后引用下列函数并按照8.6节叙述的方法定义二次对象。

```
gluQuadricTexture (quadSurfObj, GL_TRUE)
```

该函数中参数quadSurfObj是二次对象的名字。如果要关闭二次曲面的纹理，需将符号常量GL_TRUE改成GL_FALSE。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.21.15　齐次纹理坐标

一个四维纹理空间位置用下列函数描述：

```
glTexCoord4* (sCoord, tCoord, rCoord, htexCoord);
```

纹理坐标与场景坐标的变换一样用 4 × 4 矩阵变换：每一坐标被齐次参数（参见 5.2 节）来除。因此，上面函数中纹理坐标 s、t 和 r 被齐次参数 htex 来除，以生成一个实际的纹理空间位置。

纹理空间的齐次坐标在多投影效果混合在一个显示中时很有用。例如，一个对象的透视投影可能包含一个由不同透视投影变换生成的纹理图案。然后该纹理图案可用齐次纹理坐标修改以调整该纹理的透视。使用齐次纹理坐标管理纹理映射还可生成其他效果。

10.21.16　其他的 OpenGL 纹理选项

OpenGL中还有一些用来完成许多其他的纹理管理和应用功能的函数。如果我们得到的纹理图案（从一张照片或其他来源）不是 2 的幂次，OpenGL 提供有一个修改图案尺寸的函数。在有些OpenGL的实现中，有多个纹理子程序用于将多个纹理图案传递给一个对象。环境映射可在OpenGL中通过建立一个球面形状的纹理映射来模拟，而球面环境图案的纹理坐标以及其他纹理应用则可自动生成。

10.22　小结

通常，场景中的对象可由发散光源和其他对象的反射表面进行照明。光源可以模拟成点光源，也可以模拟成扩大的尺寸。另外，光源可以是方向光源，还可以当做无穷远距离光源或局部光源来处理。光能衰减一般按距离的二次函数的倒数应用于传递的光，而投射光也按角度衰减。场景中的反射面有不透明的、完全透明的或部分透明的。对反射和折射两种情况都可以用漫反射和镜面反射两个概念描述。

表面位置的光强度用光照模型计算，而多数图形软件包中的基本光照模型是物理定律的简单近似。这些光照计算给出表面位置反射光及穿过对象的透射光的 RGB 分量的光强度值。基本光照模型一般可处理多个点光源，这些点光源可以是距离光源、局部光源或投射光源。场景中的环境光用对每一 RGB 分量和对所有表面均固定的光强度来描述。表面漫反射强度与表面法线的角度余弦成正比。镜面反射强度用Phong模型计算。可以使用材料的透明系数来近似透明效果，而使用Snell定律计算折射角度，可以准确地得到透过透明材质的光线的几何路径。单个光源的阴影效果可通过识别从光源无法看到的场景区域来添加。获得透明材料光反射和透射效果所必需的计算一般也不在基本光照模型中，但我们可以使用分散漫反射分量的方法来模拟。

在实际显示时，由光照模型计算出的光强度值，必须将其映射到所用系统中允许的强度等级。强度的对数标尺可用于按相同亮度感觉对光强度进行分级。另外，对强度值进行 gamma 校正可以消除由于显示设备的非线性所带来的错误。在2级监视器上，我们利用半色调模式和抖动技术来模拟光强度值的范围。还可以使用半色调近似技术来增加每像素有两个以上强度的系统的光强度等级。当场景中描绘出的点的个数与显示设备上的像素个数相等时，可以利用按序抖动、误差分散和点分散等方法来模拟强度范围。

图形软件包中的表面绘制通过将基本光照模型的计算应用于扫描线过程来实现，该过程从几个表面点的强度值推广到表面上的所有投影像素位置。使用恒定强度表面绘制，也称为平面绘制，我们可使用计算出的一个颜色显示一个表面的所有点。平面绘制在观察点和光源位置远离场景对象时

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对多边形或曲面多边形网是精确的。Gouraud明暗处理计算多边形顶点处的光强度值，然后在多边形表面上将强度进行插值，从而得到网格曲面的近似反射效果。Phong明暗处理结果较准确，但速度较慢，它在多边形表面上对多边形顶点的平均法向量进行插值，然后，根据面上各点处的法向量使用基本光照模型来计算该点处的光强度值。快速Phong明暗处理采用泰勒扩展式近似方法来加速计算。

光线跟踪技术提供了一种获得全局镜面反射和透射效果的准确方法。该技术跟踪从像素点射入场景中的光线，通过在对象之间的反射和折射累计光强度的作用。我们可以为每个像素构造一棵光线跟踪树，从终止节点回溯至根节点，可以累计出该像素所对应的光强度值。在光线跟踪算法中，可以结合空间分割技术，即仅在整个空间的某个子区间内测试光线与对象是否相交。这样，就大大降低了求交的计算量。分布式光线跟踪在每个像素位置跟踪多条光线，并在一些光线参数，如方向和时间上随机分布光线。这为模拟对象表面的光滑度、透明度、有限照相机光圈、分布式光源、阴影效果和动感模糊等提供了一种准确的手段。

辐射度方法通过计算场景中不同表面间的辐射能量的转移，从而得到准确的漫反射模拟效果。逐步求精方法加速了辐射度计算，每次仅考察一个表面上的能量转移。结合使用光线跟踪和辐射度方法，可以生成高度真实感的图形。

环境映射是一种全局光照效果的快速近似方法，它使用一个环境阵列来存储场景中的背景光强度信息。然后，根据特定的观察方向将该阵列映射至场景中的对象表面。

光子映射提供了对复杂场景全局光照精确和高效的模拟。从光源生成随机光线，每一光线的光照效果存储在一个光子图中，该图将光照信息与场景几何分开。光照效果的精度随着产生的光线的增加而改善。

使用多边形表面、纹理映射、凹凸映射或帧映射可以为对象添加表面细节。我们可以将小多边形表面叠加到大表面上以得到多种不同的图案，也可将纹理图案在一维、二维和三维空间中定义，并用于线条、表面或空间体上。过程纹理映射使用函数计算对象光照效果的变化。凹凸映射通过运用凹凸函数扰动对象表面法向量来模拟不规则的对象表面。帧映射则是凹凸映射的一种延伸，它允许表面的水平和垂直变化来模拟各向异性的材料。

OpenGL的核心库包括多个函数集用于设定点光源、指定基本光照模型中的各种参数、选择表面绘制方法、激活半色调逼近子程序和将纹理图案应用于对象。表10.2和表10.3总结了这些OpenGL光照、表面绘制和纹理映射函数。

表 10.2 OpenGL 照明和表面绘制函数小结

函数	描述
glLight	指定一个光源特征值
glEnable (lightName)	激活一个光源
glLightModel	指定全局光照参数值
glMaterial	为一光学表面指定一个值
glFog	为雾气参数指定一个值；用 glEnable 函数激活雾气效果
glColor4f (R, G, B, A)	为一个模拟透明性的表面指定一个 alpha 值。在函数 glBlendFunc 中，设源混合因子为GL_SRC_ALPHA而目标混合因子为GL_ONE_MINUS_SRC_ALPHA
glShadeModel	指定 Gouraud 表面绘制或单色表面绘制
glNormal3	指定一个表面法向量
glEnable (GL_NORMALIZE)	指出表面法向量要转换成单位向量
glEnableClientState(GL_NORMAL_ARRAY)	激活表面法向量数组的处理子程序
glNormalPointer	创建将用于向量数组的表面法向量表
glEnable (GL_DITHER)	激活以半色调逼近图案应用于表面绘制的操作

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

表 10.3　OpenGL 纹理映射函数小结

函数	描述
glTexImage1D	为建立一维纹理空间指定参数（用 glEnable 激活纹理）
glTexImage2D	为建立二维纹理空间指定参数
glTexImage3D	为建立三维纹理空间指定参数
glTexParameter	为纹理映射子程序指定参数
glTexCoord	为一维、二维、三维或四维纹理空间的纹理坐标指定参数
glTexEnv	指定纹理环境参数如纹理映射的混合颜色
glCopyTexImage	复制一块帧缓存像素颜色用做纹理图案
glCopyTexSubImage	复制一块帧缓存像素颜色用做纹理子图案
glTexCoordPointer	在顶点表的辅助表中指定纹理坐标
glBindTexture	将一个名字赋给一个纹理图案，并激活一个命名的图案
glDeleteTextures	消除一组指定名字的纹理
glGenTextures	自动生成纹理名
glIsTexture	确定一个指定名字的纹理是否存在的查询命令
glTexSubImage	创建一个纹理子图案
gluBuild*Mipmaps	为一维、二维或三维纹理空间自动生成纹理缩减图案
gluBuild*MipmapLevels	为一维、二维或三维纹理空间自动生成指定级数的纹理缩减图案
glGetTexLevelParameter	询问系统以确定是否提供了一个纹理参数
gluQuadricTexture	为一个二次表面激活或关闭纹理

参考文献

基本光照模型和面绘制技术在 Gouraud(1971)和 Phong(1975)、Freeman(1980)、Bishop and Wiemer(1986)、Birn(2000)、Akenine-Möller and Haines(2002)以及 Olano, Hart, Heidrich, and McCool (2002)中讨论。光照模型和绘制方法的实现算法在Glassner(1990)、Arvo(1991)、Kirk(1992)、Heckbert (1994)、Paeth(1995)和Sakaguchi, Kent, and Cox(2001)中给出。半色调方法在Velho and Gomes(1991)中给出。按序抖动、误差分散及点分散方面的进一步信息请参见 Knuth(1987)。

光线跟踪过程在 Whitted(1980)、Amanatides(1984)、Cook, Porter, and Carpenter(1984)、Kay and Kajiya(1986)、Arvo and Kirk (1987)、Quek and Hearn (1988)、Glassner (1989)、Shirley (1990 and 2000)和 Koh and Hearn (1992)中处理。辐射度算法可在 Goral, Torrance, Greenberg, and Battaile (1984)、Cohen and Greenberg (1985)、Cohen, Chen, Wallace, and Greenberg (1988)、Wallace, Elmquist, and Haines (1989)、Chen, Rushmeier, Miller, and Turner (1991)、Dorsey, Sillion, and Greenberg(1991)、Sillion, Arvo,Westin, and Greenberg (1991)、 He, Heynen, Phillips, Torrance, Salesin, and Greenberg (1992)、Cohen and Wallace (1993)、 Lischinski, Tampieri, and Greenberg(1993)、Schoeneman, Dorsey, Smits, Arvo, and Greenberg (1993)和 Sillicon and Puech(1994)中找到。光子映射算法在Jensen (2001)中详细讨论。纹理映射方法和应用在 Williams (1983)、Segal, Korobkin, van Widenfelt, Foran, and Haeberli (1992)和 Demers (2002)中讨论。能量传递方程的一般讨论、绘制过程和我们对光及颜色的感知在 Glassner (1995)中给出。

使用OpenGL光照和绘制函数的更多的程序设计例子在 Woo, Neider, Davis, and Shreiner (1999)中给出。OpenGL 光照、绘制和纹理函数的编程示例还可在 Nate Robins 的辅导网站: http://www.cs.utah.edu/ ~ narobins/opengl.html 中找到。完整的 OpenGL 光照和绘制函数列表在 Shreiner (2000)中提供。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

练习题

10.1 编写一个子程序来实现漫反射公式（10.12），使用一个点光源和恒定表面明暗处理来绘制指定多面体的各表面。通过一组多边形表，包括各个多边形表面的法向量来描述对象，附加的输入参数包括环境光强度、光源强度和对象表面的反射系数，所有坐标信息可以在观察参照系中直接给出。

10.2 修改练习题 10.1 中的子程序，绘制球面的多边形网格面片。

10.3 修改练习题 10.2 中的子程序，使用 Gouraud 明暗处理绘制该球面。

10.4 修改练习题 10.3 中的子程序，使用 Phong 明暗处理绘制该球面。

10.5 编写一个子程序，实现漫反射和镜面反射的等式（10.17），使用一个点光源和 Gouraud 明暗处理来绘制一个球面的多边形网格面片。可以使用一组多边形表，包括每个多边形面的法向量来描述对象。附加输入包括泛光强度、光源强度、表面反射系数和镜面反射参数，所有的坐标信息可以在观察参照系中直接给出。

10.6 修改前一练习题的子程序，使用 Phong 明暗处理绘制多边形面片。

10.7 修改前一练习题的子程序，加入一个线性光强度衰减函数。

10.8 修改前一练习题的子程序，在场景中包含两个点光源。

10.9 修改前一练习题的子程序，通过一片玻璃观察球表面。

10.10 讨论在镜面反射模型中，比较$(\mathbf{N}\cdot\mathbf{H})^{n_s}$与$(\mathbf{V}\cdot\mathbf{R})^{n_s}$可能出现的结果差异。

10.11 请验证：当所有向量均共面时，等式（10.22）中的$2\alpha=\phi$，但通常情况下$2\alpha\neq\phi$。

10.12 讨论如何将不同的可见面判别算法与光强度模型相结合，显示一组不透明的多面体。

10.13 讨论如何将各种可见面判别算法修改后以处理透明对象，请问有没有不能处理透明对象的可见面判别算法？

10.14 请基于一种可见面判别算法，设计一个可以在远处点光源照明的场景中判别阴影区域的算法。

10.15 在$n\times n$的像素网格上，每个像素可以用m个不同光强显示，请问利用半色调近似技术可以显示多少个光照等级。

10.16 在一个3×3像素网格的2级RGB系统中，半色调近似方法可以生成多少个不同的颜色组合？

10.17 在3×3像素网格和每像素两个强度等级（0，1）的情况下，编写一个子程序，使用半色调近似技术来显示一组给定表面的光强度变化。

10.18 编写一个子程序，使用等式（10.48）中的迭代关系生成按序抖动矩阵。

10.19 编写一个子程序，使用按序抖动方法来显示一个给定的光强数组。

10.20 编写一个子程序，对一给定的$m\times n$光强矩阵实现误差分散算法。

10.21 编写一个程序来为一场景实现基本光线跟踪算法，该场景使用一视点位置上的单点光源照明。在场景中，一个球在一正方形棋盘之上盘旋。

10.22 编写一个程序为一场景实现光线跟踪算法，场景中使用一给定点光源组进行照明，包含多个球面和多边形表面。

10.23 编写程序，利用空间分割技术来实现基本光线跟踪算法，场景中使用一给定点光源组进行照明，包含多个球面和多边形表面。

10.24 编写程序实现下列特征的分布式光线跟踪：(1)使用每像素 16 个抖动光束进行采样；(2)分布式反射方向（光泽）；(3)分布式的折射方向（半透明）；(4)附加光源。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

10.25 编写一算法，使用分布式光线跟踪来模拟运动对象的动感模糊。

10.26 当光源为立方体的一内侧面时，实现辐射度算法以绘制立方体的内侧面。

10.27 设计一算法以实现逐步求精的辐射度算法。

10.28 编写一子程序，以完成一球面的环境映射。

10.29 编写一程序以实现纹理映射：(a)球面，(b)凸多面体。

10.30 修改前一练习题中的程序使图案映射到网格中的一个面片。

10.31 修改前一练习题中的程序使图案映射到球面的一个指定部分。

10.32 编写一个程序，将一个给定的一维纹理图案当做一个对角线带映射到立方体的指定面片上。

10.33 修改前一练习题中的程序，给定球面上的两点，将一维纹理映射到该球面上。

10.34 给定一球面，编写一凹凸映射过程以生成橘子的凹凸表面。

10.35 编写一凹凸映射算法，根据某凹凸函数生成对象表面的扰动法向量。

10.36 编写一个OpenGL程序，显示包含一个球面和一个多面体的场景，使用两个光源进行光照：一个是局部红色光源而另一个是远距离白色光源。用Gouraud表面绘制建立漫反射和镜面反射的表面参数，并应用二次强度衰减函数。

10.37 修改前一练习题中的程序，用两个投射光源取代该红色局部光源：一个红色和一个蓝色。

10.38 修改前一练习题中的程序，将雾气加入场景。

10.39 修改前一练习题中的程序，通过一块半透明玻璃观察该场景。

10.40 编写一个完整的 OpenGL 程序，使用图 10.114 所示的各种一维纹理图案显示对角线。

10.41 编写一个程序，使用二维 OpenGL 纹理图案在蓝色背景上显示一个黑白棋盘。

10.42 修改前一练习题中的程序，背景用白色，棋盘为红色和蓝色的方格。

10.43 编写一个程序，使用二维OpenGL纹理图案显示一个有均匀的红色斜带的白色矩形。将背景设成蓝色。

10.44 修改前一练习题中的程序，将该纹理图案映射到一个球面。

10.45 修改前一练习题中的程序，将该纹理图案映射到 GLUT 茶壶表面。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第11章　图形用户界面和交互输入方法

虽然我们已经可以用前面章节中讨论的方法与程序指令来构造程序和提供输入数据，但是具备交互指定图形数据输入的能力对于一个图形系统来说通常是很有用处的。例如，在一个程序执行的过程中，我们可能想要通过点击屏幕上的某一位置来改变观察点或者物体在场景中的位置，也可能想通过选择菜单项来改变动画参数。许多图形化设计系统通过交互指定控制点的坐标值来构造样条曲线，并且采用交互式绘制方法来制图。图形程序使用多种类型的数据。人们已经设计出各种各样的交互式输入方法来处理这些数据。此外，系统界面也采用大量的交互式图形技术，包括窗口、图标、菜单、鼠标以及其他光标控制设备。

11.1　图形数据的输入

图形程序使用多种输入数据，例如坐标位置的数值、属性值、字符串参数的数值、几何变换的参数值、观察条件和照明参数等。包括ISO和ANSI标准在内的许多图形软件包都提供了丰富的输入功能来处理这些数据。但是，输入的过程需要与窗口管理器和特殊的硬件设备进行交互。因此，有些图形系统，特别是那些主要提供设备无关的输入功能的系统，通常包含较少的用于处理输入数据的交互过程。

对图形软件包中输入功能的标准组织方式将输入功能按照其处理的数据类型进行分类。这种方案允许任何一种物理输入设备，例如键盘或鼠标，都可以用来输入各种类型的图形数据，尽管有些设备在用于输入某种数据时比其他设备更适合。

11.2　输入设备的逻辑分类

如果按照输入的数据类型将输入功能分类，则用来输入某种特定数据的设备就被称为这种数据类型的**逻辑输入设备**（logical input device）。标准的逻辑输入数据设备分为以下六类：

LOCATOR	指定坐标位置(x, y)的设备（定位设备）
STROKE	指定一组坐标位置的设备（笔划设备）
STRING	输入文字的设备（字符串设备）
VALUATOR	指定标量值的设备（定值设备）
CHOICE	选择菜单项的设备（选择设备）
PICK	选择图形的组成部分的设备（拾取设备）

11.2.1　定位设备

交互地选择一个坐标位置的标准方法是使用屏幕光标进行定位。在某些特定应用中也可以使用其他方法进行定位，例如选择某个菜单项。我们可以使用鼠标、游戏杆、轨迹球、空间球、拇指轮、拨号盘、数字化仪的触笔或手动光标以及其他光标定位设备来实现定位。当屏幕光标到达要求的位置时，按下某一键将激活对该屏幕点坐标的存储操作。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

键盘可以按几种方式用做定位设备。一个通用键盘一般有四个控制键，可以将光标向上、向下、向左和向右移动。增加另外四个键，就可以将光标沿对角线方向移动。持续按下选择的光标键，可以实现光标的快速移动。作为其他选择，游戏杆、游戏盘、轨迹球或拇指轮可以装在键盘上，从而控制相应的光标移动。对于某些应用来说，使用键盘键入坐标值或其他代码来指定坐标值也是很方便的。

其他设备，例如光笔，也用来交互输入坐标位置。由于光笔通过检测屏幕荧光体发射的光来进行操作，必须在实现时进行某些特殊的处理。

11.2.2　笔划设备

这一类逻辑输入设备用于输入一组顺序的坐标点。笔划设备的输入相当于多次调用定位设备。输入的一组点常用于显示折线。许多用于产生定位输入的物理设备均可以作为笔划设备。鼠标、轨迹球、游戏杆或数据板手动光标的连续移动，经过转换便成为一组坐标位置值。图形数据板是一种普通的笔划设备。可以通过按键激活使数据板进入“连续”模式。当光标在数据板表面上移动时，就可以产生一组坐标值。这样的过程常用于允许艺术家在屏幕上绘画的画笔系统，以及跟踪布局并将其数字化（用于以后存储）的工程系统。

11.2.3　字符串设备

最基本的用于字符串输入的物理设备是键盘。输入的字符串通常作为图形的标记。

其他一些物理设备也可用于在写字模式下生成字符图案。这时，通过笔划或定位设备在屏幕上逐步绘制字符。然后，一个模式识别程序将使用预定义图案的字典来解释这些字符。

11.2.4　定值设备

这一类逻辑设备在图形系统中用于输入标量值。定值设备用于设定各种图形参数，例如几何变换参数、观察参数和照明参数等，还用于为特定应用设定物理参数（温度、电压等级、强度系数等）。

用来提供定值输入的典型物理设备是一组控制旋钮。通过旋转旋钮可以输入预先指定的任意范围内的浮点数。向一个方向旋转旋钮将增加输入值，而向相反方向旋转则减小输入值。旋转式电位器将旋钮的转动转换成对应的电压。该电压再转换成预定义的标量范围内（例如 -10.5 到 25.5）的一个数。除了旋转式电位器，滑动电位器有时也用来将线性运动转换成标量值。

任何一个带有一组数字键的键盘都可以作为定值设备。用户可以直接键入浮点格式的数值，尽管这比使用旋转式或滑动电位器的操作速度慢，但比较简单。

游戏杆、轨迹球、数据板和其他交互设备，可以在对照一个标量范围解释压力或运动后将其改装成定值输入设备。向一个方向的移动，例如从左向右，就增加输入的标量值；而相反方向的移动则减小标量值。

提供标量输入的另一种技术，是在屏幕上显示滑动块、按键、旋转式标尺和菜单。图 11.1 给出了几种可能的标尺表示。从鼠标、游戏杆、空间球或其他设备获得的定位输入，可以用来确定显示器上的一个坐标位置。然后，将该屏幕位置转换成一个数值作为输入。作为一种反馈机制，可以将某种符号显示在所选择的位置，也可以在屏幕的适当位置回显数值以便确认该选择。

11.2.5　选择设备

图形软件使用菜单来选择程序设计选项、参数值和用于构图的对象形状。常用于选中一个菜单项的选择设备是光标定位设备，如鼠标、跟踪球、键盘的光标键、触摸板或按钮盒。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

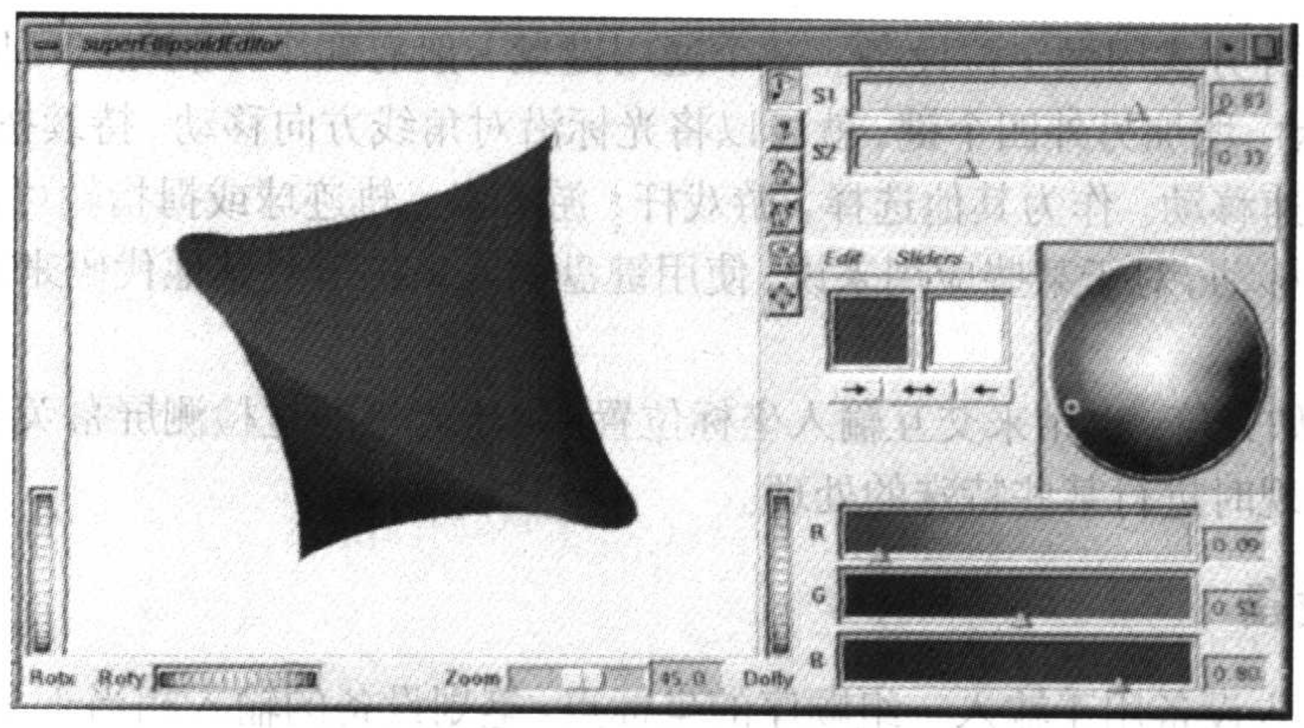


图 11.1　在屏幕上显示的用于交互式选择参数值的标尺。其中，滑动块和拇指轮用来选择大椭圆的参数 $s1$ 和 $s2$、RGB 颜色分量、旋转角及缩放参数；将一个小圆安排在颜色轮中，用来选择组合的 RGB 颜色；键盘上的方向键也可以对选中的标量值进行较小的修改

作为独立部件设计的键盘功能键或“按钮盒”常用于输入菜单选项。通常，每一个按键是可编程的，因此其功能可以根据不同应用而改变。输入设备有时会预定义某些按键的功能。

我们可以使用光标控制设备对屏幕上列出的菜单选项进行选择。选定一个坐标位置(x, y)后，将其与每一个列出的菜单项范围进行比较。选中一个水平和垂直边界坐标值为x_{min}、x_{max}、y_{min}和y_{max}的菜单项的条件是输入的坐标同时满足下面两个不等式：

$$x_{min} \leqslant x \leqslant x_{max}, \qquad y_{min} \leqslant y \leqslant y_{max} \tag{11.1}$$

对每次只显示少量选项的较大菜单，通常使用触摸屏。选定的屏幕位置要和每一菜单选项所占用的区域进行比较来决定将要进行的操作。

选择输入的替代方法有键盘和语音输入。可以使用标准键盘来键入命令或菜单选项。对于这类选择输入的方法，某些缩写的格式非常有用。可以将菜单列表编号或给其一个短标识名。在语音输入系统中，也可以使用类似的编码方法。语音输入在选项数量较少时（少于 20）比较有用。

11.2.6　拾取设备

拾取设备用于选择场景中即将进行变换或编辑的部分。有多种方法可用于拾取场景中的对象，任何一种以拾取为目标的输入机械都可以归属为拾取设备。通过屏幕光标定位来完成拾取操作是最常用的方法。用户可以使用鼠标、游戏杆或键盘将光标定位，然后按下选择按键记录下光标的屏幕像素坐标。然后就用这个屏幕坐标来拾取一个完整的对象、一个网格曲面上的一个面片、多边形的一条边或一个顶点。其他拾取方法包括突出显示当前候选对象、通过名称选择对象或者采用几种方法的组合。

如果采用光标定位方法，拾取程序需要使用该场景的逆观察和几何变换将选中的屏幕坐标位置映射到场景的世界坐标位置，然后才能用世界坐标位置与场景中各个对象的坐标范围相比较。如果某一对象的坐标范围包含该拾取位置，则找到这一拾取对象。该对象的名称、坐标和其他信息都可以用来进行需要的变换或编辑操作。但是，如果两个或两个以上对象的坐标范围同时包围该选中位置，则必须进行进一步的检查。根据待选对象的类型和场景复杂度，可能需要经过几个层次的搜索以确定选中的对象。例如，我们试图拾取一个坐标范围与其他三维物体重叠的球体，光标拾取的位置就要与这两个物体的每个面片的坐标范围进行比较。如果仍然无法确定拾取对象，就要进一步与每条线段的坐标范围进行比较。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如果根据坐标范围的比较无法惟一地确定拾取对象，则通过计算拾取位置与各线段之间的最近距离来判定拾取对象。图 11.2 中的拾取位置就落在了两条线段的坐标范围中。对于一条以点(x_1, y_1)和点(x_2, y_2)为端点的线段，从拾取点(x, y)到该线段距离的平方由下式计算：

$$d^2 = \frac{[\Delta x(y - y_1) - \Delta y(x - x_1)]^2}{\Delta x^2 + \Delta y^2} \tag{11.2}$$

其中，$\Delta x = x_2 - x_1, \Delta y = y_2 - y_1$。如果要加速距离的计算，可以使用各种近似方法或其他标识方法，例如简化为计算拾取位置到线段端点的距离。

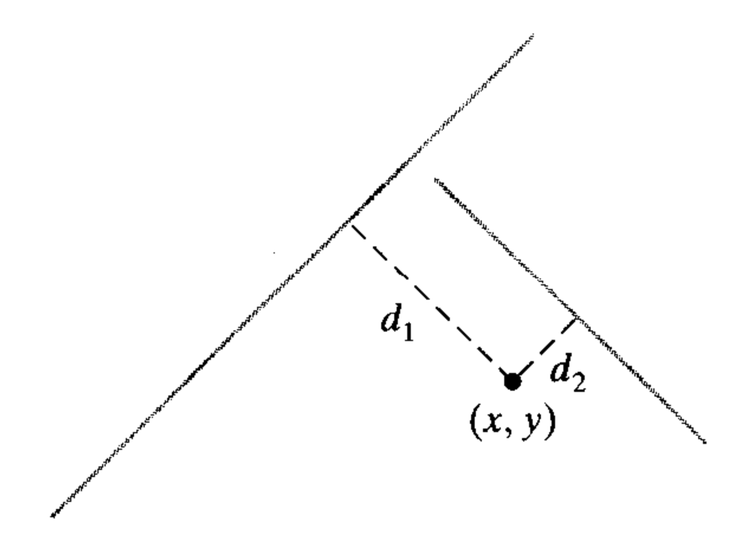


图 11.2　拾取位置到线段的距离

如果避免将拾取位置与曲面面片和线段进行坐标范围比较，拾取操作还可以被简化。当拾取位置位于两个或两个以上对象的坐标范围内时，拾取程序可以返回一组候选的拾取对象。

另一种拾取技术是将一个**拾取窗口**（pick window）与选中的光标位置相关联。如图 11.3 所示，该窗口以光标坐标点为中心，对每一候选线段进行裁剪来确定与拾取窗口相交的线段。拾取线段时，通过让拾取窗口适当变小，就可以找到惟一穿过该窗口的线段。某些图形软件包通过以拾取窗口作为裁剪窗口进行观察和投影变换，然后重建三维场景来实现三维拾取功能。这种重建不产生任何显示输出，只是利用裁剪程序来确定落在拾取观察体内的对象。落在拾取观察体内的所有对象的信息都被返回进行处理。这些信息包括对象的名称和深度范围，其中深度范围可以用于选择在拾取观察体内离用户最近的对象。

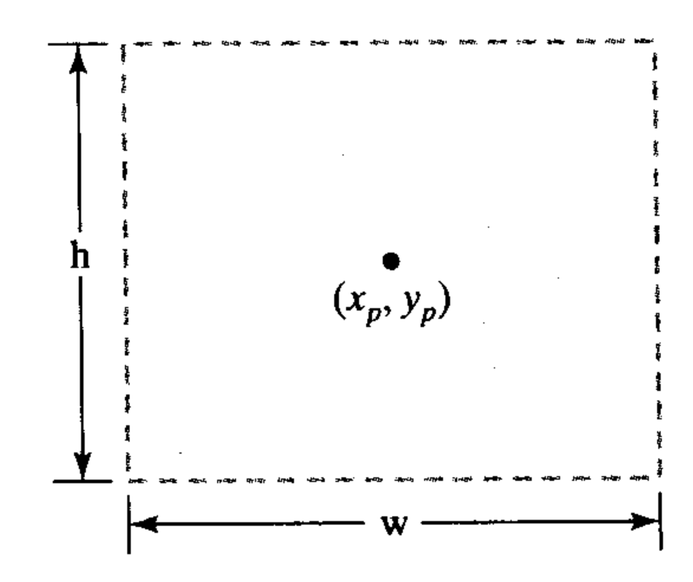


图 11.3　以拾取坐标(x_p, y_p)为中心的拾取窗口，宽为 w，高为 h

醒目显示候选对象的方法可以避免计算拾取距离或窗口裁剪交点，并让用户解决拾取的多义性。一种办法是逐个突出显示覆盖拾取位置（或拾取窗口）的对象。当醒目显示一个对象时，用户可以使用键盘发出“拒绝”或“接受”的指令。当用户接受某个对象为拾取对象时，这次拾取操作就结束了。因此，拾取操作也可以不进行光标定位，而仅仅依靠逐个醒目显示场景中的所有对象来完成。第一个按键用于启动醒目显示对象序列并依次向后遍历。第二个按键用来在醒目显示所需对象时停止这一过程。如果这种处理需要搜索太多的对象，则可以增加一个按键来加速这一过程并帮

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

助确认对象。第一个按键用来开始快速地依次突出显示对象。第二个按键用来停止这一过程。第三个按键则用来在所需的对象于按下第二个按钮之前就已经突出显示过的情况下慢慢地往回退。最后，总是通过停止键来结束整个拾取过程。

如果对象可以通过名称来选择，我们可以使用键盘输入来进行拾取。这是一种直接的但交互性欠佳的拾取选择方法。某些图形软件包允许对图形对象在多个层次上命名，直到单独的图形元素。描述性的名称可以在拾取过程中帮助用户。但该方法有一些缺点，它通常比在屏幕上交互地拾取处理得要慢，而且用户可能需要提示来回忆各种对象的名称。

11.3 图形数据的输入功能

采用输入设备逻辑分类的图形软件包提供多种函数来选择输入设备和数据类型。这些函数允许用户指定以下选项：

- 图形程序和设备如何进行交互（输入模式）。程序或设备之一可以启动数据输入还是两者可以同时操作。
- 使用哪一种物理设备为特定逻辑分类提供输入（例如，使用一个数据板作为笔划设备）。
- 何时输入数据，使用哪一种设备在输入时将特定数据类型传递到指定的数据变量中。

11.3.1 输入模式

交互式图形系统中的某些输入函数指明程序如何与输入设备进行交互。程序可以在处理过程中随时启动输入（称为请求模式），或者由输入设备独立地提供更新了的输入数据（称为取样模式），或者输入设备也可以独立地存储所有采集到的输入数据（称为事件模式）。

在**请求模式**（request mode）中，由应用程序启动数据输入。处理从提出请求时被挂起，直到接收了所要的数据。该输入模式与通用程序设计语言中的一般输入操作相对应。程序和输入设备交替工作，设备处于等待状态直到收到输入请求，然后程序处于等待状态直到收到输入数据。

在**取样模式**（sample mode）下，应用程序和输入设备各自独立地操作。输入设备可能在程序处理其他数据的同时工作。输入设备每次得到的新数据将覆盖以前的输入数据成为当前值。当程序请求一个新数据时，就从输入设备取得当前值。

在**事件模式**（event mode）下，输入设备启动数据输入并交给应用程序。程序和输入设备也是同时工作的，但是输入设备将数据放进一个输入队列中，称为事件队列。所有输入数据均被存储起来。当程序需要一个新数据时，就从输入队列中取得。

在取样模式和事件模式下，任意数目的设备可以同时工作。某些设备可以处于取样模式下，而另一些则处于事件模式下。但是在请求模式下，一个时间片内只能有一个设备可以提供输入。

输入函数库中的其他函数指明用于各种逻辑数据类型的物理输入设备。对于某些输入类型，交互式软件包中的输入过程包含复杂的处理。例如，为了得到一个世界坐标位置，输入程序必须在一个输入的屏幕位置上进行若干计算（包括观察变换等）来得到在场景中的世界坐标位置。并且，这种处理还涉及来自窗口例程的信息。

11.3.2 回显反馈

在交互输入程序中，用户常常要求回显输入的数据及相关的参数。在这种情况下，输入数据被显示在一个指定的屏幕区域内。例如，回显反馈通常包括拾取窗口的大小、最小拾取距离、光标的大小和形状、拾取时醒目显示的方式、定值输入的上下界以及分辨率（尺度）。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

11.3.3　回调函数

设备无关的图形软件包可以用一个辅助库来提供有限的一组输入功能，从而以回调函数的方式来处理输入程序。这些回调函数与系统软件进行交互，指定当某个输入事件发生时程序应该采取何种行为。典型的输入事件包括移动鼠标、按下鼠标键、或按下键盘上的按键。

11.4　交互式构图技术

图形软件包常常提供多种交互方法来辅助构图，包括定位对象、应用约束、调整对象的尺寸、设计形状和图案。其中提供了各种输入选项，从而对定位设备和笔画设备的坐标输入信息进行调整和解释（根据选项）。例如，可以限制所有的线段或者是水平的，或者是垂直的。输入的坐标可以作为将要绘制的对象的位置或边界，或者用来重新安排前面已经显示的对象。

11.4.1　基本的定位方法

我们可以使用一个指点设备来交互式地选择坐标位置，一般是通过定位屏幕光标。如何使用选中的位置取决于设定的处理选项。这个坐标位置可能是一条新线段的一个端点位置，也可以用于定位对象，例如定位一个球体的中心或者作为字符串定位的中心或起始点。作为定位对象的一种辅助方法，所选择位置的数值可以被回显在屏幕上。在回显坐标值的指引下，我们可以用拨号盘或方向键等设备微调选择位置直到获得精确的定位。

11.4.2　拖曳

使用屏幕光标拖曳对象来移动该对象，是交互式构图的常用技术之一。我们先用鼠标将光标定位到一个对象，然后按下鼠标键将光标移动到一个新的位置后松开鼠标键，选择的对象就会移动到新的光标位置上。通常，在光标移动的过程中，被拖曳的对象将跟随光标的当前位置显示。

11.4.3　约束

有些应用需要预先说明一些对象的方向和对齐方式。约束是改变输入坐标值以产生对象坐标的指定方向和对齐方式的规则。例如，我们可以约束一条输入直线不是水平方向的就是垂直方向的，如图 11.4 和图 11.5 所示。可以通过比较输入直线的两个端点的坐标值来实现此约束。如果两个 y 坐标的差值小于两个 x 坐标的差值，则显示一条水平线；否则，显示一条垂直线。这种水平－垂直约束对于形成网络布局很有用，它可以在生成水平线和垂直线时不必对终点进行精确定位。

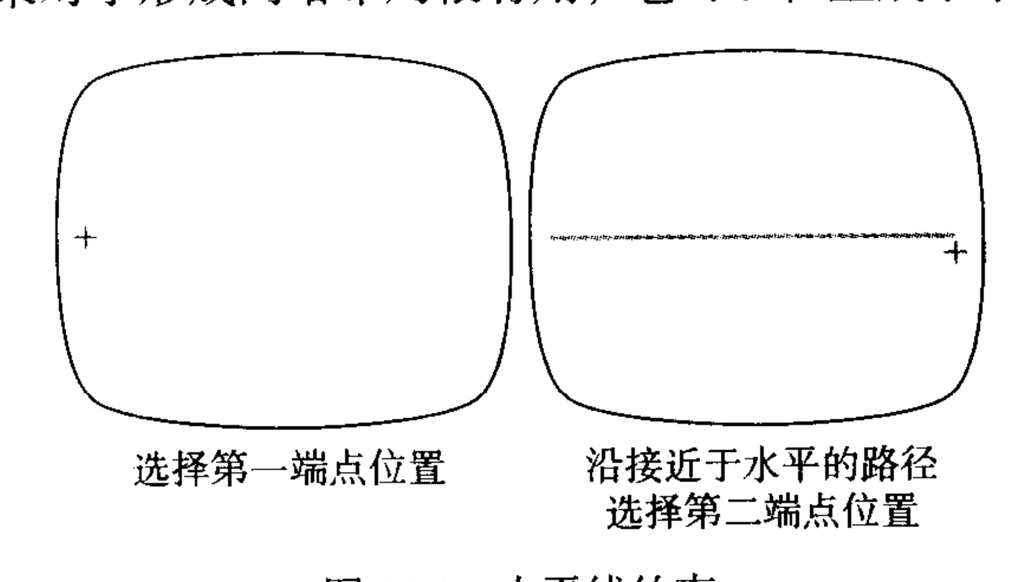


图 11.4　水平线约束

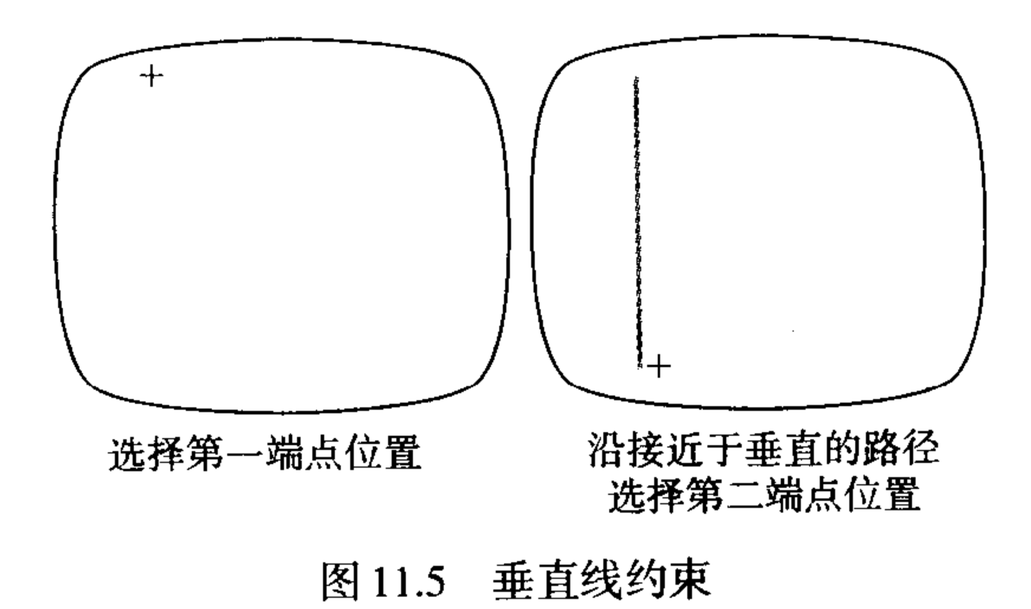


图 11.5　垂直线约束

还可以使用其他的约束对输入坐标进行各种对齐处理。可以将直线约束成具有固定角度，例如 45°；输入坐标也可以约束到预定义的路径，例如在一个圆弧上。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

11.4.4 网格

在屏幕上某一部分显示为正交直线的网格是另一类约束。在使用网格时，任何输入坐标位置将移到最近的两根网格线的交点上。图11.6示例了使用网格来绘制一条直线段。其中，两个光标位置均被移动到最近的网格交点上，然后在这两个交点之间画一条直线段。使用网格可以方便地构造对象，因为一条新的线段可以很容易地与前一条线段相连接，只需在靠近一个线段端点的网格交点附近进行定位。网格线之间的间距通常可以是用户设定的选项。同样，网格可以在显示和不显示之间进行转换，有时还可以使用部分网格，或是在不同屏幕区域有不同大小的网格。

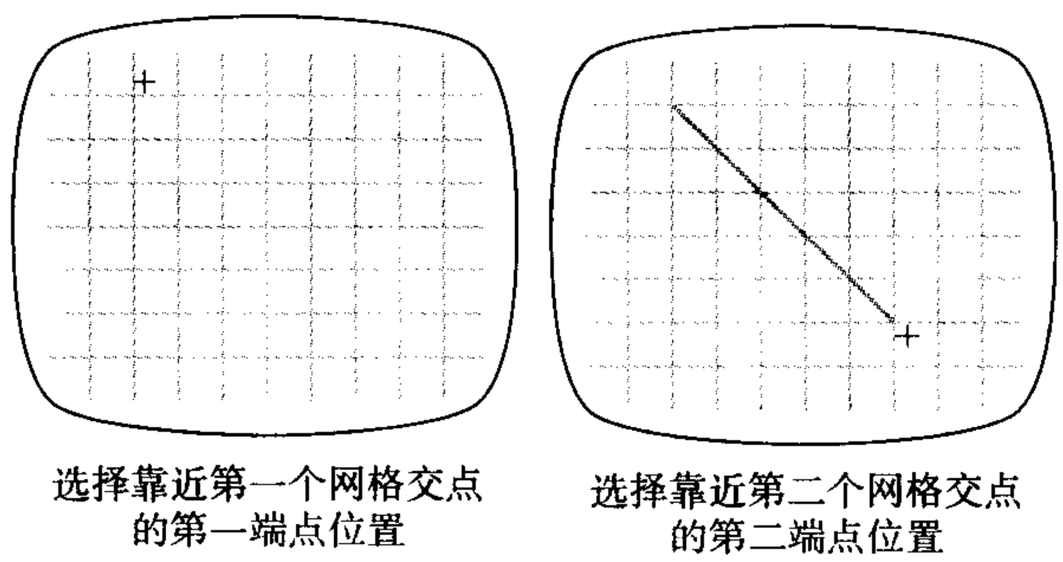


图 11.6 使用网格交点约束的端点画线

11.4.5 橡皮条方法

通过在起始点到移动的屏幕光标之间拉出一条直线的橡皮条方法，可以构造和定位直线段。橡皮条方法可以使一个对象的尺寸被交互地拉伸或收缩。图11.7示例了使用橡皮条方法指定一条直线段。我们先选择一个固定的屏幕位置作为直线段的第一个端点，然后当屏幕光标移动时，始终显示从第一个端点到当前光标位置的一条线段。再次选择的屏幕光标位置就成为该线段的另一个端点。当使用鼠标进行操作时，鼠标键按下后就开始绘制橡皮条线，直到松开鼠标键才完成一条直线段的输入。

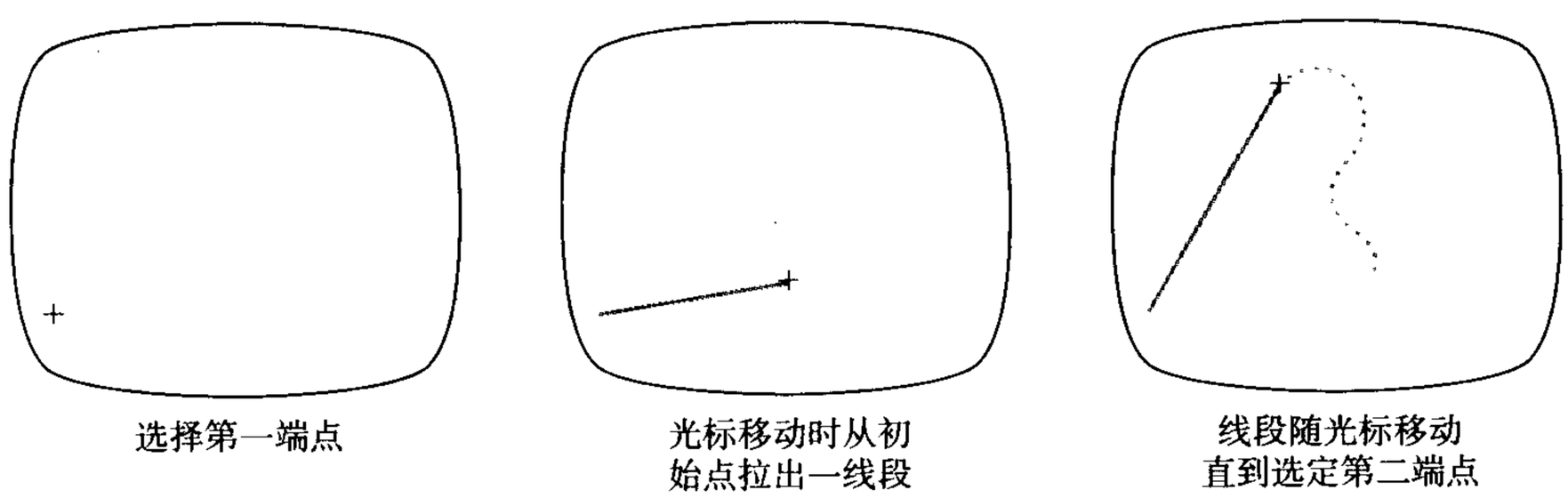


图 11.7 绘制和定位一直线段的橡皮条方法

除了直线段以外，橡皮条方法还可以构造和定位其他对象，如矩形和圆等。图 11.8 示例了使用橡皮条方法构造矩形，而图11.9示例了一个橡皮条圆的构造。有多种方式实现橡皮条的构造，例如，一个矩形的形状和尺寸可以有以下几种调整方式：单独移动顶边、底边或任意一条侧边。

11.4.6 引力场

在构造一个图形的时候，有时需要在某线段的端点之间连接另外的线段，由于在连接点精确地对屏幕光标定位是很难的，因此，图形软件包可以设计成将任意一个靠近线段的点位置转换成

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

线段上的位置。这种转换通过在线段附近建立引力场（gravity field）而实现。在引力场范围内的任意位置均被移（“吸引”）至线段上最靠近该位置的点上。图11.10使用阴影给出了一条线段的引力场区域。

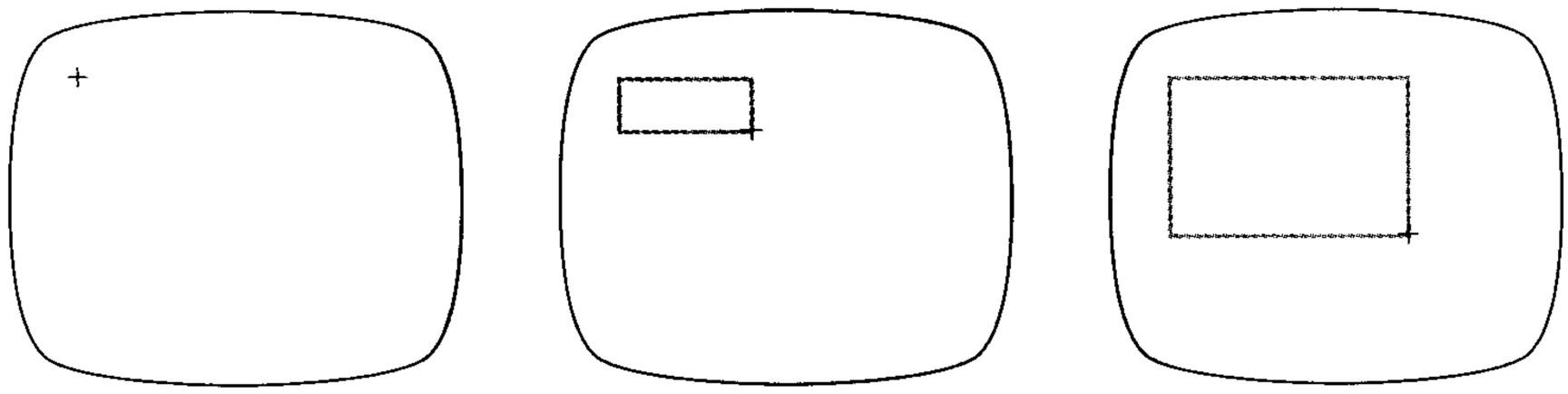


图 11.8　构造矩形的橡皮条方法

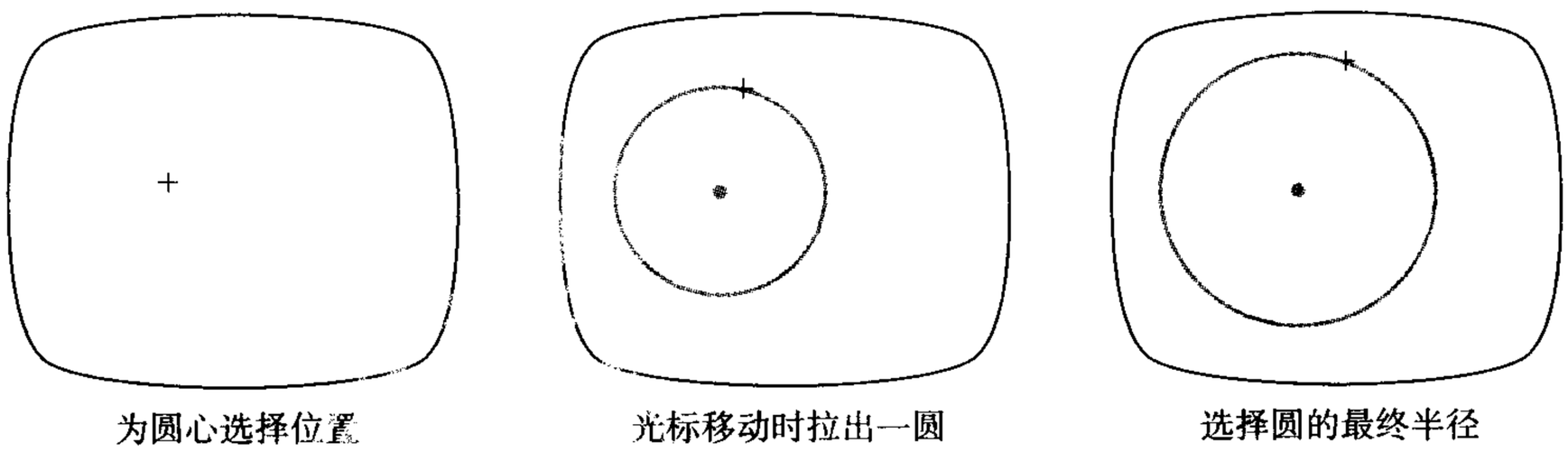


图 11.9　使用橡皮条方法构造一个圆

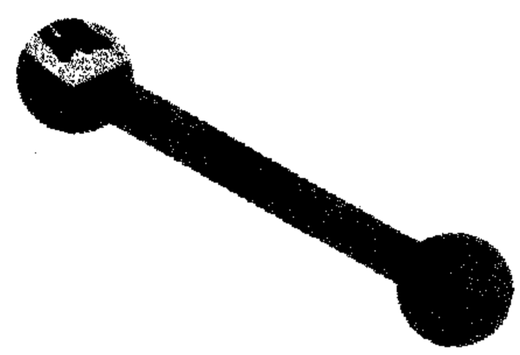


图 11.10　围绕一直线段的引力场，在阴影区内任意的选择点均被移至线段上的某一位置

围绕端点的区域被放大了，以便于用户比较容易地在端点上连接线段。在引力场的圆形区域内选择的位置都被吸引到该区域的端点上。引力场既要足够大以便能帮助定位，又要足够小从而不和其他线段发生重叠。如果已经显示了许多线段，那么引力场将会相互重叠，这时可能不容易正确地指定一点。通常不显示引力场的边界。

11.4.7　交互式绘画方法

素描、绘图、着色的选择有多种形式。直线段、多边形和圆可以使用上一节所讨论的方法来生成。曲线可以通过标准曲线形状（例如圆弧和样条）或手绘过程来绘制。样条曲线通过指定一组给出曲线大概形状的离散屏幕点来交互地构图，然后系统使用多项式曲线根据这些点来拟合曲线。在手工绘制中，通过数据板上触笔的路径或视频监视器屏幕光标的路径来生成曲线。一旦显示出一条曲线，设计者可以调整沿曲线路径上选择的点的位置，从而改变曲线形状。

线宽、线型和其他属性选项一般也包括在绘画系统中。这些选择可使用4.5节讨论的方法来实现。各种画笔形状、画笔图案、颜色组合、对象形状和表面纹理图案都可在许多系统中、特别是为艺术家工作站而设计的系统中找到。某些绘图系统根据艺术家的手加在触笔上的压力来改变线宽和

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

笔划。图 11.11 给出了一个绘画系统的窗口和菜单系统，用于让艺术家选择指定对象的形状变化、不同的表面纹理以及场景的各种光照条件。

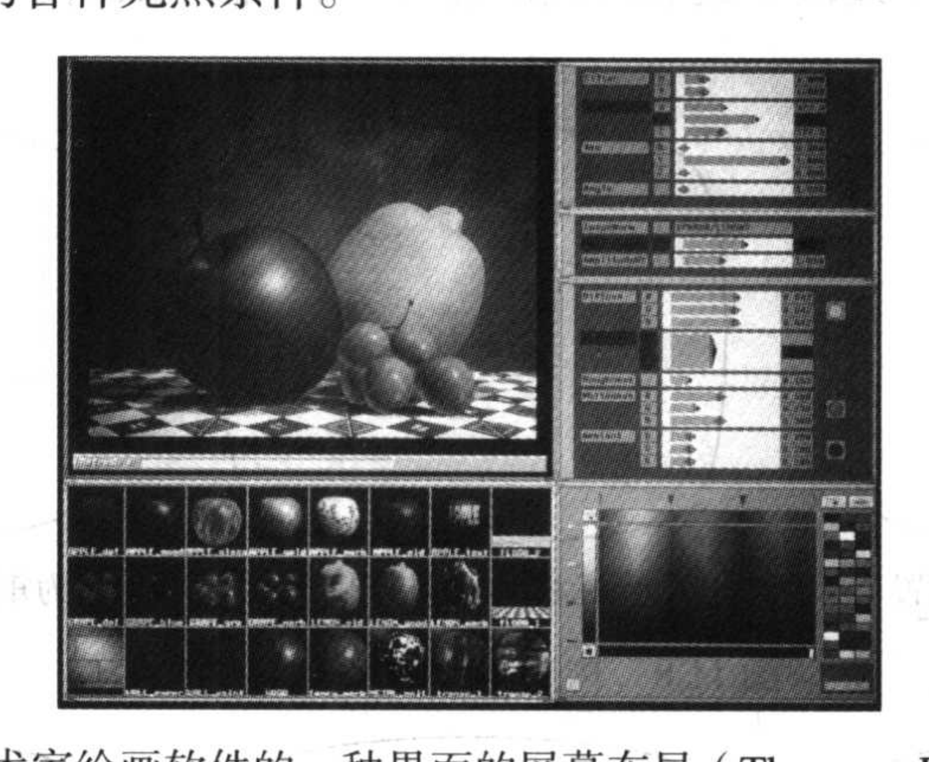


图 11.11 展示艺术家绘画软件的一种界面的屏幕布局（Thomson Digital Image 提供）

11.5 虚拟现实环境

图 11.12 给出了一种典型的虚拟现实环境。在这种环境下，使用数据手套（参见 2.4 节）来实现交互输入，这种数据手套可以抓取和移动虚拟场景中的对象。计算机生成的场景通过头套观察系统（参见 2.1 节）以立体投影图形式进行显示。跟踪设备计算头套和数据手套相对于场景中对象的位置和角度。使用这个系统，用户可以利用数据手套穿越场景，并重新安排对象位置。

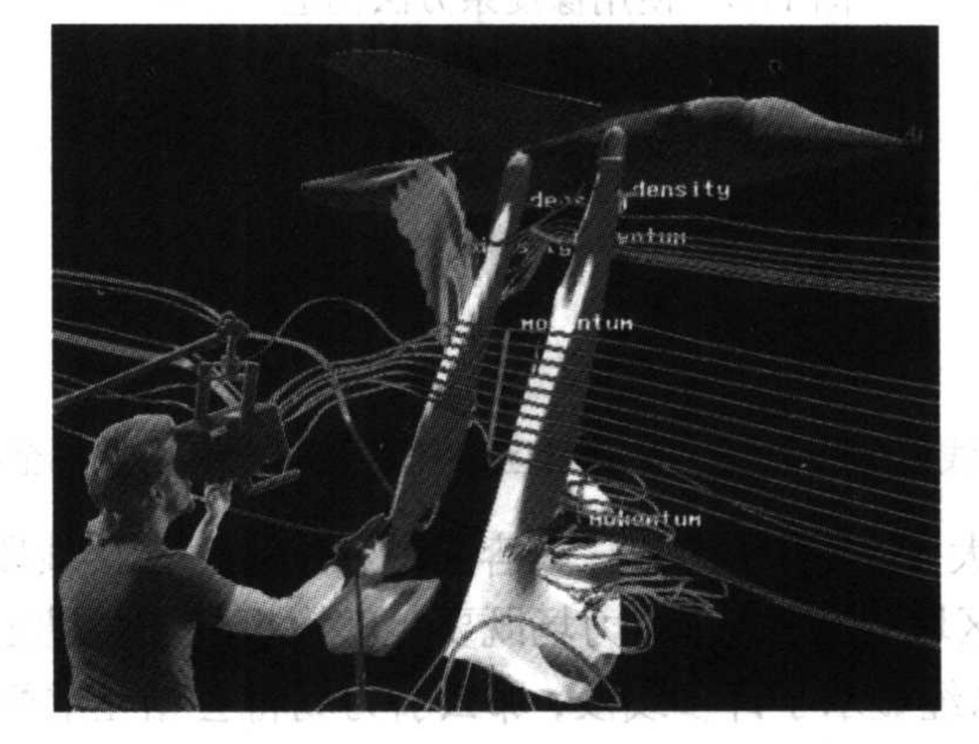


图 11.12 使用称为BOOM（Fake Space实验室）的头盔立体显示器和数据手套（VPL公司），研究员交互地操纵 Harrier 喷气式飞机四周的不稳定气流中的探针。软件由 Steve Bryson 开发，数据由 Harrier 提供（NASA Ames 研究中心的 Sam Uselton 提供）

另一个生成虚拟场景的方法是，在光栅监视器上显示两个在不同刷新周期内交替出现的立体投影图，然后使用立体眼镜来观察此景。交互式的对象操纵仍然可以通过数据手套和跟踪设备（监视数据手套相对于场景中对象的位置和角度）来实现。

11.6 OpenGL 支持交互式输入设备的函数

在 OpenGL 程序中，交互设备输入由 Utility Toolkit（GLUT）中的子程序处理，因为这些子程序需要与一个窗口系统连接。GLUT 中有从标准输入设备（包括鼠标、键盘、数据板、空间球、按钮盒和拨号盘）接受输入的函数。对每一种设备指定一个程序（回调函数）来处理从该设备发生的

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

输入。这些GLUT命令与其他的GLUT语句一起放置在main程序中。此外，来自基本库和GLU库的函数的组合也可以与GLUT的鼠标函数一起处理拾取输入。

11.6.1　GLUT 鼠标函数

我们用以下函数来指定（“登记”）一个当鼠标指针在窗口之内并且鼠标按键被按下或松开时调用的函数：

```
glutMouseFunc (mouseFcn);
```

这个名为mouseFcn的鼠标回调函数有四个参数：

```
void mouseFcn (GLint button, GLint action, GLint xMouse, GLint yMouse)
```

参数button的取值为GLUT定义鼠标按键的三个符号常量中的一个。button的允许值为GLUT_LEFT_BUTTON、GLUT_MIDDLE_BUTTON和GLUT_RIGHT_BUTTON。（对于两键鼠标，button允许为GLUT_LEFT_BUTTON和GLUT_RIGHT_BUTTON；对于单键鼠标，则button只能为GLUT_LEFT_BUTTON。）参数action的取值也是一个符号常量，指出我们要用哪种按键行为来触发鼠标激活事件。action的允许值为GLUT_DOWN或GLUT_UP，取决于我们需要通过按下还是松开鼠标键来启动一个行为。调用mouseFcn将返回鼠标光标在窗口中的位置坐标(xMouse, yMouse)。这是相对于窗口左上角的位置。xMouse是光标到窗口左边界的像素距离，yMouse是光标到窗口上边界的像素距离。

当屏幕光标在窗口内，激活鼠标按键可以选择显示图形元素（如点、线或填充区）的位置。通过比较返回的屏幕坐标与场景内显示对象的坐标范围，我们也可以把鼠标当做拾取设备。然而，OpenGL没有提供以鼠标作为拾取设备的子程序。我们将在后面的章节讨论这些子程序。

下面给出使用glutMouseFunc子程序的一个简单的例子。这个例子在窗口中每次按下鼠标左键时，就在鼠标光标所在位置画一个尺寸为3的红点。因为OpenGL基本函数的坐标原点在窗口的左上角，我们需要在过程mousePtPlot中翻转返回的yMouse值。

```
#include <GL/glut.h>

GLsizei winWidth = 400, winHeight = 300; // Initial display-window size.

void init (void)
{
   glClearColor (0.0, 0.0, 1.0, 1.0) // Set display-window color to blue.

   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void displayFcn (void)
{
   glClear (GL_COLOR_BUFFER_BIT); // Clear display window.

   glColor3f (1.0, 0.0, 0.0);     // Set point color to red.
   glPointSize (3.0);             // Set point size to 3.0.
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   /* Reset viewport and projection parameters */
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    glViewport (0, 0, newWidth, newHeight);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ( );
    gluOrtho2D (0.0, GLdouble (newWidth), 0.0, GLdouble (newHeight));

    /* Reset display-window size parameters. */
    winWidth = newWidth;
    winHeight = newHeight;
}

void plotPoint (GLint x, GLint y)
{
   glBegin (GL_POINTS);
      glVertex2i (x, y);
   glEnd ( );
}

void mousePtPlot (GLint button, GLint action, GLint xMouse, GLint yMouse)
{
   if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN)
      plotPoint (xMouse, winHeight - yMouse);

   glFlush ( );
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (100, 100);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Mouse Plot Points");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);
   glutMouseFunc (mousePtPlot);

   glutMainLoop ( );
}
```

下一个例子用鼠标输入来选择直线段的端点位置。选中的直线段首尾相连，展示了交互构造一条折线的过程。初始时，必须用鼠标左键在窗口内选择两个位置来生成第一段直线段。之后，每一个新选择的位置就可以与上一次选择的位置构成一条新的直线段。图11.13显示了这个例子的输出。

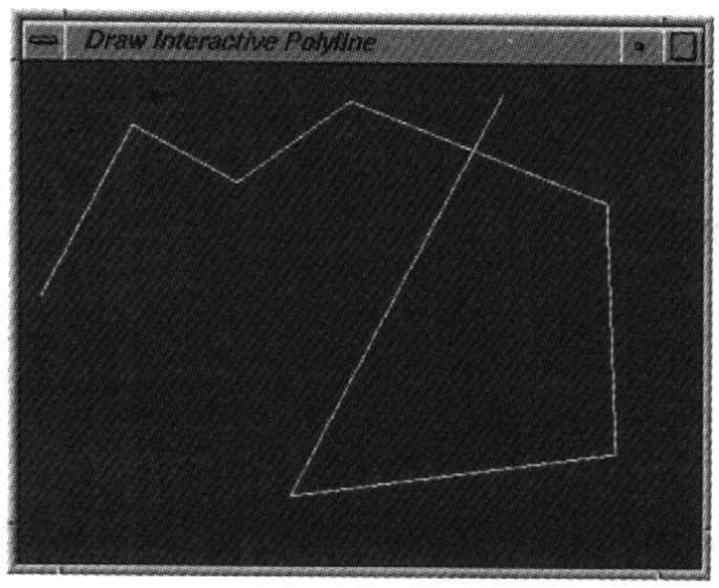


图 11.13 交互式鼠标程序 polyline 的示例输出

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
#include <GL/glut.h>

GLsizei winWidth = 400, winHeight = 300; // Initial display-window size.
GLint endPtCtr = 0;                       // Initialize line endpoint counter.

class scrPt {
public:
   GLint x, y;
};

void init (void)
{
   glClearColor (0.0, 0.0, 1.0, 1.0) // Set display-window color to blue.

   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void displayFcn (void)
{
   glClear (GL_COLOR_BUFFER_BIT);
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   /* Reset viewport and projection parameters */
   glViewport (0, 0, newWidth, newHeight);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );
   gluOrtho2D (0.0, GLdouble (newWidth), 0.0, GLdouble (newHeight));

   /* Reset display-window size parameters. */
   winWidth = newWidth;
   winHeight = newHeight;
}

void drawLineSegment (scrPt endPt1, scrPt endPt2)
{
   glBegin (GL_LINES);
      glVertex2i (endPt1.x, endPt1.y);
      glVertex2i (endPt2.x, endPt2.y);
   glEnd ( );
}

void polyline (GLint button, GLint action, GLint xMouse, GLint yMouse)
{
   static scrPt endPt1, endPt2;

   if (ptCtr == 0) {
      if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN) {
         endPt1.x = xMouse;
         endPt1.y = winHeight - yMouse;
         ptCtr = 1;
      }
      else
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        if (button == GLUT_RIGHT_BUTTON) // Quit the program.
           exit (0);
    }
    else
       if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN) {
          endPt2.x = xMouse;
          endPt2.y = winHeight - yMouse;
          drawLineSegment (endPt1, endPt2);

          endPt1 = endPt2;
       }
       else
          if (button == GLUT_RIGHT_BUTTON) // Quit the program.
             exit (0);
    glFlush ( );
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (100, 100);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Draw Interactive Polyline");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);
   glutMouseFunc (polyline);

   glutMainLoop ( );
}
```

可以使用的另一个 GLUT 鼠标子程序是

```
glutMotionFunc (fcnDoSomething);
```

当鼠标在窗口内移动并且一个或多个鼠标按键被激活时，这个例程调用fcnDoSomething。被调用的fcnDoSomething 函数有两个参数：

```
void fcnDoSomething (GLint xMouse, GLint yMouse)
```

其中(xMouse, yMouse)是当鼠标被移动并且按键被按下时，鼠标光标相对于窗口左上角的位置。

类似地，当鼠标在窗口内移动而鼠标键并未被按下时，我们也可以执行一些动作：

```
glutPassiveMotionFunc (fcnDoSomethingElse);
```

同样，相对于窗口左上角的鼠标位置坐标(xMouse, yMouse)将传递给 fcnDoSomethingElse。

11.6.2 GLUT 键盘函数

对于键盘输入，我们用以下函数指定一个当键盘上的一个键被按下时调用的函数。

```
glutKeyboardFunc (keyFcn);
```

被指定的函数有三个参数：

```
void keyFcn (GLubyte key, GLint xMouse, GLint yMouse)
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

参数 key 的取值是一个字符值或者对应的 ASCII 编码。返回的鼠标光标在窗口内的位置坐标(xMouse, yMouse)是相对于窗口左上角的。当一个指定的按键被按下时，我们就可以用鼠标位置来启动某些行为。

下面的代码给出一个简单的使用键盘输入的曲线绘制程序。按住“c”键，在窗口内移动鼠标就可以产生一条手绘曲线。例子在记录下来的鼠标位置上显示一串红点。如果鼠标移动缓慢，就可以得到一条实曲线。鼠标按键在此例中没有任何作用。图 11.14 是这个示例程序的输出。

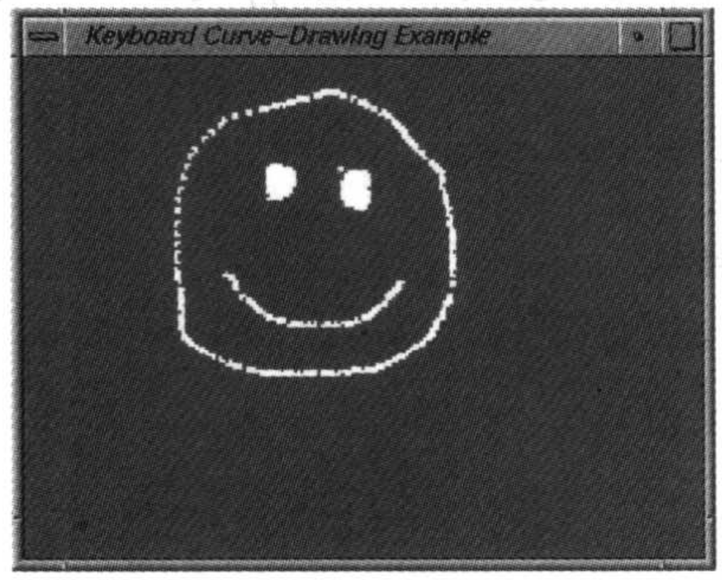


图 11.14　用程序 curveDrawing 输出手绘曲线的例子

```
#include <GL/glut.h>

GLsizei winWidth = 400, winHeight = 300; // Initial display-window size.

void init (void)
{
   glClearColor (0.0, 0.0, 1.0, 1.0); // Set display-window color to blue.

   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void displayFcn (void)
{
   glClear (GL_COLOR_BUFFER_BIT); // Clear display window.

   glColor3f (1.0, 0.0, 0.0);     // Set point color to red.
   glPointSize (3.0);             // Set point size to 3.0.
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   /* Reset viewport and projection parameters */
   glViewport (0, 0, newWidth, newHeight);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );
   gluOrtho2D (0.0, GLdouble (newWidth), 0.0, GLdouble (newHeight));

   /* Reset display-window size parameters. */
   winWidth = newWidth;
   winHeight = newHeight;
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void plotPoint (GLint x, GLint y)
{
   glBegin (GL_POINTS);
      glVertex2i (x, y);
   glEnd ( );
}

/* Move cursor while pressing c key enables freehand curve drawing. */
void curveDrawing (GLubyte curvePlotKey, GLint xMouse, GLint yMouse)
{
   GLint x = xMouse;
   GLint y = winHeight - yMouse;

   switch (curvePlotKey)
   {
      case 'c':
         plotPoint (x, y);
         break;
      default:
         break;
   }
   glFlush ( );
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (100, 100);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Keyboard Curve-Drawing Example");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);
   glutKeyboardFunc (curveDrawing);

   glutMainLoop ( );
}
```

我们可以使用以下命令指定对于功能键、方向键以及其他特殊键的处理函数：

```
glutSpecialFunc (specialKeyFcn);
```

被指定的函数也有三个参数：

```
void specialKeyFcn (GLint specialKey, GLint xMouse, GLint yMouse)
```

参数 specialKey 的取值是具有整数值的 GLUT 符号常量。功能键的符号常量从 GLUT_KEY_F1 到 GLUT_KEY_F12。方向键的符号常量类似GLUT_KEY_UP和GLUT_KEY_RIGHT。其他特殊键（如“page up”、“end”和“insert”键）用 GLUT_KEY_PAGE_DOWN、GLUT_KEY_HOME 等指定。“backspace”、“delete”和“escape”键通过 glutKeyboardFunc 用它们的 ASCII 编码指定，分别为 8、127 和 27。

以下代码展示了一个同时支持鼠标、键盘和功能键的交互程序。鼠标输入用于选择一个红色正方形的左下角的位置。键盘输入用于缩放正方形的大小。每次单击鼠标左键生成一个新的正方形。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
#include <GL/glut.h>
#inclue <stdlib.h>

GLsizei winWidth = 400, winHeight = 300; // Initial display-window size.
GLint edgeLength = 10;                    // Initial edge length for square.

void init (void)
{
   glClearColor (0.0, 0.0, 1.0, 1.0) // Set display-window color to blue.

   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void displayFcn (void)
{
   glClear (GL_COLOR_BUFFER_BIT);    // Clear display window.

   glColor3f (1.0, 0.0, 0.0);        // Set fill color to red.
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   /* Reset viewport and projection parameters */
   glViewport (0, 0, newWidth, newHeight);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );
   gluOrtho2D (0.0, GLdouble (newWidth), 0.0, GLdouble (newHeight));

   /* Reset display-window size parameters. */
   winWidth = newWidth;
   winHeight = newHeight;
}

/* Display a red square with a selected edge-length size. */
void fillSquare (GLint button, GLint action, GLint xMouse, GLint yMouse)
{
   GLint x1, y1, x2, y2;

   /* Use left mouse button to select a position for the
    * lower-left corner of the square.
    */
   if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN)
   {
      x1 = xMouse;
      y1 = winHeight - yMouse;
      x2 = x1 + edgeLength;
      y2 = y1 + edgeLength;
      glRecti (x1, y1, x2, y2);
   }
   else
      if (button == GLUT_RIGHT_BUTTON) // Use right mouse button to quit.
         exit (0);
   glFlush ( );
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
/* Use keys 2, 3, and 4 to enlarge the square. */
void enlargeSquare (GLubyte sizeFactor, GLint xMouse, GLint yMouse)
{
   switch (sizeFactor)
   {
      case '2':
         edgeLength *= 2;
         break;
      case '3':
         edgeLength *= 3;
         break;
      case '4':
         edgeLength *= 4;
         break;
      default:
         break;
   }
}

/* Use function keys F2 and F4 for reduction factors 1/2 and 1/4. */
void reduceSquare (GLint reductionKey, GLint xMouse, GLint yMouse)
{
   switch (reductionKey)
   {
      case GLUT_KEY_F2:
         edgeLength /= 2;
         break;
      case GLUT_KEY_F3:
         edgeLength /= 4;
         break;
      default:
         break;
   }
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (100, 100);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Display Squares of Various Sizes");

   init ( );
   glutDisplayFunc (displayFcn);
   glutReshapeFunc (winReshapeFcn);
   glutMouseFunc (fillSquare);
   glutKeyboardFunc (enlargeSquare);
   glutSpecialFunc (reduceSquare);

   glutMainLoop ( );
}
```

11.6.3 GLUT 数据板函数

数据板通常当鼠标光标位于窗口内时才被激活。数据板输入的按键事件由以下函数记录：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glutTabletButtonFunc (tabletFcn);
```

被调用函数的参数与鼠标回调函数的参数相似：

```
void tabletFcn(GLint tabletButton, GLint action, GLint xTablet, GLint yTablet)
```

参数tabletButton的取值是整型标志符，如1、2、3等。按键行为参数action 的取值为GLUT_UP或GLUT_DOWN。返回参数 xTablet 和 yTablet 是数据板坐标。我们可以用以下函数指定数据板的有效按键数目：

```
glutDeviceGet (GLUT_NUM_TABLET_BUTTONS);
```

数据板光笔或光标的移动由以下函数处理：

```
glutTabletMotionFunc (tabletMotionFcn);
```

其中被调用函数有两个参数：

```
void tabletMotionFcn (GLint xTablet, GLint yTablet)
```

参数返回值 xTablet 和 yTablet 是数据板表面的坐标。

11.6.4　GLUT 空间球函数

对于一个选中的窗口，以下函数指定当空间球按键被激活时的操作。

```
glutSpaceballButtonFunc (spaceballFcn);
```

回调函数有两个参数：

```
void spaceballFcn (GLint spaceballButton, GLint action)
```

空间球按键的标识方法与数据板按键相同，参数 action 的取值也是 GLUT_UP 或 GLUT_DOWN。通过以GLUT_NUM_SPACEBALL_BUTTONS为参数调用glutDeviceGet来指定空间球的有效按键数目。

当鼠标光标位于窗口之内时，空间球的平移运动通过以下函数调用来记录：

```
glutSpaceballMotionFunc (spaceballTranlFcn);
```

三维平移距离作为参数传递给被调用函数：

```
void spaceballTranslFcn (GLint tx, GLint ty, GLint tz)
```

规范化后的平移距离取值从 -1000 到 1000。

类似地，空间球的旋转由以下函数记录：

```
glutSpaceballRotateFunc (spaceballRotFcn);
```

三维旋转角度作为参数传递给被调用函数：

```
void spaceballRotFcn (GLint thetaX, GLint thetaY, GLint thetaZ)
```

11.6.5　GLUT 按钮盒函数

按钮盒输入通过以下函数获取：

```
glutButtonBoxFunc (buttonBoxFcn);
```

按键的激活事件传递给被调用的函数：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
void buttonBoxFcn (GLint button, GLint action);
```

按键用整数值来标识。键盘行为分为 GLUT_UP 或 GLUT_DOWN 两种。

11.6.6 GLUT 拨号盘函数

旋转拨号盘产生的输入由以下函数处理：

```
glutDialsFunc (dialsFcn);
```

通过回调函数可以标识拨号盘并且得到旋转的角度：

```
void dialsFcn (GLint dial, GLint degreeValue);
```

拨号盘用整数值来标识。拨号盘的旋转量表示为整数的角度值。

11.6.7 OpenGL 拾取操作

在 OpenGL 程序中，我们可以通过指点屏幕位置来交互选择对象。然而，OpenGL 中的拾取操作并不是很直接。

基本上，拾取操作是通过一个修正观察体实现的，而这个观察体根据一个指定的拾取窗口形成。我们对一个场景内的对象用整数进行标识。所有与观察体相交的对象的标识符都保存在一个拾取缓冲区数组中。因此，为了使用 OpenGL 的拾取功能，我们必须在程序中包含以下过程：

- 创建并显示一个场景
- 拾取一个屏幕位置，并且在鼠标回调函数内部进行如下操作：
  - 设置一个拾取缓冲区
  - 激活拾取操作（选择模式）
- 为对象标识符初始化一个 ID 名称栈
- 保存当前的观察和几何变换矩阵
- 指定鼠标输入的拾取窗口
- 给对象分配标识符然后用观察体再处理一次场景，从而将拾取信息存储到拾取缓冲区中
- 恢复原来的观察和几何变换矩阵
- 确定被拾取的对象的数目，然后返回正常的绘制模式
- 处理拾取信息

我们也可以修改上述的过程来在不使用鼠标交互输入的情况下选择对象。这是通过指定修正观察体的顶点（代替指定一个拾取窗口）来实现的。

以下命令用于设置一个拾取缓冲区：

```
glSelectBuffer (pickBuffSize, pickBuffer);
```

参数 pickBuffer 指定一个具有 pickBuffSize 个元素的整型数组。函数 glSelectBuffer 必须在激活 OpenGL 拾取操作（选择模式）之前被调用。对于每次拾取选中的每一个对象，向拾取缓冲区数组中添加一个整数信息记录。拾取缓冲区可以存放多个记录。记录的数目取决于拾取窗口的尺寸和位置。拾取缓冲区中的每条记录包含如下信息：

(1) 对象在名称栈中的位置，即名称栈中位于该对象之下（包括该对象）的标识符数目。

(2) 拾取对象的最小深度。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

(3) 拾取对象的最大深度。

(4) 从名称栈中第一个标识符（底部）到拾取对象的标识符之间的所有标识符的列表。

存储在拾取缓冲区中的整型深度值的取值范围是 0 到 1.0，乘以 $2^{32}-1$。

OpenGL 的拾取操作由以下命令激活：

```
glRenderMode (GL_SELECT);
```

这个命令将当前模式设置为选择模式，即一个场景虽然通过观察流水线的处理，但是并不存储在帧缓冲区中。在正常的绘制模式下应该被显示的每个对象的信息记录被放置在拾取缓冲区中。此外，这个命令返回拾取对象的个数，即在拾取缓冲区中信息记录的个数。调用 glRenderMode（GL_RENDER）就可以将当前模式设置为正常的绘制模式（默认模式）。这个命令的第三个参数选项是 GL_FEEDBACK。这种反馈模式也不显示对象，而是将对象的坐标及其他信息存储到一个反馈缓冲区中。这种模式用于查询一个场景中对象的信息，如元素类型、属性等。

以下命令用于激活拾取操作的整型 ID 名称栈：

```
glInitNames ( );
```

ID 名称栈只能在选择模式下被使用，其初始状态为空。以下函数用于向栈中放入一个无符号整数值：

```
glPushName (ID);
```

这个函数将参数 ID 放在栈顶，栈中已有的元素依次下压。以下函数用于替换栈顶元素：

```
glLoadName (ID);
```

但是我们不能用这个函数向一个空栈中放入元素。以下函数用于弹出栈顶元素：

```
glPopName ( );
```

以下 GLU 函数定义了选定视区内的一个拾取窗口：

```
gluPickMatrix (xPick, yPick, widthPick, heightPick, vpArray);
```

参数 xPick 和 yPick 给出相对于视区左下角的拾取窗口中心的双精度浮点屏幕坐标值。如果坐标通过鼠标输入给出，因为鼠标坐标是相对于视区左上角的，我们需要翻转输入的yMouse值。widthPick 和 heightPick 分别指定双精度浮点数值的拾取窗口的宽和高。参数 vpArray 指定一个包含当前视区的坐标位置和尺寸等参数的整型数组。我们可以通过 glGetIntegerv 函数（参见 6.4 节）获得视区的参数。拾取窗口即可作为裁剪窗口构造观察变换的修正观察体。与修正观察体相交的对象的信息被放入拾取缓冲区中。

我们以下面的程序为例说明OpenGL的拾取操作。这个程序显示三个彩色的矩形（参见图11.15）。我们用 5 × 5 的拾取窗口，拾取窗口的中心位置由鼠标输入决定。因此，我们需要用视区的高度（vpArray 中的第四个参数）来翻转 yMouse 的值。红色矩形的 ID 为 30，蓝色矩形的 ID 为 10，绿色矩形的 ID 为 20。根据鼠标输入的位置，我们可以一次拾取 0 到 3 个矩形。矩形的 ID 按照红、蓝、绿颜色的顺序被放入 ID 名称栈。因此，我们既可以使用 ID（拾取记录中的最后一项）也可以使用名称栈中的位置号（拾取记录中的第一项）来处理被拾取的矩形。例如，如果栈位置号为 2，则我们拾取了蓝色矩形，并且在记录最后列出了两个矩形的标识符。我们也可以使用被拾取对象的标识符。这个示例程序中列出了拾取缓冲区的内容。矩形均在 xy 平面上定义，因此所有深度值为 0。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

例11.1列出了当鼠标输入位置在红色和绿色矩形边界处时的输出示例。这个示例程序没有提供终止机制，因此它可以处理任意数量的鼠标输入。

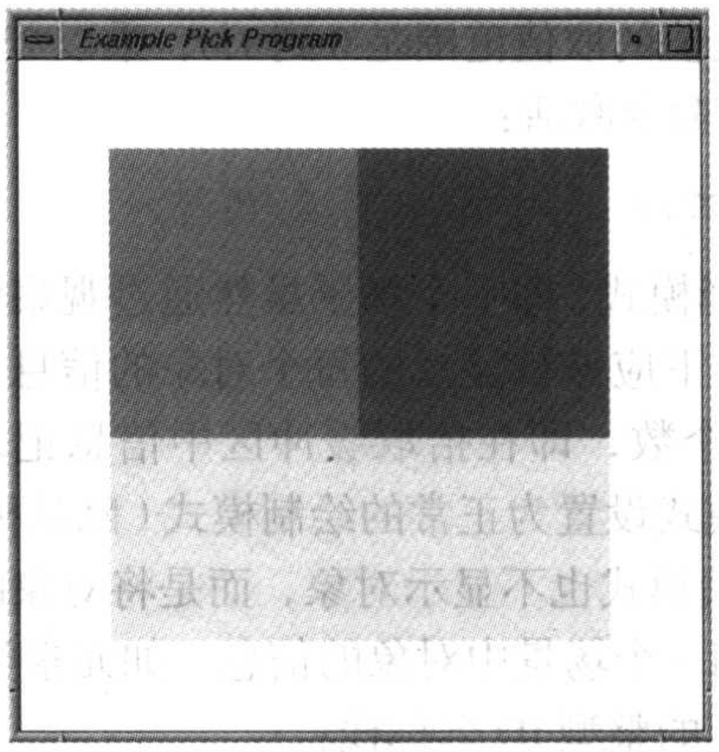


图 11.15 拾取示例程序显示的三个彩色矩形

```
#include <GL/glut.h>
#include <stdio.h>

const GLint pickBuffSize = 32;

/* Set initial display-window size. */
GLsizei winWidth = 400, winHeight = 400;

void init (void)
{
   /* Set display-window color to white. */
   glClearColor (1.0, 1.0, 1.0, 1.0);
}

/* Define 3 rectangles and associated IDs. */
void rects (GLenum mode)
{
   if (mode == GL_SELECT)
      glPushName (30);   // Red rectangle.
   glColor3f (1.0, 0.0, 0.0);
   glRecti (40, 130, 150, 260);

   if (mode == GL_SELECT)
      glPushName (10);   // Blue rectangle.
   glColor3f (0.0, 0.0, 1.0);
   glRecti (150, 130, 260, 260);

   if (mode == GL_SELECT)
      glPushName (20);   // Green rectangle.
   glColor3f (0.0, 1.0, 0.0);
   glRecti (40, 40, 260, 130);
}

/* Print the contents of the pick buffer for each mouse selection. */
void processPicks (GLint nPicks, GLuint pickBuffer [ ])
{
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    GLint j, k;
    GLuint objID, *ptr;

    printf (" Number of objects picked = %d\n", nPicks);
    printf ("\n");
    ptr = pickBuffer;

    /* Output all items in each pick record. */
    for (j = 0; j < nPicks; j++) {
       objID = *ptr;

       printf (" Stack position = %d\n", objID);
       ptr++;

       printf (" Min depth = %g,", float (*ptr/0x7fffffff));
       ptr++;

       printf (" Max depth = %g\n", float (*ptr/0x7fffffff));
       ptr++;

       printf (" Stack IDs are: \n");
       for (k = 0; k < objID; k++) {
          printf (" %d ",*ptr);
          ptr++;
       }
       printf ("\n\n");
    }
}

void pickRects (GLint button, GLint action, GLint xMouse, GLint yMouse)
{
    GLuint pickBuffer [pickBuffSize];
    GLint nPicks, vpArray [4];

    if (button != GLUT_LEFT_BUTTON || action != GLUT_DOWN)
       return;

    glSelectBuffer (pickBuffSize, pickBuffer); // Designate pick buffer.
    glRenderMode (GL_SELECT);                  // Activate picking operations.
    glInitNames ( );                           // Initialize the object-ID stack.

    /* Save current viewing matrix. */
    glMatrixMode (GL_PROJECTION);
    glPushMatrix ( );
    glLoadIdentity ( );

    /* Obtain the parameters for the current viewport. Set up
     * a 5 by 5 pick window, and invert the input yMouse value
     * using the height of the viewport, which is the fourth
     * element of vpArray.
     */
    glGetIntegerv (GL_VIEWPORT, vpArray);
    gluPickMatrix (GLdouble (xMouse), GLdouble (vpArray [3] - yMouse),
                   5.0, 5.0, vpArray);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
    gluOrtho2D (0.0, 300.0, 0.0, 300.0);
    rects (GL_SELECT); // Process the rectangles in selection mode.

    /* Restore original viewing matrix. */
    glMatrixMode (GL_PROJECTION);
    glPopMatrix ( );

    glFlush ( );

    /* Determine the number of picked objects and return to the
     * normal rendering mode.
     */
    nPicks = glRenderMode (GL_RENDER);

    processPicks (nPicks, pickBuffer); // Process picked objects.

    glutPostRedisplay ( );
}

void displayFcn (void)
{
    glClear (GL_COLOR_BUFFER_BIT);
    rects (GL_RENDER);                // Display the rectangles.
    glFlush ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
    /* Reset viewport and projection parameters. */
    glViewport (0, 0, newWidth, newHeight);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ( );

    gluOrtho2D (0.0, 300.0, 0.0, 300.0);
    glMatrixMode (GL_MODELVIEW);

    /* Reset display-window size parameters. */
    winWidth = newWidth;
    winHeight = newHeight;
}

void main (int argc, char** argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition (100, 100);
    glutInitWindowSize (winWidth, winHeight);
    glutCreateWindow ("Example Pick Program");

    init ( );
    glutDisplayFunc (displayFcn);
    glutReshapeFunc (winReshapeFcn);
    glutMouseFunc (pickRects);

    glutMainLoop ( );
}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

例 11.1　程序 pickrects 的输出示例

```
Number of objects picked = 2

  Stack position = 1
  Min depth = 0,    Max depth = 0
  Stack IDs are:
  30

  Stack position = 3
  Min depth = 0,    Max depth = 0
  Stack IDs are:
  30    10    20
```

11.7　OpenGL 的菜单功能

除了输入设备子程序之外，GLUT还包括多种向程序中添加简单的弹出式菜单的函数。使用这些函数可以设置和访问多种菜单和子菜单。GLUT 菜单命令和其他 GLUT 函数一起放置在 main 程序中。

11.7.1　创建 GLUT 菜单

以下语句创建一个弹出式菜单：

```
glutCreateMenu (menuFcn);
```

其中参数menuFcn是当一个菜单项被选中时调用的函数的名字。这个函数有一个整型参数对应于选中项的位置。

```
Void menuFcn (GLint menuItemNumber)
```

传递给 menuItemNumber 的整数值被 menuFcn 用来执行某些操作。当一个菜单被创建时，它被关联到当前的窗口。

一旦指定了菜单项被选中时调用的函数，列在菜单上的菜单项也必须被指定。我们使用一系列的语句来设定每个菜单项的名称和位置。这些语句的通用形式如下：

```
glutAddMenuEntry (charString, menuItemNumber);
```

参数charString设定显示在菜单上的文字，参数menuItemNumber设定该菜单项在菜单中的位置。例如，下列语句创建一个具有两个菜单项的菜单。

```
glutCreateMenu (menuFcn);
    glutAddMenuEntry ("First Menu Item", 1);
    glutAddMenuEntry ("Second Menu Item", 2);
```

然后，我们必须通过以下函数指定用来选择菜单项的鼠标键：

```
glutAttachMenu (button);
```

其中 button 的取值是三个 GLUT 符号常量中的一个，分别指定鼠标的左、中、右键。

以下示例程序展示了如何创建和使用GLUT菜单，它提供两个选项来显示对一个三角形的内部填充。最初，三角形由两个白点和一个红点定义，并且内部填充颜色由三个顶点颜色的插值决定。我们用 glShadeModel 函数（参见 4.14 节和 10.20 节）来选择多边形填充的方式是同色填充还是用顶点颜色插值填充（Gouraud绘制）。示例程序中创建了一个菜单用于在这两种模式之间选择。当鼠标

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

光标位于窗口内部时，用鼠标右键就可以调出菜单进行选择。如图 11.16 所示，此弹出式菜单显示在鼠标光标的左上角。当鼠标光标移动到一个菜单项上时，该项就会被突出显示。如果松开鼠标右键，突出显示的菜单项就被选中。如果选中“Solid-Color Fill”项，就用最后一个指定的顶点（红点）的颜色来填充三角形。在菜单显示过程 fillOption 的最后，我们调用 glutPostRedisplay 命令（参见 6.4 节）来指示当菜单显示时，三角形必须被重画。

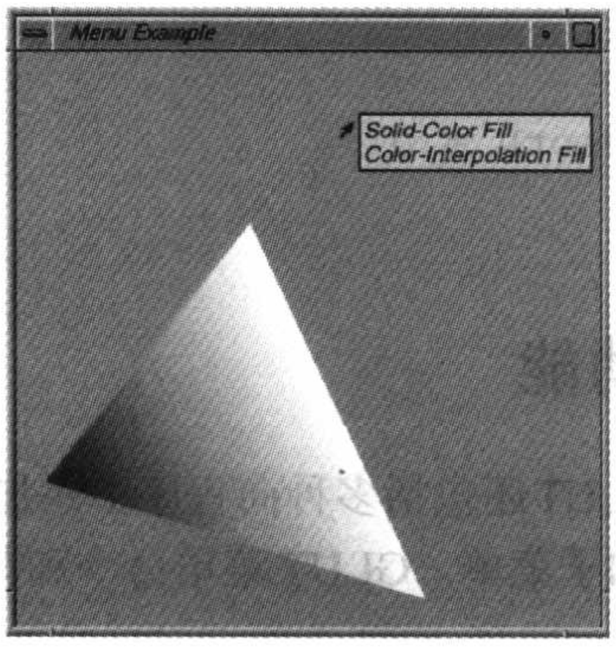


图 11.16 菜单示例程序显示的 OpenGL 弹出式菜单

```
#include <GL/glut.h>

GLsizei winWidth = 400, winHeight = 400; // Initial display-window size.

GLfloat red = 1.0, green = 1.0, blue = 1.0; // Initial triangle color: white.
GLenum fillMode = GL_SMOOTH; // Initial polygon fill: color interpolation.

void init (void)
{
   glClearColor (0.6, 0.6, 0.6, 1.0); // Set display-window color to gray.

   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (0.0, 300.0, 0.0, 300.0);
}

void fillOption (GLint selectedOption)
{
   switch (selectedOption) {
      case 1: fillMode = GL_FLAT; break;    // Flat surface rendering.
      case 2: fillMode = GL_SMOOTH; break; // Gouraud rendering.
   }
   glutPostRedisplay ( );
}

void displayTriangle (void)
{
   glClear (GL_COLOR_BUFFER_BIT);

   glShadeModel (fillMode);        // Set fill method for triangle.
   glColor3f (red, green, blue); // Set color for first two vertices.

   glBegin (GL_TRIANGLES);
      glVertex2i (280, 20);
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
      glVertex2i (160, 280);
      glColor3f (red, 0.0, 0.0); // Set color of last vertex to red.
      glVertex2i (20, 100);
   glEnd ( );

   glFlush ( );
}

void reshapeFcn (GLint newWidth, GLint newHeight)
{
   glViewport (0, 0, newWidth, newHeight);

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );
   gluOrtho2D (0.0, GLfloat (newWidth), 0.0, GLfloat (newHeight));

   displayTriangle ( );
   glFlush ( );
}

void main (int argc, char **argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (200, 200);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Menu Example");

   init ( );
   glutDisplayFunc (displayTriangle);

   glutCreateMenu (fillOption);              // Create pop-up menu.
      glutAddMenuEntry ("Solid-Color Fill", 1);
      glutAddMenuEntry ("Color-Interpolation Fill", 2);

   /* Select a menu option using the right mouse button. */
   glutAttachMenu (GLUT_RIGHT_BUTTON);

   glutReshapeFunc (reshapeFcn);

   glutMainLoop ( );
}
```

11.7.2　创建和管理多个 GLUT 菜单

当一个菜单被创建时，它被关联到当前的窗口（参见 6.4 节）。我们可以为一个窗口创建多个菜单，也可以为不同的窗口创建不同的菜单。每个菜单在创建时分配到一个整数标识符。菜单标识符按照菜单创建的顺序从 1 开始编号。glutCreateMenu 子程序返回这个标识符。我们可以用以下语句记录新创建菜单的标识符。

```
menuID = glutCreateMenu (menuFcn);
```

最新创建的菜单成为当前窗口的**当前菜单**（current menu）。可以用以下命令激活一个菜单成为当前窗口的当前菜单。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
glutSetMenu (menuID);
```

menuID对应的菜单即成为当前菜单。当与此菜单关联的鼠标键在窗口内按下时，此菜单就会弹出。

以下命令用于清除一个菜单：

```
glutDestroyMenu (menuID);
```

如果menuID对应的菜单是窗口的当前菜单，那么即使存在其他的菜单，清除当前菜单后该窗口也没有当前菜单。

以下函数用于获得当前窗口的当前菜单的标识符：

```
currentMenuID = glutGetMenu ( );
```

如果当前窗口没有菜单，或者之前的当前菜单被glutDestroyMenu清除了，返回值为0。

11.7.3 创建GLUT子菜单

我们也可以将一个子菜单关联到一个菜单上。首先，用glutCreateMenu创建子菜单并且列出子菜单项；然后，将子菜单作为一个项添加到主菜单上。下列语句用于将一个子菜单添加到一个主菜单（或其他子菜单）的菜单项列表：

```
submenuID = glutCreateMenu (submenuFcn);
   glutAddMenuEntry ("First Submenu Item", 1);
   .
   .
   .
glutCreateMenu (menuFcn);
   glutAddMenuEntry ("First Menu Item", 1);
   .
   .
   .
   glutAddSubMenu ("Submenu Option", submenuID);
```

glutAddSubMenu函数也可以用来将子菜单添加到当前菜单。

下面的例子展示了创建子菜单的过程。这个例子由前面的菜单示例程序修改而来，增加了一个用于选择三角形前两个顶点颜色（蓝、绿和白）的子菜单。主菜单有三个项，其中第三个项的右边有一个箭头，指出如果突出显示该项会弹出一个新的菜单，如图11.17所示。主菜单处理函数和子菜单处理函数的最后都包含了glutPostRedisplay函数。

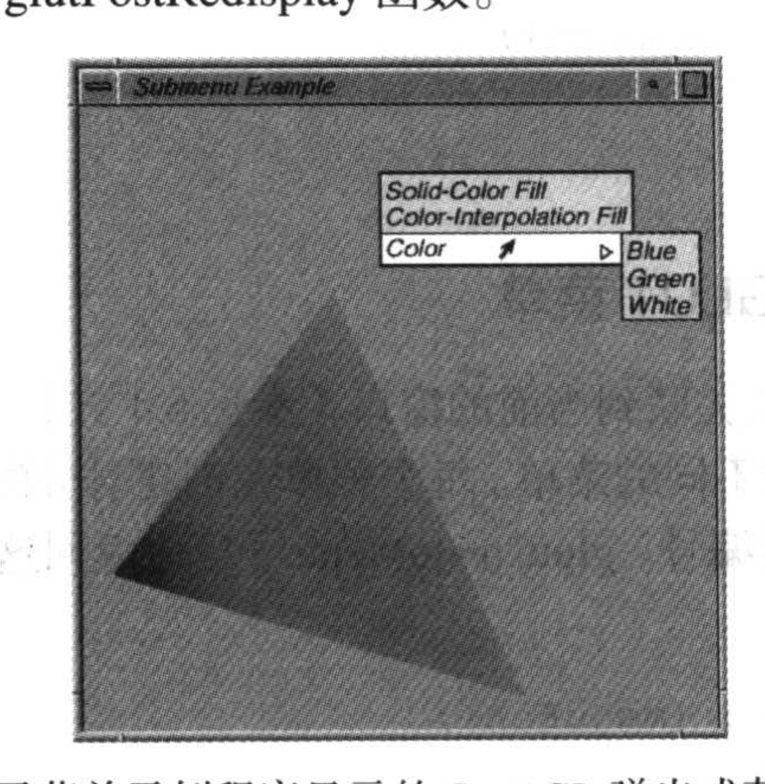


图11.17 子菜单示例程序显示的OpenGL弹出式菜单和子菜单

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
#include <GL/glut.h>

GLsizei winWidth = 400, winHeight = 400; // Initial display-window size.

GLfloat red = 1.0, green = 1.0, blue = 1.0; // Initial color values.
GLenum renderingMode = GL_SMOOTH;           // Initial fill method.

void init (void)
{
   glClearColor (0.6, 0.6, 0.6, 1.0); // Set display-window color to gray.

   glMatrixMode (GL_PROJECTION);
   gluOrtho2D (0.0, 300.0, 0.0, 300.0);
}

void mainMenu (GLint renderingOption)
{
   switch (renderingOption) {
      case 1: renderingMode = GL_FLAT;   break;
      case 2: renderingMode = GL_SMOOTH; break;
   }
   glutPostRedisplay ( );
}

/* Set color values according to the submenu option selected. */
void colorSubMenu (GLint colorOption)
{
   switch (colorOption) {
      case 1:
          red = 0.0; green = 0.0; blue = 1.0;
          break;
      case 2:
          red = 0.0; green = 1.0; blue = 0.0;
          break;
      case 3:
          red = 1.0; green = 1.0; blue = 1.0;
   }
   glutPostRedisplay ( );
}

void displayTriangle (void)
{
   glClear (GL_COLOR_BUFFER_BIT);

   glShadeModel (renderingMode); // Set fill method for triangle.
   glColor3f (red, green, blue); // Set color for first two vertices.

   glBegin (GL_TRIANGLES);
       glVertex2i (280, 20);
       glVertex2i (160, 280);
       glColor3f (1.0, 0.0, 0.0); // Set color of last vertex to red.
       glVertex2i (20, 100);
   glEnd ( );

   glFlush ( );
}

void reshapeFcn (GLint newWidth, GLint newHeight)
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
{
   glViewport (0, 0, newWidth, newHeight);

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );
   gluOrtho2D (0.0, GLfloat (newWidth), 0.0, GLfloat (newHeight));

   displayTriangle ( );
   glFlush ( );
}

void main (int argc, char **argv)
{
   GLint subMenu;                          // Identifier for submenu.

   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition (200, 200);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Submenu Example");

   init ( );
   glutDisplayFunc (displayTriangle);

   subMenu = glutCreateMenu (colorSubMenu);
      glutAddMenuEntry ("Blue", 1);
      glutAddMenuEntry ("Green", 2);
      glutAddMenuEntry ("White", 3);

   glutCreateMenu (mainMenu);  // Create main pop-up menu.
      glutAddMenuEntry ("Solid-Color Fill", 1);
      glutAddMenuEntry ("Color-Interpolation Fill", 2);
      glutAddSubMenu ("Color", subMenu);

   /* Select menu option using right mouse button. */
   glutAttachMenu (GLUT_RIGHT_BUTTON);

   glutReshapeFunc (reshapeFcn);

   glutMainLoop ( );
}
```

11.7.4 修改 GLUT 菜单

如果要改变用来选择菜单的鼠标键，首先要取消与当前鼠标键的关联，然后再关联一个新的键。以下函数用于取消关联：

```
glutDetachMenu (mouseButton);
```

参数 mouseButton 是之前关联到这个菜单的鼠标键（左、中、右）的 GLUT 符号常量。

也可以改变一个已有菜单中的某些项。例如，可以用以下函数删除当前菜单中的一个项：

```
glutRemoveMenuItem (itemNumber);
```

其中参数 itemNumber 被赋值为欲删除菜单项的整数标识符。

其他 GLUT 子程序支持更改已有菜单项的名称和状态。例如，我们可以修改菜单项的显示文本、改变菜单项的编号，或者将一个菜单项变成子菜单。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

11.8　图形用户界面的设计

现在所有的应用软件都包含由窗口、图标、菜单及其他功能构成的图形人机界面来辅助用户使用软件和处理问题。图形软件包为每一种应用（例如工程设计、建筑设计、绘图、商用图表、地质、经济、化学和物理等）设计了专门的交互对话，从而在特定的领域中可以用熟悉的术语来选择设计选项。关于设计图形用户界面的其他考虑事项包括适应多种熟练程度的用户、一致性、错误处理和反馈。

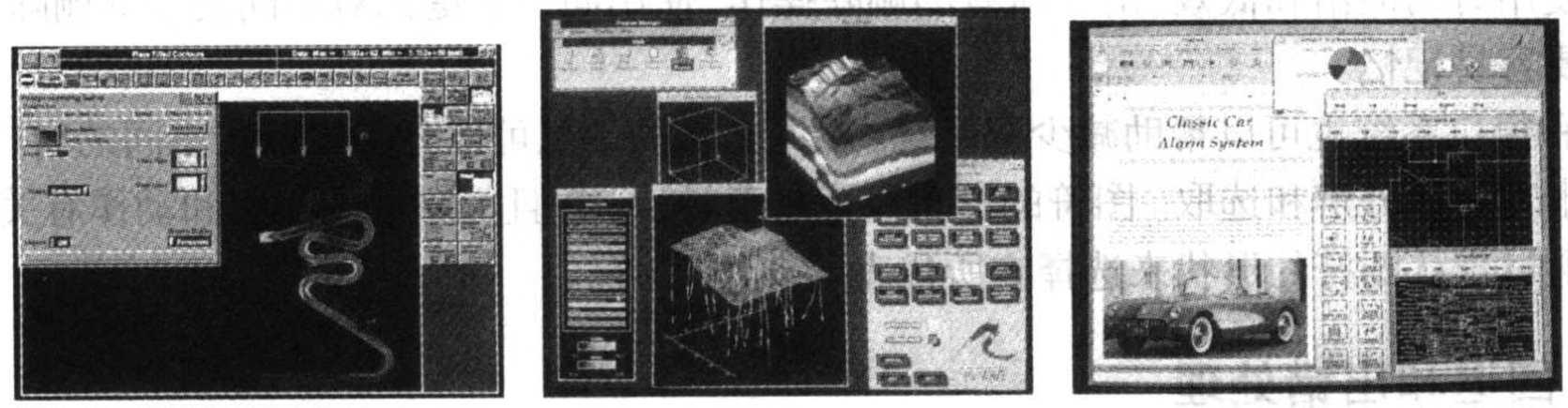
11.8.1　用户对话

对于某种特定的应用，用户模型（user' s model）是设计对话的基础。用户模型说明所设计的系统能做什么，应具备什么样的图形操作。这种模型指明了可以显示的对象类型以及如何管理对象。例如，如果一个图形系统是作为建筑设计的工具，则用户模型要说明如何使用该软件包来定位墙、门、窗和其他建筑成分，从而构造和显示大楼的视图。同样，对于一个设备布局系统，将对象定义为一组家具（桌子、椅子等），而应具备的操作功能则包括在指定的平面图内定位一件家具或移动对象。电路设计程序则可以在整个电路设计中使用电子或逻辑元件作为对象，提供增加元件、删除元件及有关的定位操作。

一般按照应用的语言来表达用户对话的所有信息。在建筑设计软件包中，这表示仅仅使用建筑术语来说明所有交互，不引入特殊的数据结构、计算机图形学术语或其他建筑设计师不熟悉的概念。

11.8.2　窗口和图符

图11.18给出了典型的具有窗口和图符的图形界面例子。可视化表示既可以用于应用中管理的对象，也可以用于对应用对象进行的操作。



(a)　(b)　(c)

图 11.18　使用窗口、图符和菜单的屏幕布局示例（(a)Intergraph公司提供，(b)Visual Numerics 公司提供，(c)Sun 公司提供）

窗口系统的标准功能包括打开和关闭窗口、对窗口重定位、缩放功能等，其他操作需要与滑动块、按键、菜单和图符等协同进行。某些通用系统可提供多个窗口管理程序，从而使不同风格的窗口可以同时在各自的管理程序控制下实现。窗口管理程序可以按特定的应用要求进行设计。

用来代表墙、门、窗或电路元件等对象的图符称为**应用图符**（application icon）。代表旋转、放大、成比例变化、裁剪和粘贴等动作的图符称为**控制图符**（control icon）或**命令图符**（command icon）。

11.8.3　适应多种熟练程度的用户

通常，交互式图形界面提供多种选择动作的方法。例如，选项的选择可以通过将光标指向一个图符，然后按下不同的鼠标键而实现；也可以通过下拉或弹出式菜单进行选择；还可以通过键入命令进行选择。这种功能使软件能适应不同熟练程度的用户。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于经验不足的用户，具有容易理解的操作加上详细提示的界面，要比具有复杂的操作集的界面高效得多。一组简化的菜单和选项比较容易学习和记忆，用户可以把注意力集中在应用上而不是在界面的细节上。对于一个没有经验的应用软件用户，简单的“指点-按键”操作是最容易的。因此，一般的界面都设法掩盖软件的复杂性，从而使初学者在使用该系统时不会因众多的细节而不知所措。

另一方面，有经验的用户需要操作的速度。这意味着较少的提示和来自键盘或多种“鼠标-按键”的输入。由于有经验的用户记住了常用动作的快捷键，因此经常通过功能键或同时按下组合键来选择这些动作。

同样，也可以分成几个层次来设计帮助功能，使初学者可以进行较为详细的对话，有经验的用户则可以减少或去掉提示和消息。帮助功能还可以包含一个或多个辅导性应用，从而介绍该软件的功能和使用方法。

11.8.4 一致性

一致性是界面设计中需要考虑的要点之一。例如，一个特定的图符应该始终只有一个含义而不能依靠上下文来代表多个动作或对象。其他的一致性例子有：菜单总是放在相同的关联位置，从而使用户不必寻找特定的选项；总是使用相同的组合键来代表同一个动作；总是使用一种颜色编码，从而使相同的颜色在不同情况下不会产生不同的含义。

11.8.5 减少记忆量

界面的操作应该组织成必须容易理解和记忆。模糊的、复杂的、不一致的和缩写的命令格式会导致软件使用时的混淆和低效。对于所有的删除操作，使用同一个键比对不同类型的删除操作使用不同的键要容易记忆。

图符和窗口系统也可以帮助减少记忆量。不同类型的信息可以分别显示在不同的窗口中，因而用户可以很容易地分辨和选取。图符的形状应该易于辨认，并且与应用的对象和动作有关。用户可以简单地通过辨认图符的形状来选择它所代表的动作。

11.8.6 回退和出错处理

在一系列操作过程中，回退和取消机制是用户界面的另一个共同的特点。有了纠正操作错误的功能，用户就可以放心地使用系统的各种功能。有的系统可以回退若干步操作，因此可以把系统回退到某些特定的位置。有些操作不能回退，例如，关闭了应用程序而没有保存更改。对于这种类型的操作，系统在执行请求的操作之前要求用户确认。

设计一些好的诊断程序和提供出错消息，可以帮助确定发生错误的原因。另外，界面还通过对可能导致错误的一些动作进行预测来减少错误的可能性。如果用户试图执行一些模棱两可或错误的操作，例如向多个对象应用同一个操作，则用户会获得警告消息。

11.8.7 反馈

用户界面的另一个重要特性是反馈，即对用户的操作要及时响应。这一点对经验不足的用户尤其重要。当用户输入一个操作后，界面必须给出某种响应。否则，用户可能会搞不清系统的进展情况以及是不是要重新输入。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

反馈可以按照多种形式给出，如醒目显示某个对象，出现某个图符或信息，或者用不同的颜色显示选中的菜单项。如果在短时间内不能完成某个操作，则可以显示一些信息（如闪烁的消息、时钟、沙漏或进程条）来告诉用户系统的进展情况。系统也可以在完成操作的过程中逐步显示部分结果，即最终结果按照一次一部分进行显示。系统还可以在执行一条指令的过程中，让用户输入其他的命令。

对于不同类型的反馈，可以设计专门的符号。例如，一个交叉符号、一张皱眉的脸或一个向下的拇指符号常用来指出一个错误；一个闪烁的“正在工作”符号用来指出处理正在进行之中。这种类型的反馈对于熟练用户是很有效的，但对于初学者可能需要更详细的信息，不仅要清楚地指出系统正在做什么，还要指出用户接下来要做什么。

通常应提供足够清晰的反馈信息，从而使其不易被忽略且易于理解，但也不能过分突出反馈信息以至于影响用户的注意力。当按下功能键时，可能给出听得见的点击声作为反馈，也可能醒目显示该功能键作为反馈。听觉反馈的优点是其不使用屏幕空间，用户也不必把注意力集中到工作区内以得到反馈信息。如果反馈信息显示在屏幕上，那么使用固定区域，可以使用户知道在什么地方可查看该信息。有时，将反馈信息安排在光标附近有一定的好处。也可以用不同颜色显示反馈，从而与其他显示对象相区别。

某些类型的输入要求有回显反馈。输入的字符在键入时即回显在屏幕上，使用户可以立即发现并纠正错误。按键和拨号盘输入可以使用相同的方法进行回显。通过拨号盘或显示的标尺而输入的标尺值通常回显在屏幕上，以便检查输入值的精度。对于坐标点的选择操作，可以使用光标或其他符号在所选位置进行回显。如果选择的位置要求精确的回显，那么可以将坐标值显示在屏幕上。

11.9　小结

图形程序的输入可以来自多种不同的硬件设备，多个设备可以提供同一类的输入数据。通过对输入设备进行逻辑分类，图形输入函数可以设计成与特定的输入设备无关。设备按图形输入类型进行分类。ISO 和 ANSI 标准指定的六类逻辑设备是定位设备、笔画设备、字符串设备、定值设备、选择设备和拾取设备。定位设备是任何一种用来输入一个坐标位置的设备。笔画设备输入一串坐标值。字符串设备用来输入文字。定值设备是任何一种用来输入标量值的设备。选择设备用于选择菜单项。拾取设备让用户在一个场景中选取一个对象。设备无关的图形软件包可以用一个辅助库来提供有限的一组输入功能。

一个图形软件包中的输入函数可以定义成三种输入模式。请求模式将输入的控制交给应用程序。取样模式允许输入设备和程序同时工作。事件模式允许输入设备启动数据输入并控制数据的处理。一旦为一个逻辑类和用来输入该类数据的特定物理设备选定了某种模式，那么程序中的输入函数就可以将数据值传递给程序。一个应用程序可以同时使用若干个在不同模式下工作的输入设备。

各种应用，包括设计和绘画软件都使用了一些交互式构图方法。这些方法能让用户定位对象、按预定义的方向和对齐方式约束图形以及交互式地绘制对象。网格、引力场和橡皮条方法用来帮助定位和其他构图操作。

图形用户界面已经成为应用软件的标准特性。一个应用软件的对话从描述该应用软件功能的用户模型开始设计。所有对话元素都以应用语言给出。

窗口系统提供一个典型的子程序接口来操纵窗口、菜单和图符。通用的窗口系统设计成支持多个窗口管理程序。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

用户对话设计的指导思想是易用性、清晰性和灵活性。而且，图形用户界面应设计成能保持用户交互的一致性及支持各种熟练程度的用户。另外，界面应设计成可以尽可能减少对用户的记忆要求，提供足够的反馈，并提供适当的回退和错误处理能力。

GLUT工具包提供了多种交互设备的输入函数，包括鼠标、数据板、空间球、按钮盒和拨号盘。而且，GLUT还支持鼠标和键盘的组合输入。拾取操作的实现需要用到GLU库和OpenGL基本库中的函数。GLUT中提供了一组函数用于显示菜单和子菜单。表11.1和表11.2分别给出了对OpenGL输入函数和菜单函数的总结。

表 11.1 OpenGL 输入函数小结

函数	描述
glutMouseFunc	指定当鼠标按键按下时调用的鼠标回调函数
glutMotionFunc	指定当鼠标光标移动并且鼠标按键按下时调用的鼠标回调函数
glutPassiveMotionFunc	指定当鼠标光标移动并且鼠标按键未按下时调用的鼠标回调函数
glutKeyboardFunc	指定当键盘上的标准键按下时调用的键盘回调函数
glutSpecialFunc	指定当键盘上的特殊功能键按下时调用的键盘回调函数
glutTabletButtonFunc	指定当鼠标光标位于窗口内按下数据板按键时调用的数据板回调函数
glutTabletMotionFunc	指定当鼠标光标位于窗口内移动光标或光笔时调用的数据板回调函数
glutSpaceballButtonFunc	指定当鼠标光标位于窗口内或使用其他窗口激活方法时，按下空间球按键时调用的空间球回调函数
glutSpaceballMotionFunc	指定当空间球在活动窗口内发生平移时调用的空间球回调函数
glutSpaceballRotateFunc	指定当空间球在活动窗口内发生旋转时调用的空间球回调函数
glutButtonBoxFunc	指定当按键按下时调用的按钮盒回调函数
glutDialsFunc	指定当拨号盘旋转时调用的拨号盘回调函数
glSelectBuffer	指定拾取缓冲区的名称的大小
glRenderMode	以GL_SELECT为参数激活拾取模式。此函数也用于激活正常绘制模式或反馈模式
glInitNames	激活对象 ID 名称栈
glPushName	向 ID 栈中压入一个对象的标识符
glLoadName	用指定值置换 ID 栈中的栈顶标识符
glPopName	清除 ID 栈的栈顶元素
gluPickMatrix	为拾取操作定义拾取窗口并且形成修正观察体

表 11.2 OpenGL 菜单函数小结

函数	描述
glutCreateMenu	创建一个弹出式菜单并且指定一个当菜单项被选中时调用的过程；创建的菜单分配到一个整数标识符
glutAddMenuEntry	指定一个显示在菜单上的选项
glutAttachMenu	指定一个用于选择菜单项的鼠标按键
glutSetMenu	指定当前窗口的当前菜单
glutDestroyMenu	清除指定菜单标识符对应的菜单
glutGetMenu	返回当前窗口的当前菜单的标识符
glutAddSubMenu	将一个指定菜单作为子菜单添加到菜单列表中，其中指定菜单已经由 glutCreateMenu 创建
glutDetachMenu	取消一个指定鼠标键与当前菜单的关联
glutRemoveMenuItem	从当前菜单中删除一个指定选项

参考文献

逻辑（或虚拟）输入设备的发展在 Wallace(1976)和 Rosenthal, Michener, Pfaff, Kessener,and Sabin(1982)中讨论。多种输入子程序的实现请参见 Glassner(1990)、Arvo(1991)、Kirk(1992)、

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Heckbert(1994)和 Paeth(1995)。其他使用鼠标和键盘输入的编程示例请参见 Woo, Neider, Davis, and Shreiner(1999)。Shreiner(2000)给出了 OpenGL 基本库和 GLU 库中函数的完整列表。关于 GLUT 输入和菜单函数，请参见 Kilgard(1996)。

用户界面设计指导在 Shneiderman(1986)、Apple(1987)、Bleser(1988)、Brown and Cunningham (1989)、Digital(1989)、OSF/MOTIF(1989)和 Laurel(1990)中给出。关于 X 窗口系统的信息，请参见 Young(1990)和 Cutler, Gilly, and Reilly(1992)。

练习题

11.1 开发一个能使用定位设备在屏幕上定位对象的程序。给出一个几何形状的对象菜单，让用户选择对象及安放位置。该程序应允许对任意多个对象确定位置，直到给出一个结束符号。

11.2 对上述程序加以扩充，从而使对象在定位前可以缩放和旋转。变换选择及变换参数均以菜单项形式给出。

11.3 设计一个程序，该程序允许用户使用一个笔划设备交互式地画图。

11.4 讨论可以在模式识别过程中使用的方法，从而将输入字符与存储的形状库进行匹配。

11.5 设计一个子程序，该子程序可以在屏幕上显示一个线性标尺和一个滑动块，并允许通过沿标尺线定位滑动块来输入数值。选择的数值显示在靠近线性标尺的框内。

11.6 设计一个子程序，显示一个圆形标尺和能在圆上移动以选择角度(以度为单位)的指针，或是一个滑动块。选择的角度值回显在靠近圆形标尺的框内。

11.7 编写一个绘图程序，该程序允许用户通过指定点来绘制线段，并组成图形。每条线段的坐标由定位设备进行选择。

11.8 编写一个通过在指定点间连接直线段而构图的绘图软件。建立一个围绕图中每一条线段的引力场，从而帮助将新的线段与已有线段相连接。

11.9 修改上一练习题中的绘图软件，从而允许对线段进行水平或垂直约束。

11.10 开发可以显示一个任选网格图案的绘图软件，使得选择的屏幕坐标能近似移动到网格交点上。该软件能提供画线功能，并且从一个定位设备取得顶点。

11.11 编写一个可以使用橡皮条方法绘制直线段来构图的子程序。

11.12 编写一个可以使用橡皮条方法构造直线段、矩形和圆的绘图软件。

11.13 编写一个程序，允许用户在一个二维场景中拾取对象。存储每个对象的坐标范围用来根据输入的鼠标光标位置判断拾取的对象。

11.14 编写一个程序，允许用户通过一个基本形状菜单并使用一个拾取设备，将每一个选取的形状拖曳到指定位置来设计图形。

11.15 设计请求模式中输入函数的实现。

11.16 设计取样模式中输入函数的实现。

11.17 设计事件模式中输入函数的实现。

11.18 建立请求、取样和事件模式中输入函数的通用实现。

11.19 扩展 11.6 节的 OpenGL 的画点程序，使其包含一个菜单用于选择点的大小和颜色。

11.20 扩展 11.6 节的 OpenGL 的画折线程序，使其包含一个菜单用于选择线的属性：大小、颜色和线宽。

11.21 修改上一练习题中的程序，允许为折线选择纹理图案。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

11.22 编写一个OpenGL交互程序，在窗口内以输入位置为中心画一个100 × 100像素的矩形。加入一个菜单用于选择填充矩形的颜色。矩形用同色填充。

11.23 修改上一练习题中的程序，使其当无法在窗口内显示整个矩形时拒绝输入位置。

11.24 修改上一练习题中的程序，使其包含一个菜单为矩形选择纹理。至少设置两种纹理图案。

11.25 编写一个OpenGL交互程序，在窗口内的任意位置显示一个输入字符串。输入位置作为字符串的起始位置。

11.26 编写一个OpenGL交互程序，在窗口内的任意位置定位一个二维对象。对象类型通过一个基本形状菜单来选择，至少包括正方形、圆和三角形。

11.27 修改上一练习题中的程序，允许按任意对齐方式显示二维对象。该程序允许从菜单中选择任意多个对象，直到从菜单中选择退出项。

11.28 修改上一练习题中的程序，允许缩放和旋转对象。添加用于几何变换的菜单项。

11.29 编写一个OpenGL交互程序，在窗口内定位一个三维对象。对象的类型从一个GLUT线框实体菜单中选择，并且以输入位置为中心。

11.30 修改上一练习题中的程序，允许以线框方式或实体方式显示对象。在实体显示方式下，在观察点加入一个点光源，并且使用默认的照明和表面阴影参数。

11.31 编写一个程序实现在一个包含多个对象的三维场景中的OpenGL拾取操作。对每一次拾取操作，创建一个小的拾取窗口，并且将拾取窗口内最远的对象调到最前面。

11.32 编写一个OpenGL交互程序显示二维三次Bézier曲线。四个控制点的位置由鼠标输入。

11.33 修改上一练习题中的程序，允许选择Bézier曲线的次数为3、4或5。

11.34 编写一个OpenGL交互程序显示二维三次B样条曲线。样条参数由输入决定，控制点位置由鼠标选定。

11.35 编写一个OpenGL交互程序显示三次Bézier曲面片。控制点的x和y坐标由鼠标输入，z坐标通过输入距地平面的高度指定。

11.36 选择几种读者所熟悉的图形学应用，并建立用户模型，能够作为该领域内图形应用系统用户界面的设计基础。

11.37 请列出能在一个用户界面中提供的帮助设备，并讨论哪一种帮助对不同层次用户是合适的。

11.38 请总结可以处理回退和回显的方法。指出哪一种比较适合于初学者，哪一种比较适合于有经验的用户。

11.39 请列出几种向用户表示菜单的格式，并解释每一种在什么情况下是合适的。

11.40 讨论对于各种层次用户反馈的不同选择。

11.41 列出一个窗口系统在处理具有多个重叠窗口的屏幕布局时，必须完成的功能。

11.42 建立一个窗口管理软件包的设计。

11.43 为着色程序设计一个用户界面。

11.44 为一个二层结构的建模软件设计一个用户界面。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第 12 章　颜色模型和颜色应用

到目前为止，我们对颜色的讨论集中在通过红、绿、蓝三色混合而产生颜色的机制上。这种模型用于在视频监视器上显示颜色。但在图形应用中，还要使用其他一些颜色模型。有些模型用于叙述在打印机和绘图仪上如何输出彩色，有些模型用于传输和存储颜色信息，而其他一些模型则为程序提供更直观的颜色参数接口。

12.1　光的特性

从前面的章节，我们注意到光有很多不同的特性，而且在不同的上下文中我们用不同的方式来描述光的特性。物理学将光描述为一种辐射能量。但在本学科中，我们需要不同的概念来描述人对光的感知。

12.1.1　电磁频谱

在物理学术语中，色光是一个狭窄频段内的电磁辐射。电磁频谱中的其他频段有无线电波、微波、红外线和 X 射线。图 12.1 给出了各种电磁辐射的大致的频率范围。

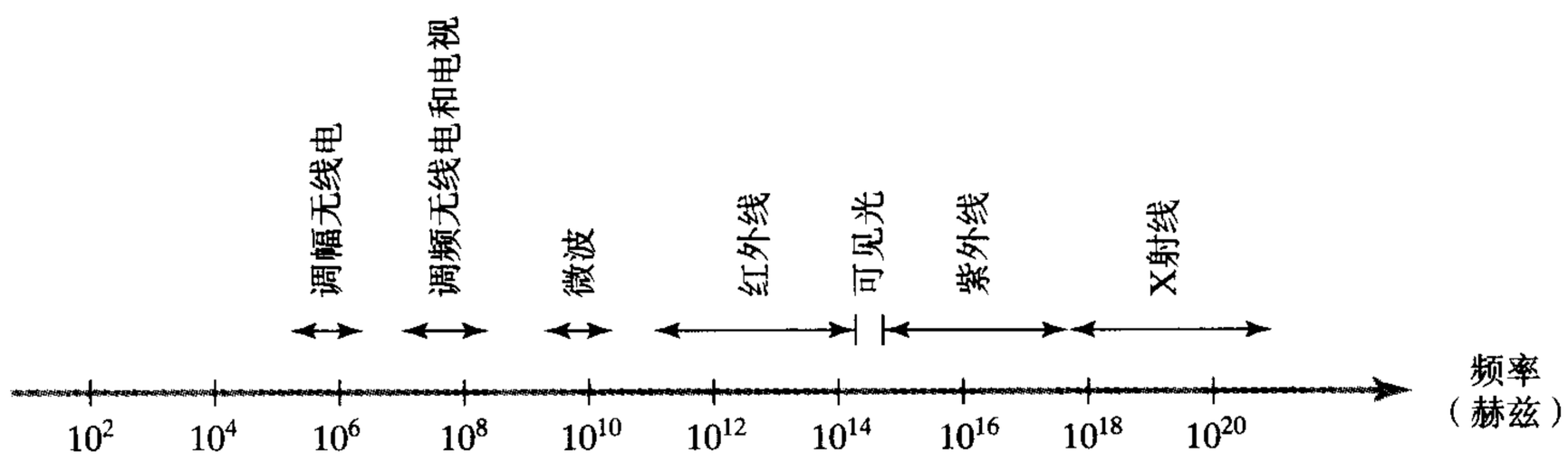


图 12.1　电磁频谱

可见光波段中的每一频率对应一种单独的**光谱颜色**（spectral color）。在低频率端是红色（3.8×10^{14} Hz），高频率端是紫色（7.9×10^{14} Hz）。人眼对红外线和紫外线之间的频率敏感。从低频到高频的光谱颜色变化分别是红、橙、黄、绿、蓝和紫。

在电磁辐射的波动模型中，光被描述为在空间中横向振荡传播的电场和磁场。电场和磁场振荡的方向与传播方向垂直，并且电场与磁场的振荡方向也互相垂直。对于一种颜色的电磁波，我们用频率（f）来表示电磁场强度的振荡率。图 12.2 给出了在一个平面上电场强度随时间变化的振荡。在电磁波上相继的两个具有相同强度的点之间的时间称为波的周期（period）$T = 1/f$。电磁波在单个振荡中传播的距离称为波长（wavelength）λ。单色波的频率和波长互成反比，比例常数是光速 c：

$$c = \lambda f \tag{12.1}$$

频率对于各种物质是一个常数，但光速和波长是依赖于物质的。在真空中，$c = 3 \times 10^{10}$ cm/s。光的波长很短，指定光谱颜色的波长单位通常用埃（1 Å = 10^{-8} cm）或纳米（1 nm = 10^{-7} cm）。纳米等同于毫微米。光谱中红色端的波长大约是 780 nm，而另一端的紫色波长大概是 380 nm。由于波长比频率在某种程度上容易处理，因此常用真空中的波长来指定光谱颜色。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

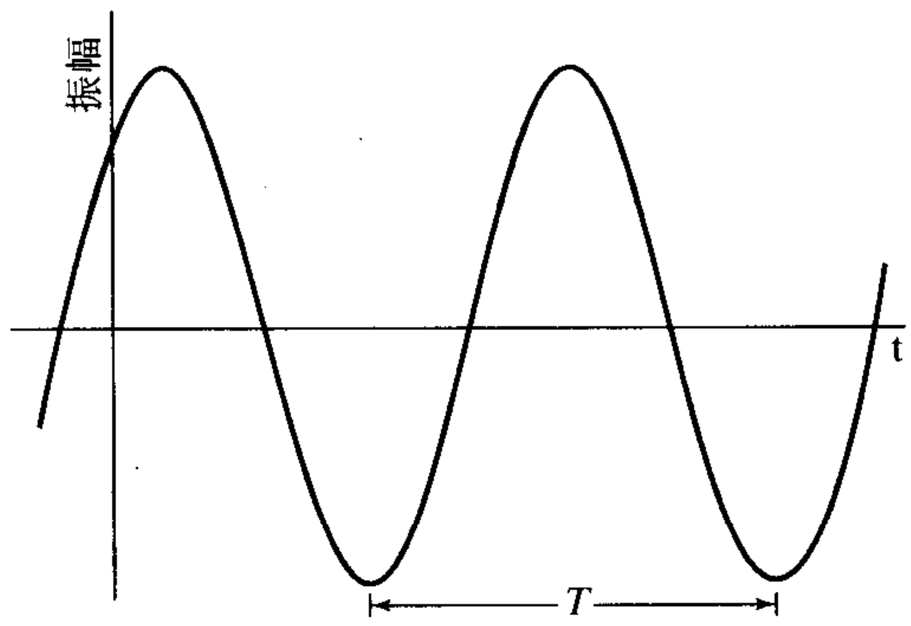


图 12.2 平面振动电磁波的一种电子频率成分的时间变化图。两个连续的波峰或两个连续的波谷之间的时间称为波的周期

太阳或灯泡等光源发射可见波段的全部频率而产生白色光。当白色光投射到一个物体上时，某些频率被反射，某些则被物体吸收了。在反射光中混合的频率确定了我们所感受到的物体的颜色。如果在反射光中以低频率为主，则物体呈现红色。此时，我们可以说光谱中红色端有一个**主频率**（dominant frequency）或**主波长**（dominant wavelength）。主频率也称为**光的色彩**（hue），或者简单地说，就是颜色（color）。

12.1.2 颜色的心理学特征

除了频率以外，描述人对光的感觉还需要其他一些特征。在观察光源时，我们的眼睛对颜色（或主频率）和另外两个基本的感觉做出反应。其中之一是**亮度**（brightness），它对应于全部光能且可量化为光源亮度（参见 10.3 节）。第三个感受的特征是光的**纯度**（purity）或**饱和度**（saturation）。纯度说明光的颜色表现接近光谱色（例如红色）的程度。浅色或暗淡的颜色的纯度（或饱和度）较低，它们比较接近白色。另一术语“**色度**”（chromaticity）通常是指说明纯度和主频率这两种颜色特征。

白色光源发射的能量分布如图 12.3 所示。从红色到紫色的每一频率成分给出不同的能量，光源颜色使用白色来描述。图12.4给出了具有主频率时光源中能量分布的方式。这种光应该是一种纯度较高的红光。现在我们来讨论与主频率对应的颜色的光。图中主频率光成分的能量密度使用E_D表示，白光中其他频率给出的能量密度使用E_W表示。我们可以计算曲线以下的面积来获得光源亮度，即发射的整体能量。纯度依赖于E_D和E_W之间的差别。主频率的E_D与白色光线成分E_W相比越大，则光的纯度越大。当$E_W = 0$时纯度为100%，而当$E_W = E_D$时纯度为0%。

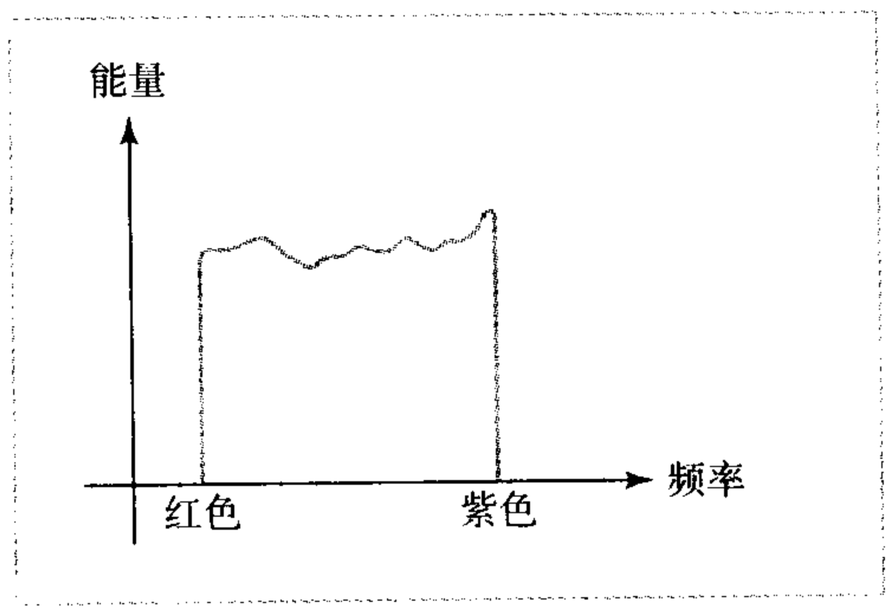


图 12.3 白色光源的能量分布

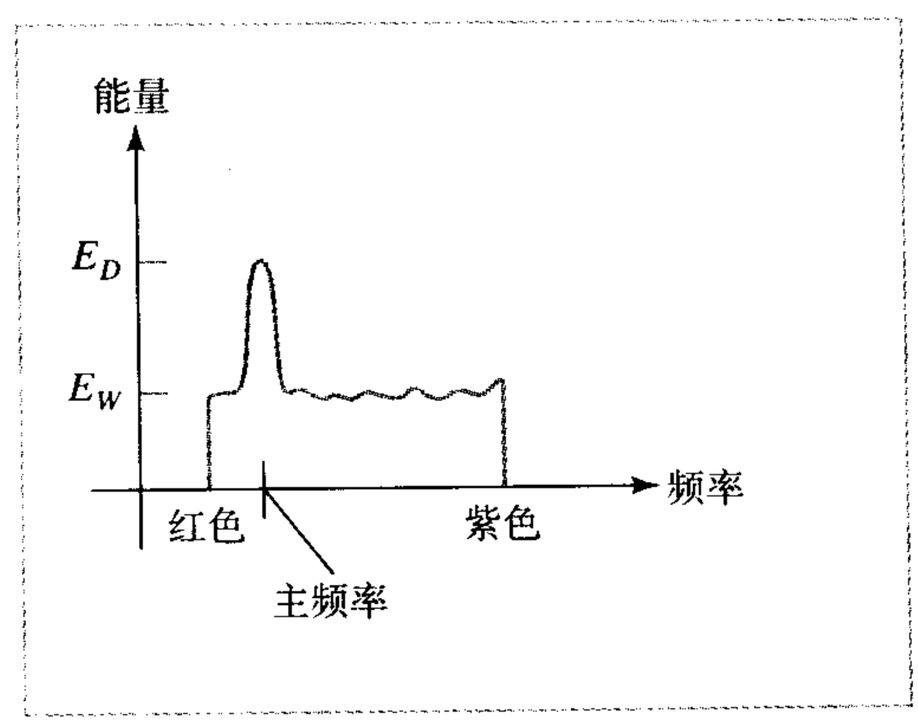


图 12.4 频率范围中，靠近红色端的主频率的光源能量分布

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

12.2　颜色模型

颜色模型（color model）是在某种特定上下文中对于颜色的特性和行为的解释方法。没有哪一种颜色模型能解释所有的颜色问题，因此我们使用不同的模型来帮助说明能看到的不同的颜色特征。

12.2.1　基色

当光由两个或多个具有不同主频率的光源混合而成时，我们可以改变各个光源的强度来生成一系列其他颜色的光。这是一种构造颜色模型的方法，其中用来生成其他颜色的光源的色彩称为**基色**（primary color），通过基色可以产生的所有颜色的集合称其为该颜色模型的**颜色范围**（color gamut）。如果两种基色混合生成白色光，那么就称其为**互补色**（complementary color）。互补色的例子有红色和青色、绿色和品红以及蓝色和黄色。

在实际的基本颜色中，没有哪一组集合能组合生成所有可见的颜色。然而，三种基色对多数应用来说是足够的，而且不包含在指定基色集的颜色范围中的颜色仍然可以使用扩充的方法进行描述。给定三种基色，通过混色处理可以表现任意第四种颜色。将一种或两种基色与第四种颜色混合，产生的颜色可以与余下基色的混合色相匹配。从这一扩充的含义出发，三基色可以用来描述所有颜色。图12.5给出了三基色的颜色匹配函数（color-matching function）和用来生成任何一种光谱颜色的红、绿、蓝的量。图 12.5 给出的曲线经过大量调查综合而得。500 nm 附近的颜色只能与从蓝光和绿光混合所得的光中“减去”红光而得的光相匹配。这就是说，500 nm 左右的颜色与一定量的红光混合可以得到图中指出的蓝－绿混合色。因此，RGB 彩色监视器不能显示 500 nm 左右的颜色。

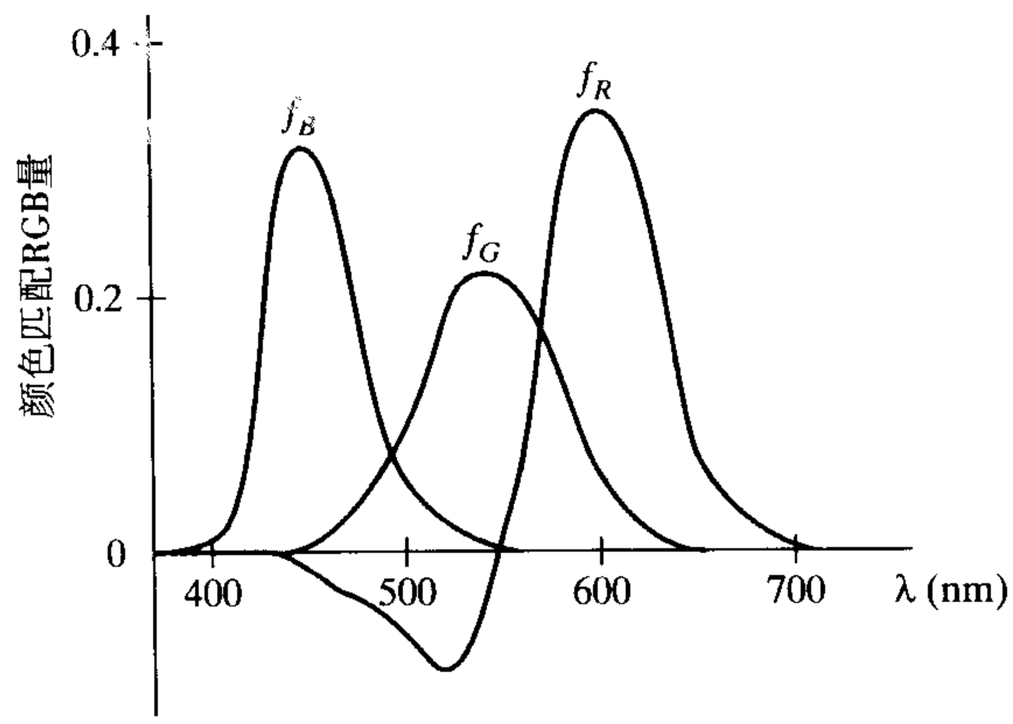


图 12.5　显示光谱颜色（400 nm ~ 700 nm）所需的 RGB 基色量

12.2.2　直观的颜色概念

艺术家在创作彩色画时，使用彩色颜料与黑、白颜料混合以获得各种明暗效果、色泽和色调。艺术家往纯色颜料里添加黑色颜料来生成各种有不同**明暗**（shade）效果的该种色彩。黑色颜料越多，就会更暗。同样，往原色中添加白色颜料可以获得不同的**色泽**（tint），加入较多的白色颜料时，可以使其更亮。**色调**（tone）则通过同时添加黑、白颜料来获得。

对于许多人来说，这些颜色概念比使用三种基色的配合来叙述颜色更直观。通常人们对向纯红色中添加白色得到浅红色、向纯蓝色中添加黑色得到深蓝色比较容易理解。因此，为用户提供颜色板的图形软件包通常使用两种或多种颜色模型。一种为用户提供直观的界面，而其他的则用来为输出设备描述颜色。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

12.3 标准基色和色度图

由于没有哪一组彩色光源可用来组合显示所有可能的颜色，国际照明委员会（CIE）在1931年定义了三种标准基色，这三种基色是从数学上定义的想像的颜色。定义三基色的同时还定义了一组输出全部为正值的颜色匹配函数（参见图12.6），指出描述任何一种光谱颜色所需的每一种基色的量。这便给出了定义各种颜色的国际标准，而且使用CIE基色避免了颜色的负值匹配，以及与选择一组实际基色有关的其他问题。

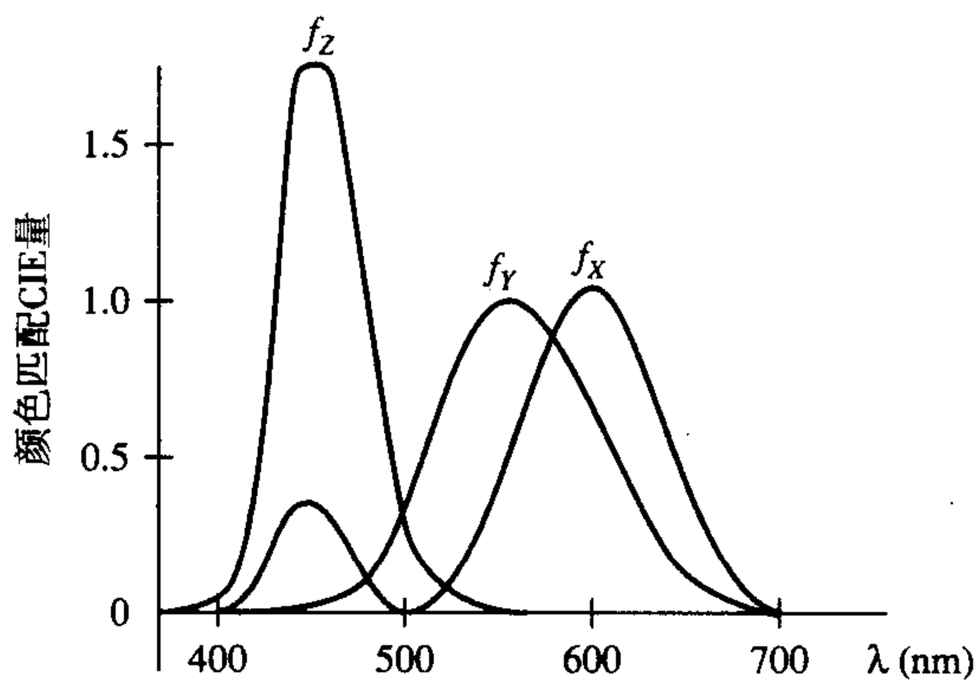


图 12.6 三个 CIE 基色的彩色匹配函数

12.3.1 XYZ 颜色模型

通常把一组CIE基色称为XYZ颜色模型，其中X、Y和Z表示产生一种颜色所需的CIE基色的量。因此，在XYZ模型中描述一种颜色的方式与在RGB模型中是一样的。

在三维XYZ颜色空间中，任何一种颜色$C(\lambda)$可以表示成

$$C(\lambda) = (X,\ Y,\ Z) \tag{12.2}$$

其中X、Y、Z由彩色匹配函数（参见图12.6）计算得来：

$$\begin{aligned} X &= k\int_{\text{visible}\,\lambda} f_X(\lambda)\,I(\lambda)\,d\lambda \\ Y &= k\int_{\text{visible}\,\lambda} f_Y(\lambda)\,I(\lambda)\,d\lambda \\ Z &= k\int_{\text{visible}\,\lambda} f_Z(\lambda)\,I(\lambda)\,d\lambda \end{aligned} \tag{12.3}$$

在以上计算中参数k的取值为683流明/瓦特（lumens/watt），其中流明是光通量单位，等于一个均匀点光源（曾称为烛光）在单位主体角内发出的光通量。函数$I(\lambda)$表示光谱辐射率，即在某一方向上某种光的强度。彩色匹配函数f_Y对应的参数Y是颜色的亮度（参见等式（10.26））。亮度值的范围一般是0到100.0，其中100.0表示白光的亮度。

在XYZ颜色空间中任何颜色都可以表示为三个基色的单位向量**X**、**Y**、**Z**的加性组合。方程（12.2）可以写为

$$C(\lambda) = X\mathbf{X} + Y\mathbf{X} + Z\mathbf{X} \tag{12.4}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

12.3.2　规范化的 XYZ 值

在讨论颜色性质时，可以方便地将等式（12.3）中的量对照光能量总和 $X+Y+Z$ 进行规范化，规范化的量计算如下：

$$x=\frac{X}{X+Y+Z},\qquad y=\frac{Y}{X+Y+Z},\qquad z=\frac{Z}{X+Y+Z} \tag{12.5}$$

这里 $x+y+z=1$。因此，任意颜色可仅用 x 和 y 表示。由于我们对照总能量进行规范化，参数 x 和 y 称为**色度值**（chromaticity value），因为它们仅依赖于色彩和纯度。同样，如果仅用 x 和 y 来指定颜色，也不能获得 X、Y 和 Z 的量。因此，关于一种颜色的完整描述一般要使用三个值 x、y 和 Y。其余两个 CIE 量可进行如下计算：

$$X=\frac{x}{y}Y,\qquad Z=\frac{z}{y}Y \tag{12.6}$$

其中，$z=1-x-y$。我们可以使用色度坐标(x, y)在一个二维图中表示所有颜色。

12.3.3　CIE 色度图

当绘制出可见光谱中颜色的规范化量 x 和 y 时，我们就获得了图 12.7 中舌头形状的曲线。该曲线称为 **CIE 色度图**（CIE chromaticity diagram）。曲线上的点是光谱色（纯色）。连接红色的紫色光谱点的直线称为紫色线（purple line），它并不属于光谱。内部的点表示所有可能的可见颜色的组合。图中 C 点对应于亮白色的位置。实际上，这一点作为**亮白光 C**（illuminant C）而绘出的，该光源作为平均日光的近似标准。

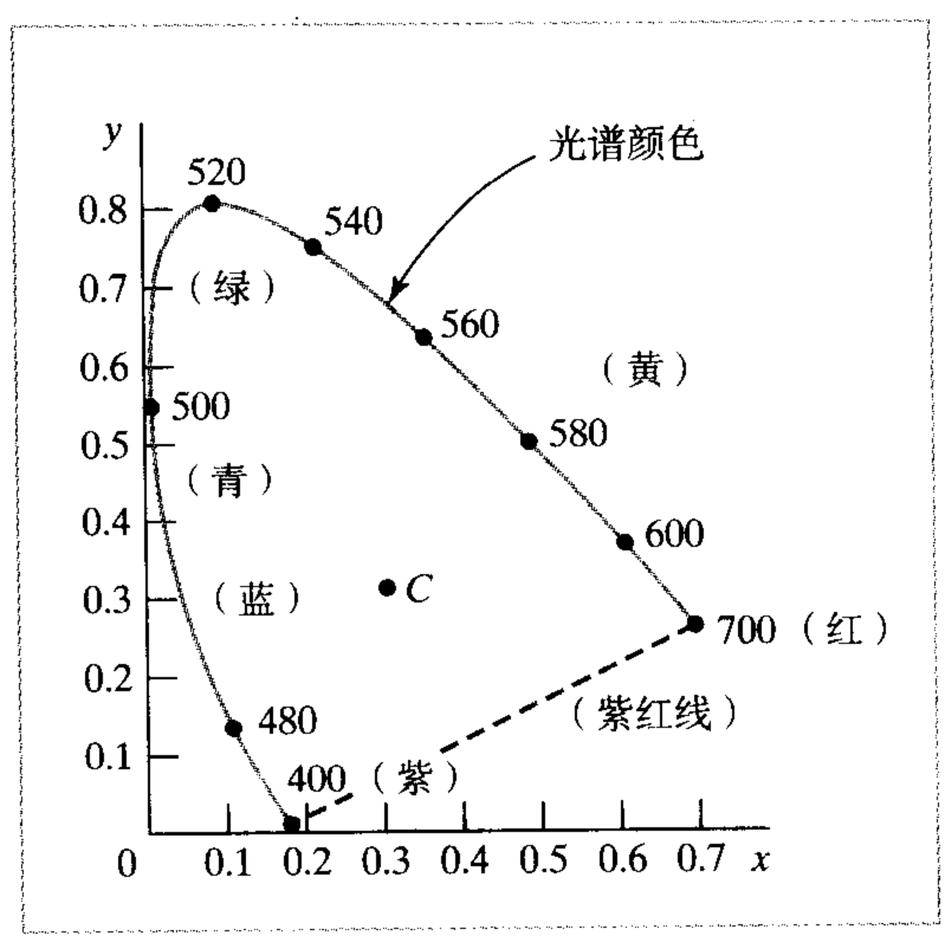


图 12.7　CIE 色度图。光谱颜色从 400 nm 到 700 nm

由于规范化而在色度图中没有给出亮度值。具有同一色度但不同亮度的颜色映射到色度图中的同一点。色度图主要用于：

- 为不同基色组比较整个颜色范围；
- 标识互补颜色；
- 确定指定颜色的主波长和纯度。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

12.3.4 颜色范围

色度图中的颜色范围表示成直线段或多边形。图 12.8 中从 C_1 到 C_2 连线上的所有颜色可以通过混合适量的 C_1 和 C_2 颜色而得到。如果 C_1 占的比例大些，则结果色比较接近 C_1 而离 C_2 较远。三点 C_3、C_4 和 C_5 的颜色范围是这三点连成的三角形。三基色只能产生在三角形内部或边上的颜色。因此，色度图可以帮助我们理解为什么没有哪一个三基色组可以通过加色混合生成所有的颜色，因为图中没有一个三角形能包含所有的颜色。在色度图上很容易对比视频监视器和硬拷贝设备的颜色范围。

12.3.5 互补色

由于两点的颜色范围是一条直线，一对互补色在色度图上对应的两个点一定位于 C 的两边并且它们的连线过 C，如图 12.9 所示。C_1 和 C_2 与 C 的距离决定了产生白色所需的两种颜色的量。

12.3.6 主波长

为了确定一种颜色的主波长，对于图 12.10 中的 C_1，我们可从 C 通过 C_1 画一条直线，并与光谱曲线相交于 C_s。颜色 C_1 就可以表示成白光 C 与光谱颜色 C_s 的混合，因此 C_1 的主波长就是 C_s。这种确定主波长的方法不适用于 C 与紫色线之间的颜色点。在图 12.10 中，画一条从 C 经过 C_2 的直线，我们得到紫色线上的一点 C_p，C_p 并不在可见光谱中。点 C_2 称为非光谱颜色，它的主波长根据位于光谱曲线上的 C_p 的补点（即点 C_{sp}）而获得。非光谱颜色是在紫－品红范围内，具有从白光减去主波长（如 C_{sp}）的光谱分布。

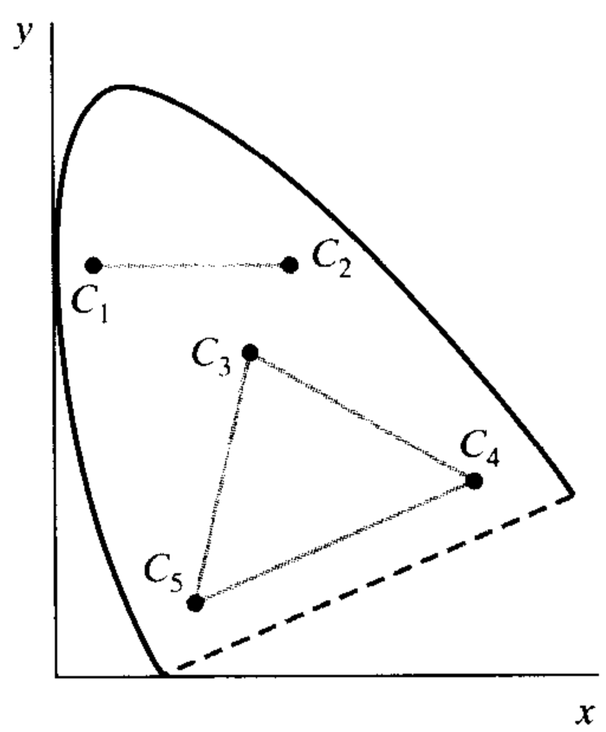


图 12.8 色度图中二基色和三基色系统定义的颜色范围

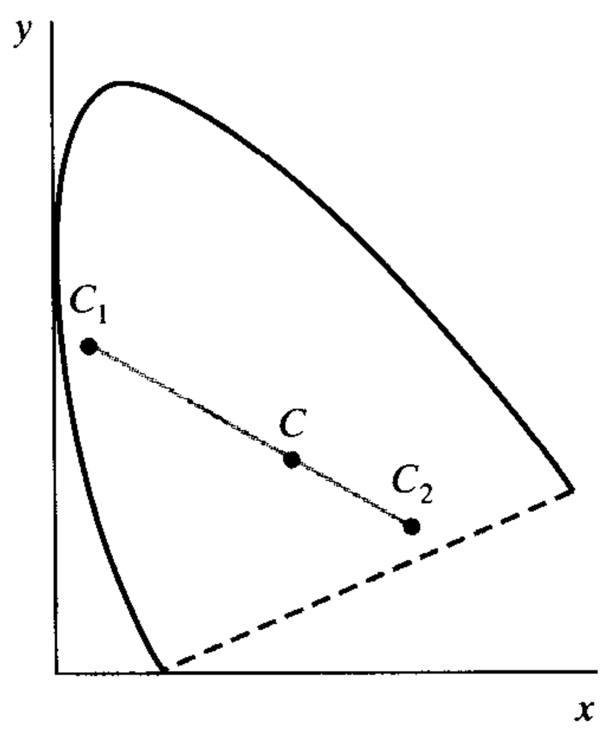


图 12.9 在色度图中表示补色

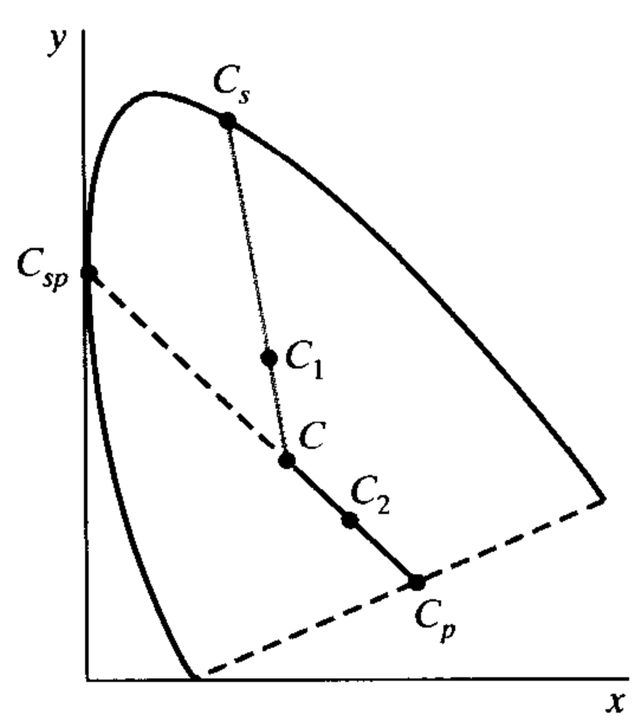


图 12.10 用色度图确定主波长和纯度

12.3.7 纯度

对于许多如图 12.10 中点 C_1 那样的颜色，我们通过沿 C 到 C_s 的直线计算 C_1 到 C 的相对位置来确定纯度。如果 d_{c1} 表示从 C 到 C_1 的距离，且 d_{cs} 表示 C 到 C_s 的距离，我们可以按比率 d_{c1}/d_{cs} 来计算纯度。在此图中，颜色 C_1 的纯度大概是 25%，因为它位于 C 到 C_s 全程的大约四分之一处。在 C_s 处颜色点的纯度为 100%。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

12.4　RGB 颜色模型

基于三刺激理论（tristimulus theory），我们的眼睛通过光对视网膜的锥状细胞中的三种视色素的刺激来感受颜色。这三种色素分别对波长为 630 nm（红色）、530 nm（绿色）和 450 nm（蓝色）的光最敏感。通过对光源中的强度比较，我们感受到光的颜色。这种视觉理论是使用三种颜色基色：红、绿和蓝在视频监视器上显示彩色的基础，称为 RGB 颜色模型。

我们可以使用图 12.11 所示，由 R、G 和 B 坐标轴定义的单位立方体来描述这个模型。坐标原点代表黑色，而其对角坐标点(1, 1, 1)代表白色。在三个坐标轴上的顶点代表三个基色，而余下的顶点则代表每一个基色的补色。

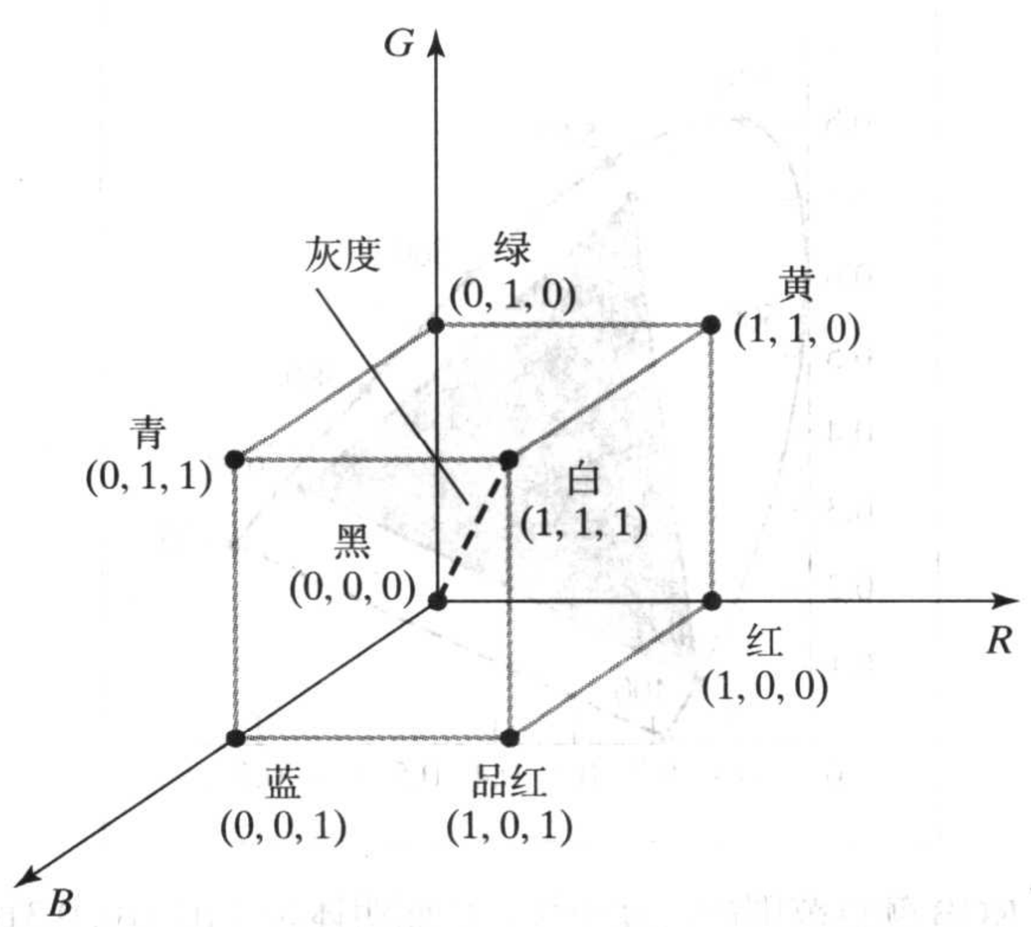
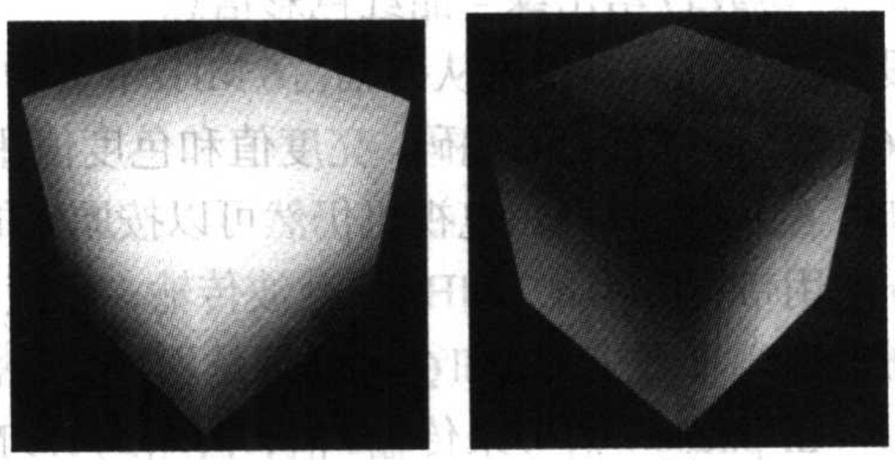


图 12.11　RGB 颜色模型，在立方体内的颜色用三基色的加性组合来描述

和 XYZ 颜色系统一样，RGB 颜色框架是一个加色模型。多种基色的强度加在一起生成另一种颜色。立方体边界中的每一个颜色点都可以表示三基色的加权向量和，用单位向量 **R**、**G** 和 **B** 表示如下：

$$C(\lambda) = (R,\ G,\ B) = R\,\mathbf{R} + G\,\mathbf{G} + B\,\mathbf{B} \tag{12.7}$$

其中 R、G 和 B 的值在 0 到 1.0 的范围内赋值。例如，顶点的品红通过将红色和蓝色相加生成三元组(1, 0, 1)获得，而白色(1, 1, 1)则是红色、蓝色和绿色顶点的和。灰度则通过立方体的原点到白色顶点的主对角线上的位置进行表示。对角线上每一点是等量的每一种基色的混合。因此，从黑色到白色之间中等明暗的灰色表示成(0.5, 0.5, 0.5)。图 12.12 给出了沿着前视面和顶视面逐渐变化的颜色。



(a)　(b)

图 12.12　RGB 颜色立方体的视图：(a)沿从白到黑的灰度对角线；(b)从黑到白的灰度对角线

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

表 12.1 列出了 NTSC（National Television System Committee）标准 RGB 磷粉的色度坐标。表中还列出了在 CIE 颜色模型中的 RGB 色度坐标，以及彩色监视器使用的磷粉近似值。图 12.13 给出了 NTSC 标准中 RGB 基色的近似颜色范围。

表 12.1 RGB(*x*, *y*)色度坐标

	NTSC 标准	CIE 模型	彩色监视器近似值
R	(0.670，0.330)	(0.735，0.265)	(0.628，0.346)
G	(0.210，0.710)	(0.274，0.717)	(0.268，0.588)
B	(0.140，0.080)	(0.167，0.009)	(0.150，0.070)

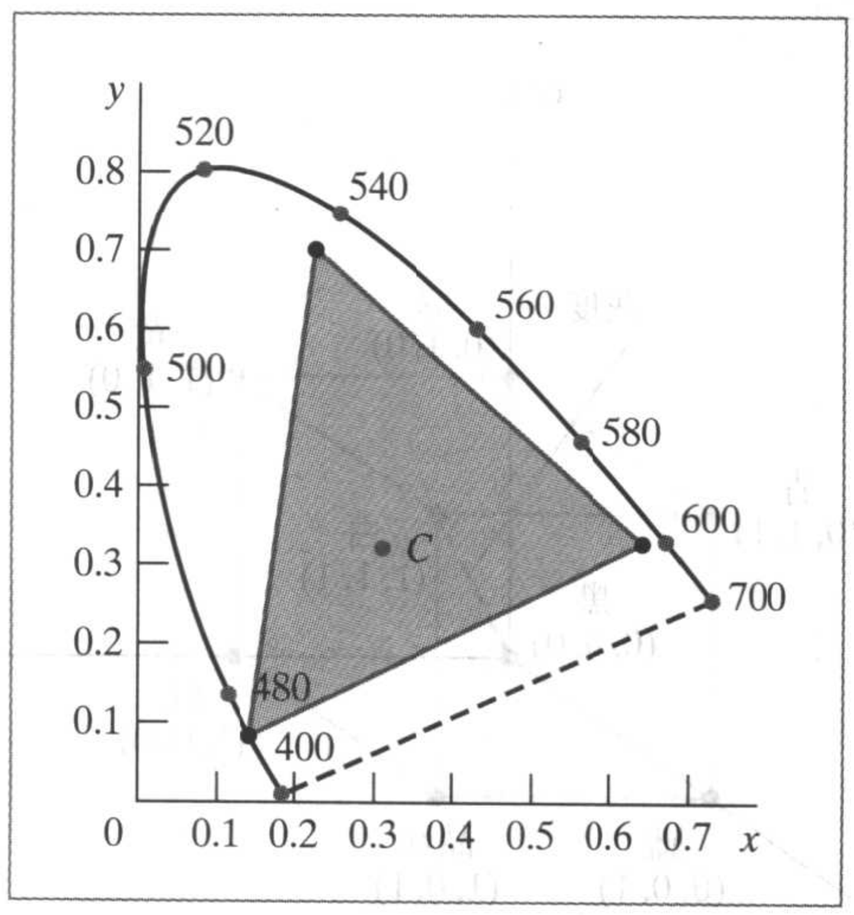


图 12.13 NTSC 标准中 RGB 颜色范围的色度坐标。C 照明体位于(0.310, 0.316)，其亮度值 $Y = 100.0$

12.5 YIQ 颜色模型

RGB 监视器要求一张图像由分开的红色、蓝色和绿色信号组成，而电视监视器则使用组合信号。形成组合视频信号的 NTSC 颜色模型是 YIQ 模型。

12.5.1 YIQ 参数

YIQ 颜色模型中的参数 Y 与 XYZ 模型中 Y 的相同。亮度（明度）信息包含在 Y 参数中，而色度信息（色彩和纯度）则结合在 I 和 Q 参数中。为 Y 参数选择的红色、蓝色和绿色的组合给出了标准的亮度曲线。由于 Y 包含了亮度信息，所以黑白电视机只使用 Y 信号。参数 I 包含有橙－青色彩信息，提供鲜艳色彩的明暗度。参数 Q 给出绿－品红色彩信息。

NTSC组合颜色信号的设计允许黑白电视机从一幅占6 MHz带宽的图像信息中提取所需的灰度信息。因此，*YIQ*信息必须在6 MHz带宽限制下编码。亮度值和色度值用不同的模拟信号进行编码，这样只是在原来的带宽内增加了颜色信息，黑白电视机仍然可以按原来的方式取得原样的亮度信号。亮度信息（Y 值）以调幅的方式用带宽约为 4.2 MHz 的载波传输；色度信息（I、Q 值）被结合在一起用带宽约为 1.8 MHz 的载波传输。参数名称 I 和 Q 指的就是用来在载波上编码颜色信息的调制方法。一种调幅编码方法用同步（in-phase）信号来传输 I 值，占用约 1.3 MHz 的带宽。另外一种相位调制编码方法（“正交”信号）占用约 0.5 MHz 的带宽来传输 Q 值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

在 NTSC 信号中，亮度信息（4.2 MHz 带宽）的编码精度高于色度信息（1.8 MHz 带宽）。这是因为人眼对亮度的变化比对色度的变化更敏感。因而，NTSC用较低的精度传输色度信息并没有造成图像颜色质量的明显降低。

我们可以使用等式（10.27）来计算 RGB 颜色的亮度值。一种产生色度值的方法是从颜色的红色和蓝色分量上减去亮度值。如下式：

$$\begin{aligned} Y &= 0.299\,R + 0.587\,G + 0.114\,B \\ I &= R - Y \\ Q &= B - Y \end{aligned} \tag{12.8}$$

12.5.2　RGB 颜色空间和 YIQ 颜色空间之间的转换

NTSC编码器可以用等式（12.8）将一个RGB信号转换为YIQ值，然后调制载波信号。从RGB空间到到 YIQ 空间的转换可以用下面的变换矩阵实现：

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.701 & -0.587 & -0.114 \\ -0.299 & -0.587 & 0.886 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{12.9}$$

相反地，NTSC 解码器将一个 NTSC 视频信号转换为 RGB 颜色值。该变换建立在 NTSC 标准的 RGB 磷粉基础上，其色度坐标由前一节给出。该解码器将视频信号分解成 YIQ 分量，然后转换成 RGB 值。我们使用等式（12.9）的逆矩阵将 YIQ 空间转换成 RGB 空间：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 1.000 & 0.000 \\ 1.000 & -0.509 & -0.194 \\ 1.000 & 0.000 & 1.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \tag{12.10}$$

12.5.3　YUV 和 YC_rC_b 系统

因为在NTSC的组合模拟视频信号中分配给色度信息较低的带宽，NTSC图像的颜色质量有些受到影响。因此，已提出多种 YIQ 编码的变体来提高视频传输的颜色质量。一种变体就是 YUV 颜色模型，它为 PAL（Phase Alternation Line）广播制式提供视频传输的组合颜色信息。PAL 制式在欧洲、非洲、澳洲和欧亚大陆被广泛采用。另外一种数字编码的 YIQ 的变体是 YC_rC_b，它是为数字视频转换设计的颜色表示模型。这种方式也被多种图形文件格式所采用，例如JPEG（参见15.4节）。

12.6　CMY 和 CMYK 颜色模型

视频监视器通过组合屏幕磷粉发射的光而生成颜色，这是一种加色处理。而打印机、绘图仪之类的硬拷贝设备通过往纸上涂颜料来生成彩色图片。我们通过反射光而看见颜色，这是一种减色处理。

12.6.1　CMY 参数

CMY 颜色模型使用青色、品红和黄色作为三基色。我们已经指出，青色可由绿色光和蓝色光相加而得。因此，当白色光从青色墨水中反射出来时，反射光中一定没有红色成分。即红色被墨水吸收了或减掉了。同样，品红墨水减掉投射光中的绿色成分，而黄色墨水减去光中的蓝色成分。图 12.14 表示 CMY 模型的单位立方体。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

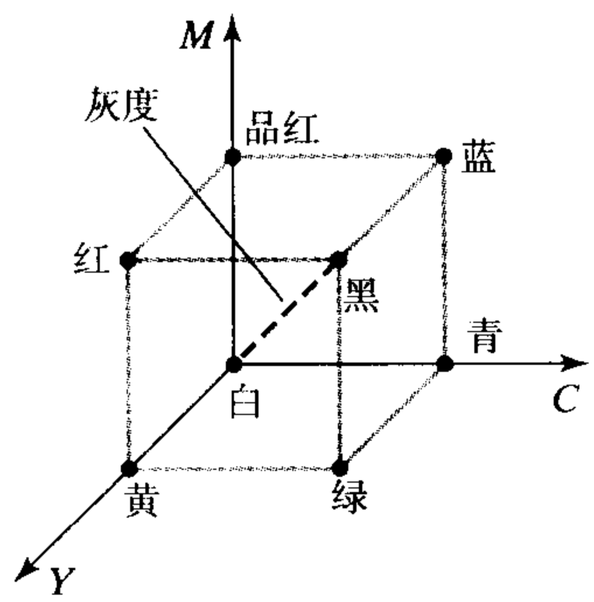


图 12.14 使用单位立方体内减色处理定义颜色的 CMY 颜色模型

在 CMY 模型中，点(1,1,1)因为减掉了所有的投射光成分而表示黑色，原点表示白色。沿着立方体对角线，每种基色量均相等而生成灰色。青色和品红墨水的混合生成蓝色，因为投射光的红色和绿色成分都被吸收了。类似地，青色和黄色墨水的混合产生绿色，品红和黄色的混合产生红色。

使用CMY模式的打印处理通过四个墨点的集合来产生颜色点，在某种程度上与RGB监视器使用三个磷粉点的集合是一样的。因此，在实际使用中，CMY 颜色模型也称为CMYK 颜色模型，其中 K 是黑色参数。三种基色（青、品红和黄）各使用一点，黑色也使用一点。因为青色、品红色和黄色墨水的混合通常生成深灰色而不是黑色，所以黑色单独包含在其中。有些绘图仪通过重叠喷上三种基色的墨水并让它们在干之前混合起来而生成各种颜色。对于黑白或灰度图像，只用黑色墨水就可以了。

12.6.2 CMY 颜色空间和 RGB 颜色空间之间的转换

我们可以使用一个变换矩阵来表示从 RGB 到 CMY 的转换。

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{12.11}$$

这里单位列向量表示RGB系统中的白色。同样，我们也可使用一个变换矩阵把CMY颜色表示转换成 RGB：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix} \tag{12.12}$$

这里，单位列向量表示 CMY 系统中的黑色。

从 RGB 空间向 CMYK 空间转换时，首先设 $K = \max(R, G, B)$，然后从方程（12.11）中计算出的 C、M、Y 都要减去 K。类似地，从 CMYK 空间向 RGB 空间转换时，首先设 $K = \min(R, G, B)$，然后从方程（12.12）中计算出的 R、G、B 都要减去 K。在实际使用中，这两个变换方程常常被修改以提高特定系统中的打印质量。

12.7 HSV 颜色模型

除了一组基色的表示方法，HSV 模型使用对用户更直观的颜色描述方法。为了给出一种颜色描述，用户需要选择一种光谱色并加入一定量白色和黑色来获得不同的明暗、色泽和色调（参见12.2节）。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

12.7.1　HSV 参数

这个模型中的颜色参数的是色彩（H）、色饱和度（S）和明度值（V）。HSV 模型的三维表示从RGB立方体演变而来。如果我们沿对角线从白色顶点向黑色顶点观察，可以看到如图12.15所示的立方体的六边形外形。六边形的边界表示不同的色彩，用于HSV六棱锥（参见图 12.16）的顶平面。在六棱锥中，色饱和度沿水平轴测量，而明度值沿通过六棱锥中心的垂直轴进行测量。

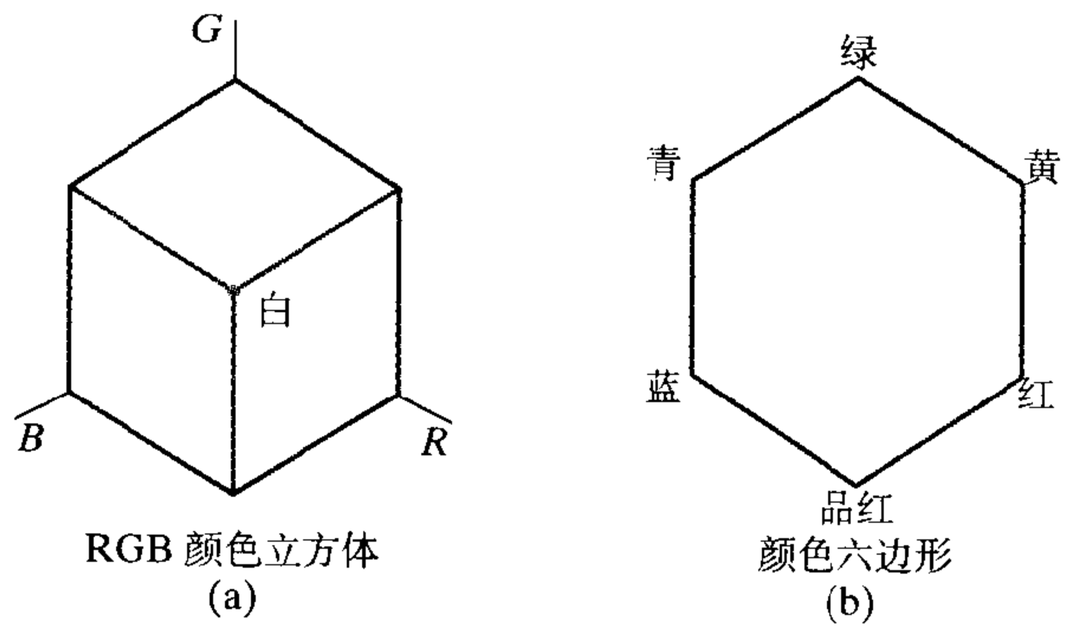


图 12.15　RGB 颜色立方体的视图：(a)沿从白色到黑色的对角线观察；(b)颜色立方体的外轮廓是一个六边形

色彩则使用与水平轴之间的角度来表示，范围从0°到360°。六边形的顶点以60°为间隔。黄色位于60°处，绿色在120°处，而青色在180°处，与红色相对。互补的颜色相距180°。

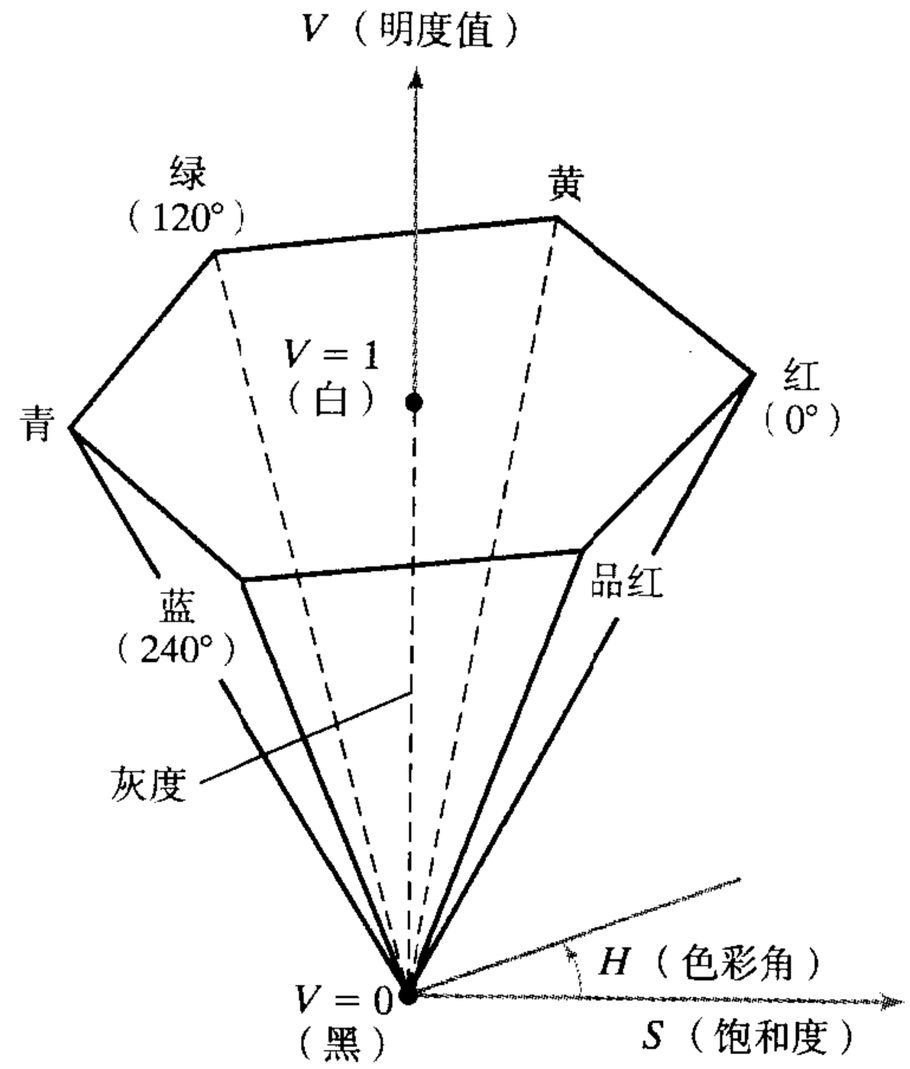


图 12.16　HSV 六棱锥

色饱和度参数 S 指明颜色的纯度，从 0 到 1.0 变化。纯色（光谱色）的 $S = 1.0$，越靠近六棱锥中心的灰度线，S 越小。灰度颜色的 $S = 0$。

明度值 V 从六边形顶点的 0 变化到顶平面的 1.0，顶点表示黑色。在六边形顶平面的颜色强度最大。纯色的 $V = 1.0$ 并且 $S = 1.0$。白色的 $V = 1.0$ 并且 $S = 0$。

HSV 对于多数用户是一个更加简单、直观的模型。从指定一种纯色彩开始，即指定色彩角 H 且让 $V = S = 1.0$，可以通过将白色或黑色加入到纯色彩中来描述所要的颜色。添加黑色则减小 V 而

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

S 保持不变。要得到深蓝色，使得 $V = 0.4$、$S = 1.0$ 且 $H = 240°$。同样，将白色加进所选的色彩中时，参数 S 减小而 V 保持不变。浅蓝色可用 $S = 0.3$、$V = 1.0$ 且 $H = 240°$ 来设定。添加一些黑色和白色，则同时减小 V 和 S。这种模型的界面通常在一个具有滑动块和颜色轮的调色板中给出 HSV 参数选择。

12.7.2 选择明暗、色泽和色调

结合明暗、色泽和色调等术语的颜色概念反映在 HSV 六棱锥的剖面中（参见图 12.17）。往纯色彩中添加黑色时把 V 减小到六棱锥的下方。因此，各种明暗用 $S = 1.0$ 且 $0.0 \leqslant V \leqslant 1.0$ 来表示。向纯色调中添加白色生成六棱锥顶平面的各种色泽，该平面上 $V = 1.0$ 且 $0 \leqslant S \leqslant 1.0$。同时添加白色和黑色可以指定各种色调，从而生成六棱锥体三角形剖切面范围内的颜色点。

人眼大概能区分 128 种不同色彩和 130 种不同色泽（色饱和度级）。依赖于所选的色彩，还可以进一步区分若干种明暗。对于黄色能分辨出 23 种明暗度，对于光谱的蓝色端能分辨 16 种。这就是说，我们能分辨出大约 128 × 130 × 23 = 382 720 种不同的颜色。对于多数图形应用，128 种色彩、8 种色饱和度级别以及 16 种明度值就足够了。按 HSV 的这一参数范围，用户可使用 16 384 种颜色，系统需使用 14 位来存放每一像素的颜色。使用颜色查找表可减少存储器的需求及增加可用颜色的数量。

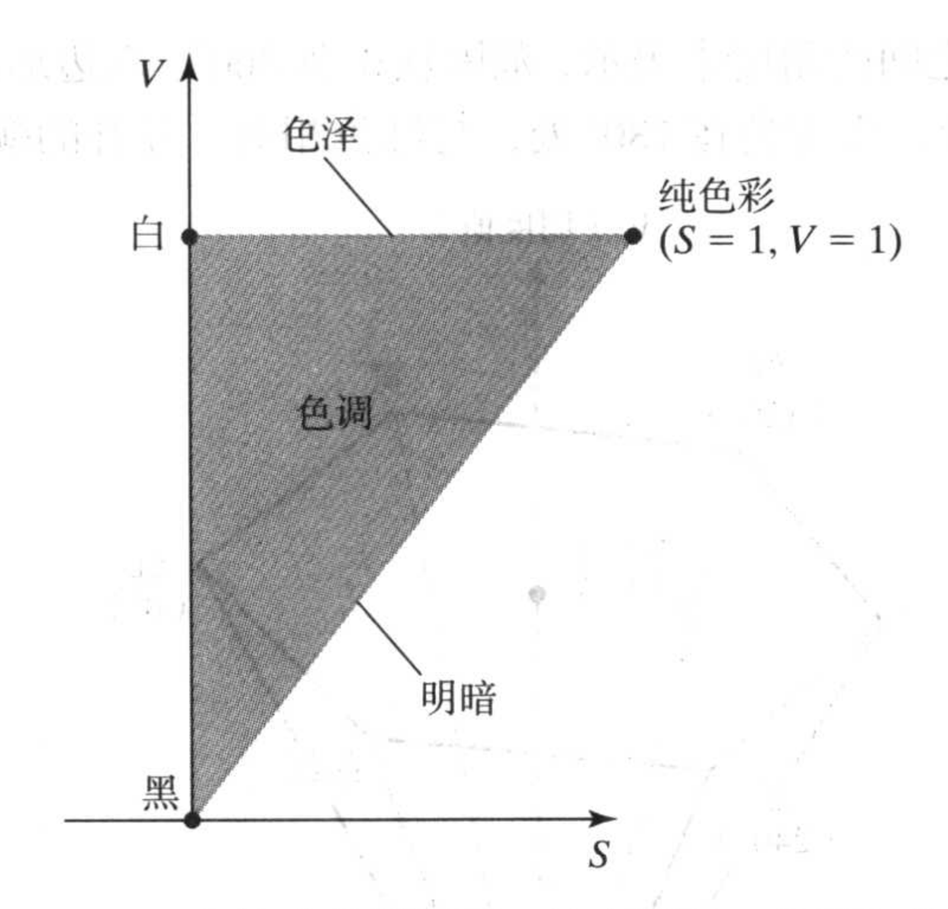


图 12.17 HSV 六棱锥的剖切面，给出明暗、色泽和色调区域

12.7.3 HSV 和 RGB 模型之间的转换

为了确定该转换所需的操作，我们先考察如何从 RGB 立方体演变为 HSV 六棱锥。RGB 立方体从黑色（原点）到白色的立方体对角线与六棱锥的 V 轴相对应。同样，RGB 立方体的每一个子立方体与六棱锥的六边形剖面区域相对应。在任意剖面中，六边形的各边和从 V 轴到任意顶点的射线都具有明度值 V。对于任何一组 RGB 值，V 与其中的最大值相等。与一组 RGB 值对应的 HSV 点位于明度值为 V 的六边形剖面上。参数 S 按照该点到 V 轴的相对距离而确定。参数 H 通过计算该点在六边形的六等分中的相对位置来确定。下列过程给出了将任意一组 RGB 值变换到对应的 HSV 值的算法。

```
class rgbSpace {public: float r, g, b;};
class hsvSpace {public: float h, s, v;};
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
const float noHue = -1.0;
inline float min(float a, float b) {return (a < b)? a : b;}
inline float max(float a, float b) {return (a > b)? a : b;}

void rgbTOhsv (rgbSpace& rgb, hsvSpace& hsv)
{
   /* RGB and HSV values are in the range from 0 to 1.0 */
   float minRGB = min (r, min (g, b)), maxRGB = max (r, max (g, b));
   float deltaRGB = maxRGB - minRGB;

   v = maxRGB;
   if (maxRGB != 0.0)
      s = deltaRGB / maxRGB;
   else
      s = 0.0;
   if (s <= 0.0)
      h = noHue;
   else {
      if (r == maxRGB)
         h = (g - b) / deltaRGB;
      else
         if (g == maxRGB)
            h = 2.0 + (b - r) / deltaRGB;
         else
            if (b == maxRGB)
               h = 4.0 + (r - g) / deltaRGB;
      h *= 60.0;
      if (h < 0.0)
         h += 360.0;
      h /= 360.0;
   }
}
```

我们通过反求 rgbTOhsv 过程中的方程来获得从 HSV 到 RGB 参数的变换。将为六棱锥的每一个六等分部分而实现这些逆操作。最终的变换方程归入下列算法中。

```
class rgbSpace {public: float r, g, b;};
class hsvSpace {public: float h, s, v;};

void hsvTOrgb (hsvSpace& hsv, rgbSpace& rgb)
{
   /* HSV and RGB values are in the range from 0 to 1.0 */
   int k
   float aa, bb, cc, f;

   if ( s <= 0.0)
      r = g = b = v;      //Have gray scale if s = 0.
   else {
      if (h == 1.0)
         h = 0.0;
      h *= 6.0;
      k = floor (h);
      f = h - k;
      aa = v * (1.0 - s);
      bb = v * (1.0 - (s * f));
      cc = v * (1.0 - (s * (1.0 - f)));
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
        switch (k)
        {
            case 0: r = v; g = cc; b = aa; break;
            case 1: r = bb; g = v; b = aa; break;
            case 2: r = aa; g = v; b = cc; break;
            case 3: r = aa; g = bb; b = v; break;
            case 4: r = cc; g = aa; b = v; break;
            case 5: r = v; g = aa; b = bb; break;
        }
    }
}
```

12.8 HLS 颜色模型

另一个基于直观颜色参数的模型是 Tektronix 公司使用的 HLS 系统。该模型表示为图 12.18 中所示的双棱锥体。该模型中的三个参数称为色彩（H）、亮度（L）和色饱和度（S）。

色彩含义与 HSV 模型中相同，它指明所选色彩的位置与水平轴之间的夹角。在此模型中，$H = 0°$ 与蓝色相对应。其余颜色按与 HSV 模型中的同样顺序围绕锥体逐一指定。品红在 60°、红色在 120° 而青色位于 $H = 180°$。互补色也是在双棱锥上互成 180°。

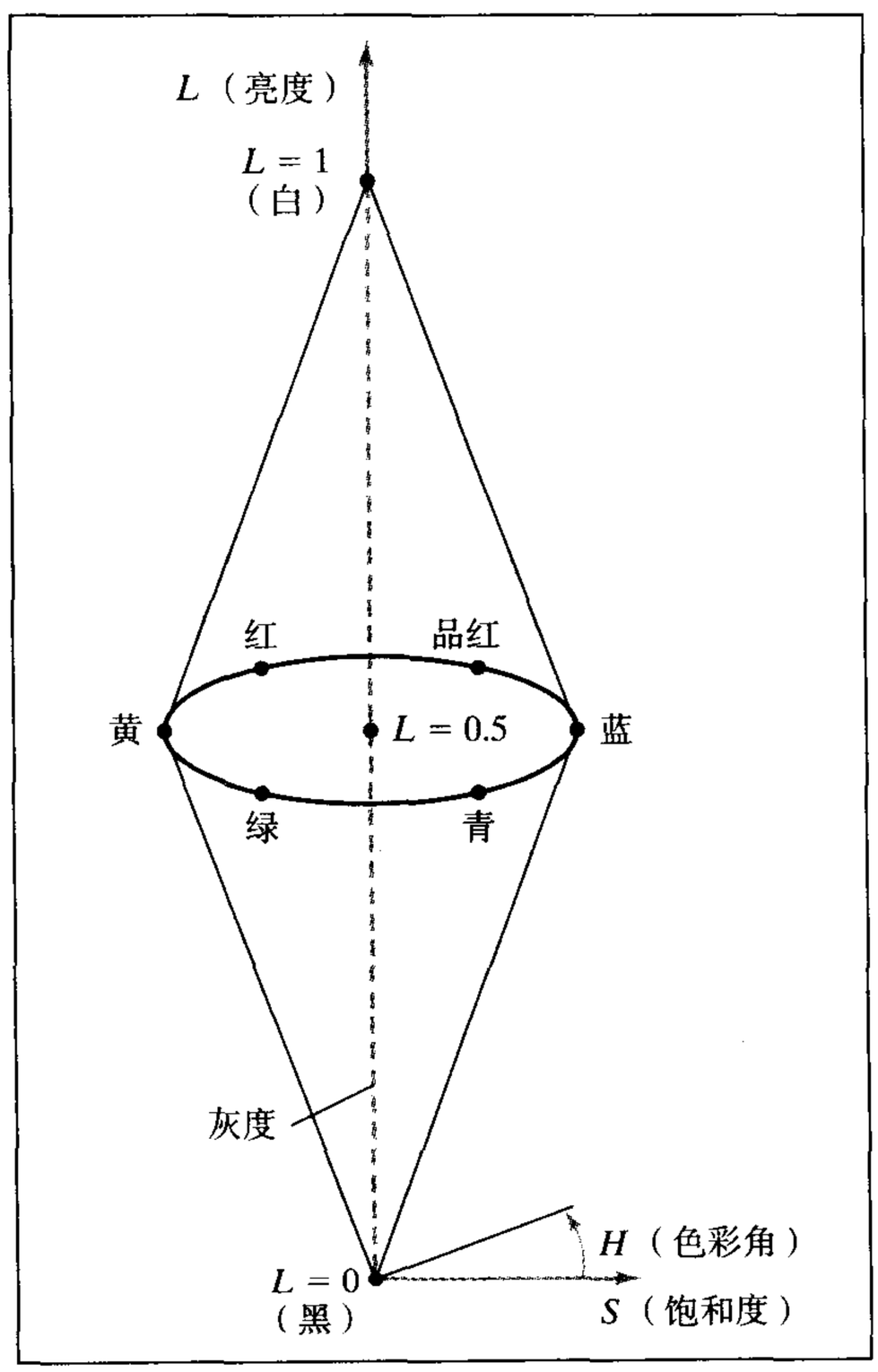


图 12.18 HLS 双锥体

该模型的垂直轴称为亮度 L。在 $L = 0$ 处为黑色，$L = 1.0$ 处为白色。灰度则沿着 L 轴分布，且“纯色彩”位于 $L = 0.5$ 的平面上。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

色饱和度参数 S 也是说明颜色的相对纯度。该参数变化范围为 0 到 1.0，对于纯色彩，$S = 1.0$ 且 $L = 0.5$。当 S 减少时，白色增加，色彩的纯度就减少。当 $S = 0$ 时，仅有灰度。

指定一种颜色的过程首先通过色彩角H进行选择，而所需的明暗、色泽及色调则通过调节 L 和 S 来获得。增加 L 使颜色更亮些，减少 L 则使其更暗些。当 S 减少时，颜色向灰色变化。

12.9　颜色选择及其应用

一个图形软件包可以提供帮助我们选择颜色的各种功能。各种组合颜色可以使用滑动块和颜色轮进行选择，而不是要求直接输入RGB的分量值。系统还可以设计成能帮助选择柔和色。此外，软件包的设计者在设计面向用户的颜色显示时可以遵循某些颜色规则。

获得一组坐标颜色的一种方法是从颜色模型的某一子空间中产生。如果颜色是从沿 RGB 或 CMY 立方体中任意直线段上的规则间隔中选择，那么我们可以得到一组匹配较好的颜色。随机选取的色彩可能会导致刺眼和不柔和的颜色组合。选择颜色组合的另一种考虑是不同颜色在不同深度上的感觉。这是因为我们的眼睛是按频率而注意到颜色的。蓝色特别有助于放松眼睛。在红色图案附近显示蓝色图案会引起眼睛的疲劳。因为在把注意力从一个区域转向另一区域时要不断地重新聚焦。分开这些颜色或使用HSV模型中的一半或更少的颜色，可以减少上述问题。按照这种技术，一次显示中或者包含蓝色和绿色，或者包含红色和黄色。

作为一种规则，使用较少的颜色比用较多的颜色能产生更令人满意的显示，而淡色和暗色的混合比纯色彩更柔和。对于背景，最好使用灰色或前景色的补色。

12.10　小结

光可以描述为在空间中传播的具有一定能量分布的电磁辐射。色光分布在电磁频谱的狭窄的频率波段中。然而，光还有许多其他的性质，我们用各种参数来表示光的不同方面的性质。根据光的波粒二象性原理，我们可以解释可见光的物理特性。我们把对光的感觉量化为主频率（色彩）、亮度（明度）和纯度（色饱和度）。色彩和色饱和度涵盖了光的色度性质。

本章介绍了颜色模型的概念来解释混合光源的效果。一种定义颜色模型的方法是，指定两个或多个用来生成各种其他颜色的基色集。然而，不存在一种模型能通过一组有限的基色来描述所有可能的颜色。通过基色可以产生的所有颜色的集合称为该颜色模型的颜色范围。互补色是混合后产生白色的两种颜色。

1931 年，CIE 采纳了三个假定的颜色（称为CIE 基色）及其彩色匹配函数作为定义所有颜色组合的标准。这一组 CIE 基色通常称为 XYZ 颜色模型，其中 X、Y 和 Z 表示生成电磁光谱中任意一种颜色所需的各种基色的量。彩色匹配函数的定义为非负的，并且 Y 表示亮度值。规范化的 X 和 Y 值（称为 x 和 y）用来绘出在CIE 色度图中所有光谱色的位置。我们可以使用该色度图来比较不同颜色模型的颜色范围，找出给定颜色的补色及确定给定颜色的主频率和纯度。

利用三基色定义的常用颜色模型有 RGB、YIQ 和 CMY 模型。视频监视器显示使用RGB 模型。电视广播中的组合视频信号使用 YIQ 模型。硬拷贝设备使用 CMY 模型输出彩色。

用户界面通常提供像HSV 和HLS 这样的直观的颜色模型用于选择颜色。这些模型允许用户通过指定色彩的值以及添加白色和黑色的量来指定颜色。往纯色彩中添加黑色产生暗色，添加白色生成色泽，同时添加白色和黑色可以指定各种色调。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

产生彩色显示的很重要一点是选择调和的颜色组合。为避免刺目的颜色组合，我们按照某些简单的规则，通常可以在颜色模型的一个子空间中选择坐标颜色。我们还应避免显示与主频率距离较远的相邻颜色。此外，还应将颜色限制在用浅色和暗色而不是用纯色组成的较少量的颜色组合中。

参考文献

关于颜色科学的综合性讨论请参见Wyszecki和Stiles(1982)。颜色模型和颜色显示技术在Smith(1978)、Heckbert(1982)、Durrett(1987)、Schwartz, Cowan, and Beatty(1987)、Hall(1989)和Travis(1991)中讨论。

各种颜色应用的算法参见Glassner(1990)、Arvo(1991)、Kirk(1992)、Heckbert(1994)和Paeth(1995)。关于人的视觉系统和我们对光和颜色的感知方面的信息请参见Glassner(1995)。

练习题

12.1 给出从RGB颜色值到HSV值的转换表达式。

12.2 给出从HSV颜色值到RGB值的转换表达式。

12.3 编写一个从显示菜单中选择HSV颜色参数的交互式过程，然后将HSV值转换成帧缓存中的RGB值。

12.4 编写一个程序，用三个滑动块交互地选择HSV颜色参数。

12.5 修改上一练习题中的程序，显示选中颜色对应的RGB分量的值。

12.6 修改上一练习题中的程序，用一个小窗口显示三个RGB分量的颜色及合成的颜色。

12.7 给出从RGB颜色值到HLS颜色参数的转换表达式。

12.8 给出从HLS颜色值到RGB值的转换表达式。

12.9 编写一个程序，从任意指定的两个RGB空间位置间生成一组线性插值的颜色。

12.10 编写一个在RGB空间的一个子空间中交互地选择颜色值的子程序。

12.11 编写一个程序，从任意指定的HSV空间的两个位置间生成线性插值颜色。

12.12 编写一个程序，从任意指定的HLS空间的两个位置间生成线性插值颜色。

12.13 在视频监视器上显示两个靠在一起的RGB颜色网格。使用一组随机选择的RGB颜色填入一个网格，再从一个较小的RGB子空间中选择一组颜色填入另一个网格。试验不同的随机选择和不同的RGB子空间，比较两个颜色网格。

12.14 分别使用HSV和HLS颜色空间选择的颜色来显示练习题12.13中的两个网格。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第13章　计算机动画

计算机生成动画的代表性应用有娱乐（电影和卡通片）、广告、科学和工程研究以及培训和教学。尽管我们在考虑动画时暗指对象的移动，但术语“**计算机动画**”（computer animation）通常指场景中任何随时间而发生的视觉变化。除了通过平移、旋转来改变对象的位置之外，计算机生成的动画还可以随时间进展而改变对象大小、颜色、透明性和表面纹理等。广告动画经常把一个对象形体变成另一个，例如将一个汽车油罐变成汽车发动机。计算机动画还可以通过改变照相机的参数（例如位置、方向和焦距）而产生。我们还可以通过改变光照效果和其他参数以及照明和绘制过程来生成计算机动画。

许多计算机动画的应用要求有真实感的显示。利用数值模型来描述的雷暴雨或其他自然景象的精确表示对评价该模型的可靠性是很重要的。同样，培训飞机驾驶员和大型设备操作员的模拟器必须生成环境的精确表示。另一方面，娱乐和广告应用有时较为关心视觉效果。因此可能使用夸张的形体与非真实感的运动和变换来显示场景。但确实有许多娱乐和广告应用要求计算机生成场景的精确表示。在有些科学和工程研究中，真实感并不是一个目标。例如，物理量经常使用随时间而变化的伪彩色或抽象形体来显示，以帮助研究人员理解物理过程的本质。

创建动画序列有两种基本方法：**实时动画**（real-time animation）和**逐帧动画**（frame-by-frame animation）。实时动画的每个片段在生成之后就立即播放，因此生成动画的速率必须符合刷新频率的约束。对于逐帧动画，场景中每一帧是单独生成和存储的。然后，这些帧可以记录在胶片上或以“实时回放”模式连贯地显示出来。简单的动画可以实时生成，而复杂动画的生成要慢得多，通常逐帧生成。然而，有些应用不论动画复杂与否，始终要求实时生成。例如，为了即时响应控制命令的改变，飞行模拟器必须生成实时的计算机动画。在这些场合，我们常常采用专用的硬件和软件系统来快速生成复杂的动画序列。

13.1　计算机动画的光栅方法

在多数情况下，我们在程序中使用实时方法生成简单的动画序列。但对于光栅扫描系统，我们通常一次一帧地生成动画序列，因而可以将生成的帧保存到文件中供以后观看。按顺序循环播放这些帧就可以观看动画，也可以将这些帧转换为胶片。如果需要实时生成动画，产生运动帧的速度必须足够快，以保证显示出连续的运动序列。对于复杂的场景，一个刷新周期的大部分时间被用来创建动画的一帧。因此，先生成的对象在刷新周期内显示的时间较长，而在刷新周期末期生成的对象几乎刚显示出来就消失了。进而，对于很复杂的场景，创建一个帧所需的时间大于一个刷新周期，这将导致运动漂移和帧破裂的显示缺陷。由于屏幕显示是通过刷新缓存中被连续修改的像素值来生成的，我们可以利用光栅屏幕刷新过程的某些特性来快速产生运动序列。

13.1.1　双缓存

在光栅系统中生成实时动画的一种方法是使用两个刷新缓存（称为双缓存）。初始状态时，在第一个缓存中创建动画的一帧；然后，当用该缓存中的帧刷新屏幕时，在第二个缓存中创建下一

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

帧；下一帧创建完成后，互换两个缓存的角色；因而刷新进程开始用第二个缓存中的帧来刷新屏幕，同时在第一个缓存中继续创建下一帧。这种轮换缓存的操作贯穿于整个动画显示过程中。支持双缓存的图形库通常提供一个激活双缓存的函数和一个互换缓存角色的函数。

当发出一个转换两个缓存的命令后，交换操作可能以不同的节拍来执行。最直接的方法是在当前刷新周期的末期（即电子束垂直回扫阶段）交换两个缓存。如果程序可以在一个刷新周期（例如1/60秒）内创建好一帧，则运动序列的显示可以与屏幕刷新速率同步。但如果创建一帧的时间长于一个刷新周期，则在创建下一帧的时候，当前帧要显示两个或更多的刷新周期。例如，如果屏幕刷新速率为每秒60帧，并且创建一帧需要1/50秒，则每帧需要显示两次，即动画速率只有每秒30帧。类似地，如果创建一帧需要1/25秒，则动画速率降为每秒20帧，因为每帧要显示三次。

使用双缓存时，如果创建一帧的时间非常接近刷新周期的整数倍，则容易出现不规则动画帧率的问题。例如，如果屏幕刷新率为60帧/秒，则当创建一帧的时间在1/60秒、2/60秒、3/60秒或更多倍的左右时，很可能导致不稳定帧率。由于生成图形元素及其属性的函数的执行时间的微小变化，有些帧创建的时间短一点，而有些帧会长一点。这将导致动画帧率突然地和不稳定地发生变化。弥补这种效应的一种方法是在程序中加入少许时延，另一种方法是改变运动或场景描述来缩短创建帧的时间。

13.1.2 用光栅操作生成动画

使用矩形像素阵列的块移动可以为有限的应用生成实时光栅动画。游戏程序经常使用这种动画技术。例如5.6节中，在xy平面上移动一个对象的简单方法是将定义该对象形状的一组像素从一个位置移动到另一个位置。90°的倍数的二维旋转是很容易实现的，尽管可以利用反走样过程绕任意角度旋转一个矩形块像素。对于非90°的倍数的二维旋转，则需要确定旋转后重叠区域的百分比。执行光栅操作序列可以实现二维或三维对象的实时动画，只要将动画限制为在投影平面上的运动。这样不会要求使用观察操作和可见面算法。

我们还可以沿一个二维路径使用**颜色表变换**（colortable transformation）来实现对象的动画。其中，我们沿运动路径的相连位置预先定义对象，并将相继块的像素值设为颜色表入口。将对象的第一个位置的像素设为前景色，而其他对象位置的像素设为背景色。然后通过修改颜色表的值将运动路径上后一对象位置的像素设为前景色而前一对象位置的像素设为背景色来实现动画（参见图13.1）。

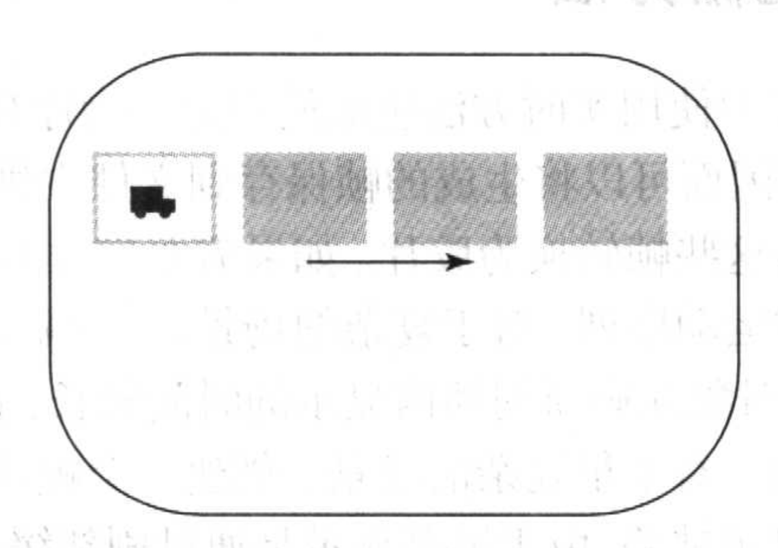


图13.1 实时光栅颜色表动画

13.2 动画序列的设计

创建动画序列是一项复杂的工作，特别是包含一个情节和多个对象时，每个对象都可能以不同的方式运动。通常，一个动画序列按照以下几步进行设计：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

- 故事情节拆分
- 对象定义
- 关键帧描述
- 插值帧的生成

情节板（storyboard）是动作的概述，它将一个运动序列定义为一组即将发生的事件。根据目标动画的类型，情节板可以由一组粗略的素描加上对运动的简单描述组成，也可以是一个关于动作的基本思路的列表。最初，运动的素描被贴在大板上用来全面观察动画项目，因而称为情节板。

为动作的每一个参加者给出**对象定义**（object definition）。对象可能使用基本形体如多边形或样条曲线进行定义。另外，情节中每一角色或对象的运动也通常给出描述。

一个**关键帧**（key frame）是动画序列中特定时刻的一个场景的详细图示。在每一个关键帧中，每一个对象（或角色）的位置依赖于该帧的时刻。某些关键帧选择在动作的极端位置，另一些则以不太大的时间间隔进行安排。复杂的运动要比简单的缓慢变化运动需要更多的关键帧。设置关键帧通常是高级动画师的任务，并且通常为动画中的每个角色安排一个单独的动画师。

插值帧（in-betweens）是关键帧之间的帧。插值帧的数量取决于用来显示动画的介质。电影胶片要求每秒 24 帧，而图形终端按每秒 60 帧以上来刷新。一般情况下，运动的时间间隔设定为每一对关键帧之间有 3 ~ 5 个插值帧。依赖于为运动指定的速度，有些关键帧可重复使用。一分钟没有重复的电影胶片需要 1440 帧。如果每两个关键帧间有 5 个插值帧，则需要 288 幅关键帧。

可能还要求其他一些依赖于应用的任务。包括运动的验证、编辑和音轨的生成与同步。生成一般动画的许多功能现在都由计算机来完成。图 13.2 和图 13.3 展示了计算机生成的动画序列的帧。

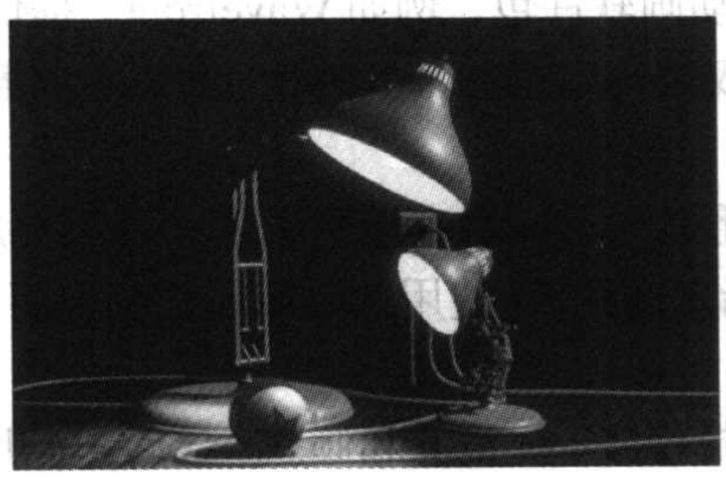


图 13.2　获奖计算机动画短片 *Luxo Jr* 中的一帧。该片使用一个关键帧动画系统和卡通动画技术设计而成，以提供灯的仿生操作。最终图像使用多光源和过程式纹理技术进行控制（Pixar 提供）

图 13.3　获得奥斯卡奖的第一个计算机动画短片 *Tin Toy* 中的一帧。使用关键帧系统设计而成，该片大量使用了面部表情模型。最终图像使用了过程式明暗技术、自身阴影技术，运动模糊技术和纹理匹配技术绘制而成（Pixar 提供）

13.3　传统动画技术

电影动画师使用多种方法来描绘和强调运动序列。这些方法包括对象变形、间隔动画帧、运动的预期与完结以及动作聚焦。

挤压和拉伸（squash and stretch）是模拟加速效果（特别是对非刚性物体）的最重要的技术之一。图 13.4 展示了使用该技术来强调一个弹跳的球的加速和减速。当球加速时，它开始拉伸。当球撞到地面并停下时，它先被压缩（挤压）然后在加速向上弹跳时又开始拉伸。

另一个技术是**定时**（timing），即确定运动帧之间的间隔。一个缓慢运动的物体用多个间隔很近的帧表示，而一个快速运动的物体用其运动路径上的较少的帧来表示。图 13.5 演示了这种效果，其中帧之间的位置变化随着球的运动变快而增加。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

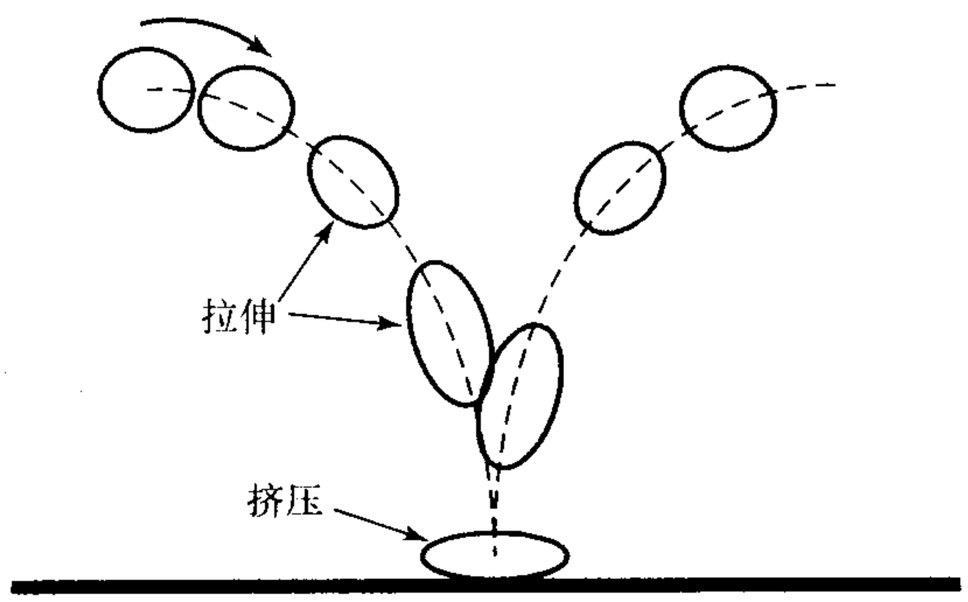


图 13.4 弹跳的球示例强调物体加速的“挤压和拉伸”技术

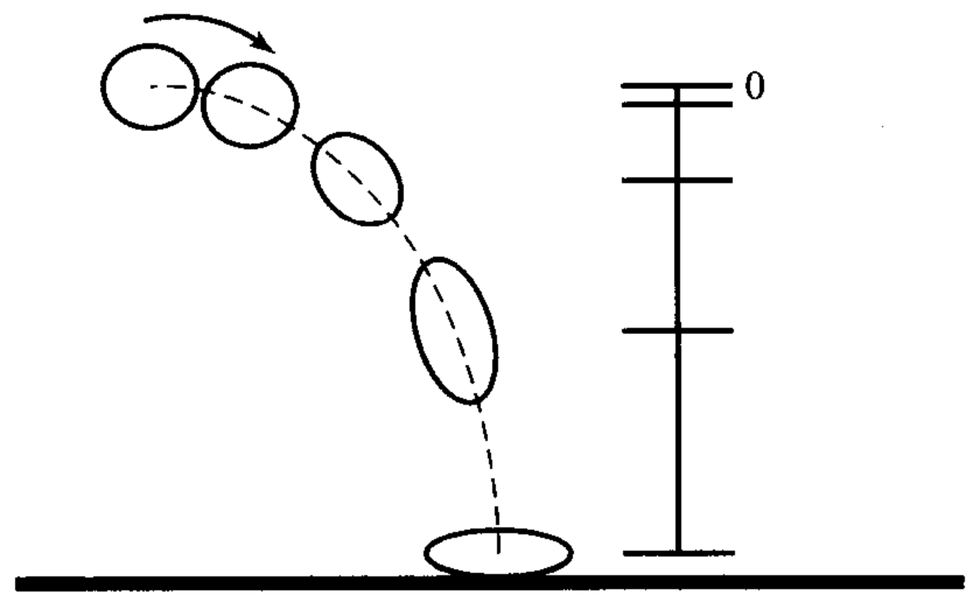


图 13.5 弹跳的球的运动帧之间的位置变化随着球速度的增加而增加

对象运动也可以通过创建预备动作来强调。预备动作指出对于即将发生的运动的**预期**（anticipation）结果。例如，一个卡通角色在起跑之前可能会向前倾并且扭转身体，或者在抛球之前做一个挥臂动作。同样，**完结动作**（follow-through action）也可以用来强调之前的运动。抛球之后，卡通角色可能继续摆臂到身后，或者跑动中突然停下来后帽子掉了。动作还可以用**分级**（staging）来强调。这可以使用任何聚焦到场景中的重要部分的方法，例如聚焦到一个正在藏东西的角色。

13.4 通用计算机动画功能

很多软件包不是支持通用动画设计，就是支持专用动画设计。典型的动画功能包括管理对象运动、生成对象视图、产生照相机运动和生成插值帧。某些动画软件包，例如 Wavefront，提供了整体动画设计和处理单个对象的专门功能。其他专用软件包仅处理动画的某些特性，例如生成插值帧或制作形体动画的系统。

动画软件包中有存储和管理对象数据库的功能。对象形状及其参数存于数据库中并可更新。其他的对象功能包括生成运动和绘制对象表面。运动可依赖指定的约束、使用二维或三维变换而生成。然后可以使用标准函数来识别可见曲面并应用绘制算法。

另一种典型功能是模拟照相机的运动，标准的运动有拉镜头、摇镜头和倾斜。最后，给出对关键帧的描述，然后自动生成插值帧。

13.5 计算机动画语言

通用语言，例如 C、C++、Lisp 或 Fortran，常用来编写设计和控制动画序列的函数。但是人们已经开发了若干种专用的动画语言，它们通常包括图形编辑器、关键帧生成器、插值帧生成器以及标准的图形子程序。图形编辑器让我们使用样条曲面、结构实体几何方法或其他表示框架来设计和修改对象形状。

动画描述中的一个重要任务是场景描述（scene description）。这包含对象和光源的定位，光度参数（光源强度和表面照明特性）的定义，以及照相机参数（位置、方向和镜头特性）的设定。另一标准功能是动作描述。这包括对象和照相机的运动路径安排。我们还需要一般的图形子程序：观察和投影变换、作为加速函数或运动路径描述的生成对象运动的几何变换、可见面识别以及表面绘制操作。

关键帧系统（key-frame system）最初是专用的动画子程序，用来从用户描述的关键帧简单地生成插值帧。现在，这些子程序通常作为更加通用的动画软件包中的一个组件。最简单的情况下，场景中每一对象定义为通过关节连接并具有有限自由度的刚体。例如，图 13.6 中的单臂机器人有 6

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

个自由度：手臂挥动、肩的转动、肘部伸展、投掷、左右摇转和滚动。我们可以让基座进行三维平移而使自由度增加到 9 个（参见图 13.7）。如果我们让基座旋转，则机器人可有 12 个自由度。相比之下，人体有 200 多个自由度。

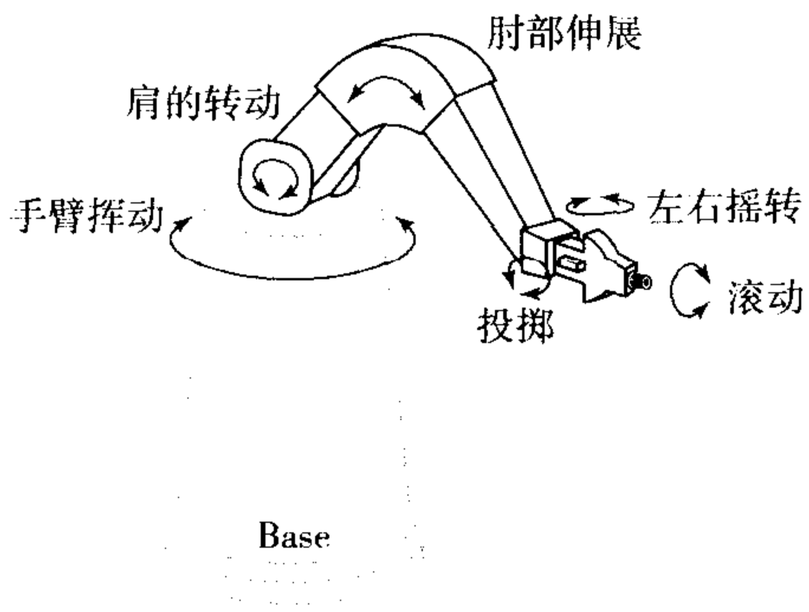


图 13.6　固定的单臂机器人的自由度

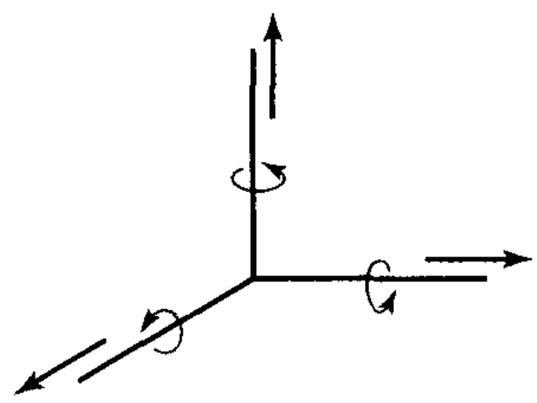


图 13.7　机器人手臂基座的平移和旋转自由度

参数系统（parameterized system）将对象运动特征作为对象定义的一部分进行描述。可调整的参数控制某些对象特征，如自由度、运动限制和允许的形体变化等。

脚本系统（scripting system）允许通过用户输入的脚本来定义对象描述和动画序列。各种对象和运动的库按脚本进行构造。

13.6　关键帧系统

关键帧系统可以从两个（或多个）关键帧的描述生成一组插值帧。运动路径可以使用运动学描述（kinematic description），如使用一组样条曲线来给出运动路径，也可以通过说明作用在动画对象上的力来给出基于物理（physically based）的运动描述。

对于复杂的场景，可以将一帧分解为多个称为 cels（celluloid transparencies）的部分或对象。术语“cels”来自卡通动画技术。背景和场景中的每个角色被分别放在不同的幻灯片上。将这些幻灯片按从背景到前景的顺序叠在一起，然后拍照得到一个完整的帧。给定动画路径，我们根据关键帧的时间通过插值得到各个角色的下一个 cels 的位置。

对象的形状由于使用复杂的对象变换而随时间改变。这些例子有衣服、面部特征、放大的细节、演化的形体、爆裂或分解的对象，或将一个对象变换成另一个。对于用多边形网格表示的曲面，这些改变将导致多边形形状的重大改变，因而每个多边形的边数在各帧之间可能不同。根据定义关键帧的需要来增减多边形的边，可以将这些改变合并到生成插值帧的过程中。

13.6.1　变形

对象的形状从一个形态到另一个形态的变换称为**变形**（morphing，是metamorphosing的缩写）。动画师通过在两个关键帧之间的插值帧转变多边形的形状来为变形建模。

为一个对象的变换指定两个关键帧，每帧包含不同数量的线段指定对象的变换。我们先调整其中一帧的对象描述，使两帧有相同数量的多边形边数（或顶点数）。图 13.8 给出了这一步预处理。关键帧 k 中的一条线段变换成关键帧 $k+1$ 中的两条线段。由于关键帧 $k+1$ 有额外的顶点，因此在关键帧 k 的顶点 1 和 2 之间添加顶点，从而平衡两帧中的顶点数（边数）。使用线性插值方法来产生插值帧。将添加的关键帧 k 的顶点沿图 13.9 中给出的直线路径迁移到顶点 3′。图 13.10 给出了一个三角形展开为一个四边形的例子。图 13.11、图 13.12 给出了电视广告中的变形例子。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

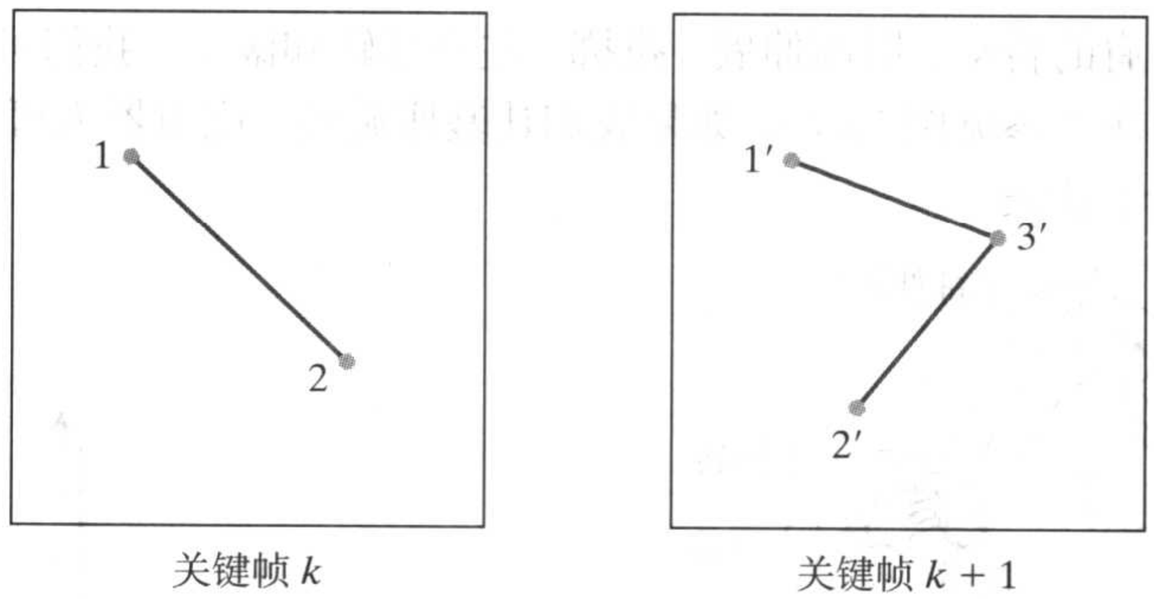


图 13.8　关键帧 k 中顶点位置 1 和 2 之间的一条边演化为关键帧 $k+1$ 中两条连接的边

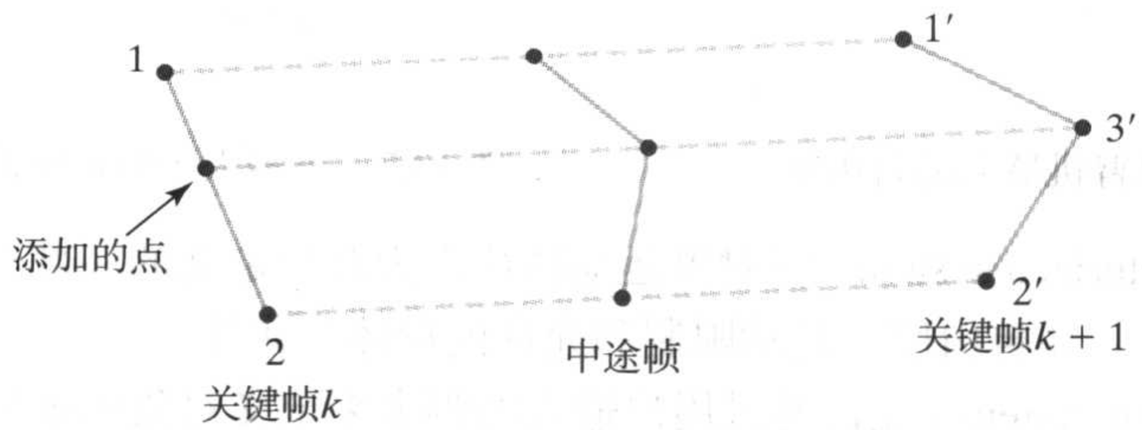


图 13.9　将关键帧 k 中的一条线段变换到关键帧 $k+1$ 中连接的两条线段的线性插值

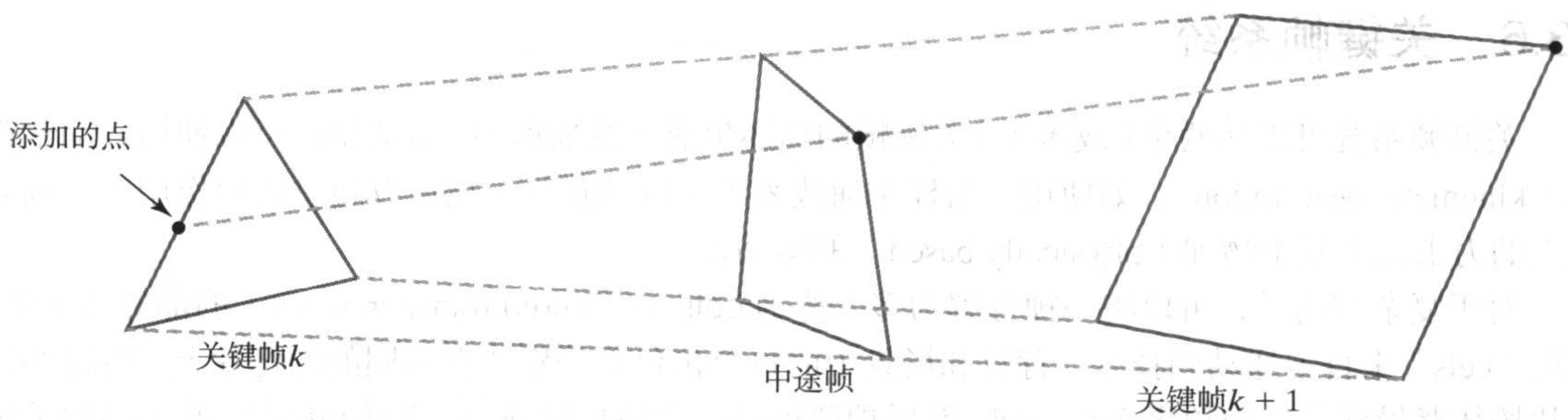


图 13.10　三角形变换到四边形的线性插值

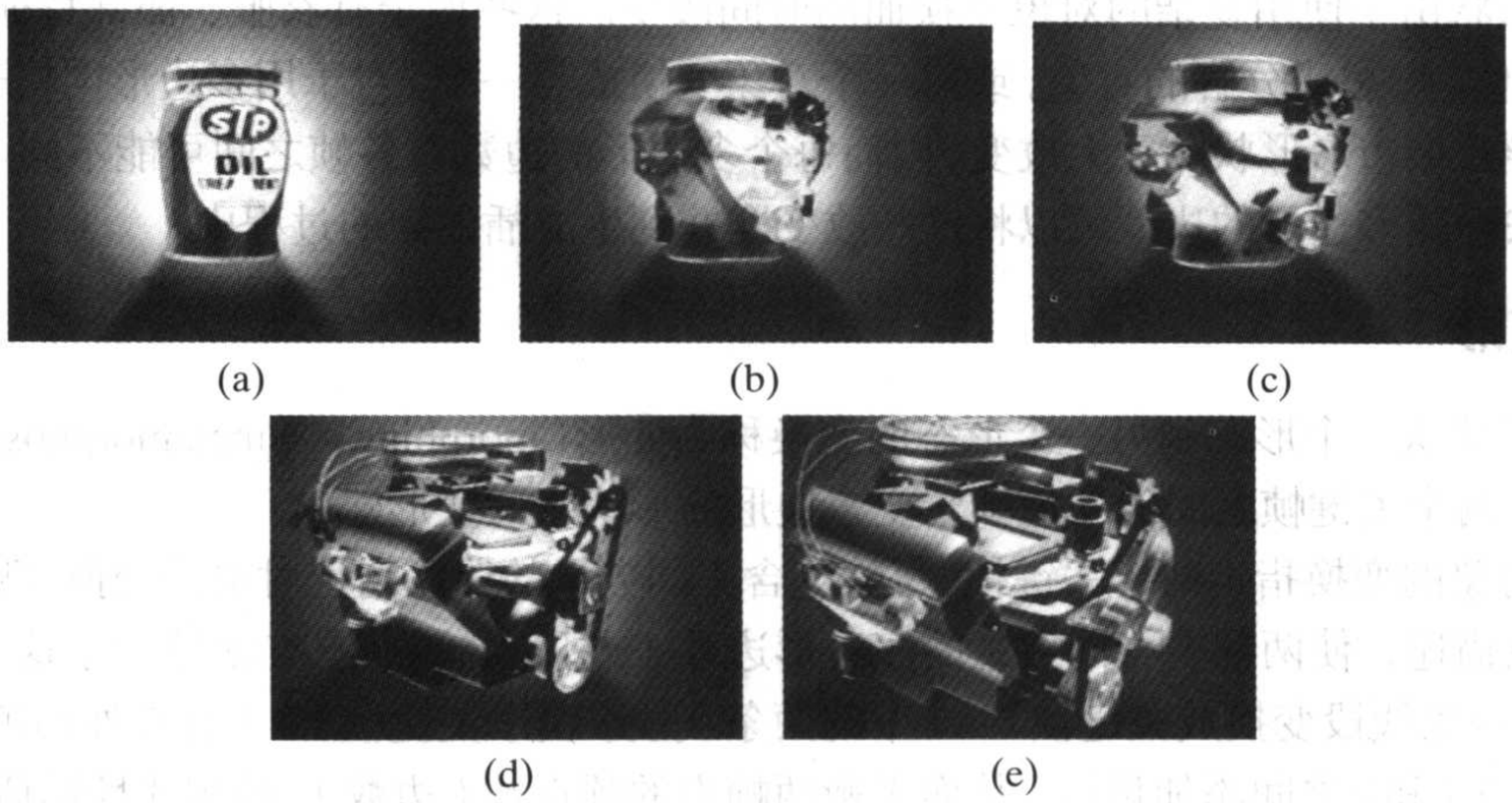


图 13.11　STP 机油壶变为发动机组的转变过程（Silicon Graphics 公司提供）

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

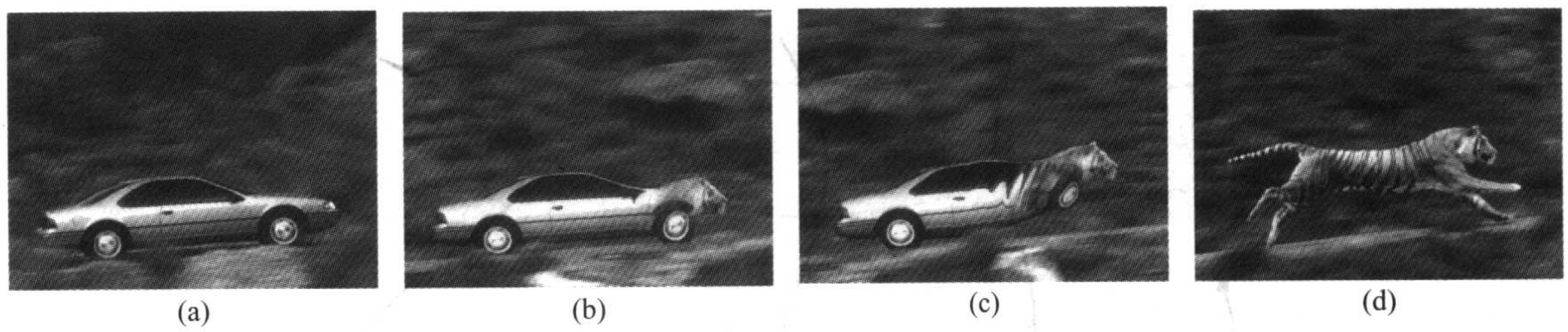


图 13.12　运动中的汽车变换成奔跑中的老虎（美国 Exxon 公司和 Pacific Data Images 提供）

我们可以把在一个关键帧中添加一定量的边或顶点称为关键帧补偿通用预处理规则。先考虑补偿边的数量，其中参数 L_k 和 L_{k+1} 表示两个相继帧的线段数。待补偿的线段的最大和最小数量确定如下：

$$L_{\max} = \max(L_k, L_{k+1}), \qquad L_{\min} = \min(L_k, L_{k+1}) \tag{13.1}$$

且

$$\begin{aligned} N_e &= L_{\max} \bmod L_{\min} \\ N_s &= int\left(\frac{L_{\max}}{L_{\min}}\right) \end{aligned} \tag{13.2}$$

边补偿预处理由下列两步实现：

(1) 将边数少的关键帧 $\text{keyframe}_{\min}$ 的 N_e 条边分成 $N_s + 1$ 部分；

(2) 将边数少的关键帧 $\text{keyframe}_{\min}$ 的余下边分成 N_s 部分。

例如，如果 $L_k = 15$、$L_{k+1} = 11$，我们将关键帧 $k+1$ 的 4 条边的每一条分成两部分。而关键帧 $k+1$ 的剩余边则不做处理。

如果补偿顶点数，可以使用参数 V_k 和 V_{k+1} 表示两个连续帧的顶点数。在这种情况下，待补偿的顶点的最大和最小数量确定如下：

$$V_{\max} = \max(V_k, V_{k+1}), \qquad V_{\min} = \min(V_k, V_{k+1}) \tag{13.3}$$

且

$$\begin{aligned} N_{ls} &= (V_{\max} - 1) \bmod (V_{\min} - 1) \\ N_p &= int\left(\frac{V_{\max} - 1}{V_{\min} - 1}\right) \end{aligned} \tag{13.4}$$

使用顶点数的预处理通过以下步骤实现：

(1) 在点数最少的关键帧 $\text{keyframe}_{\min}$ 中，在 N_{ls} 线段的部分中添加 N_p 个点；

(2) 在点数最少的关键帧 $\text{keyframe}_{\min}$ 中，在余下的边中添加 $N_p - 1$ 个点。

对于三角形 – 四边形的例子，$V_k = 3$ 而 $V_{k+1} = 4$，N_{ls} 和 N_p 均为 1。因此，在三角形的一条边上添加一个点，其余边不用添加点。

13.6.2　模拟加速度

曲线拟合算法常用来指定关键帧之间的动画路径。给定关键帧的顶点位置，可以使用线性或非线性路径进行拟合。图 13.13 给出了关键帧的非线性拟合。为了模拟加速度，我们可以调整插值的时间间隔。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

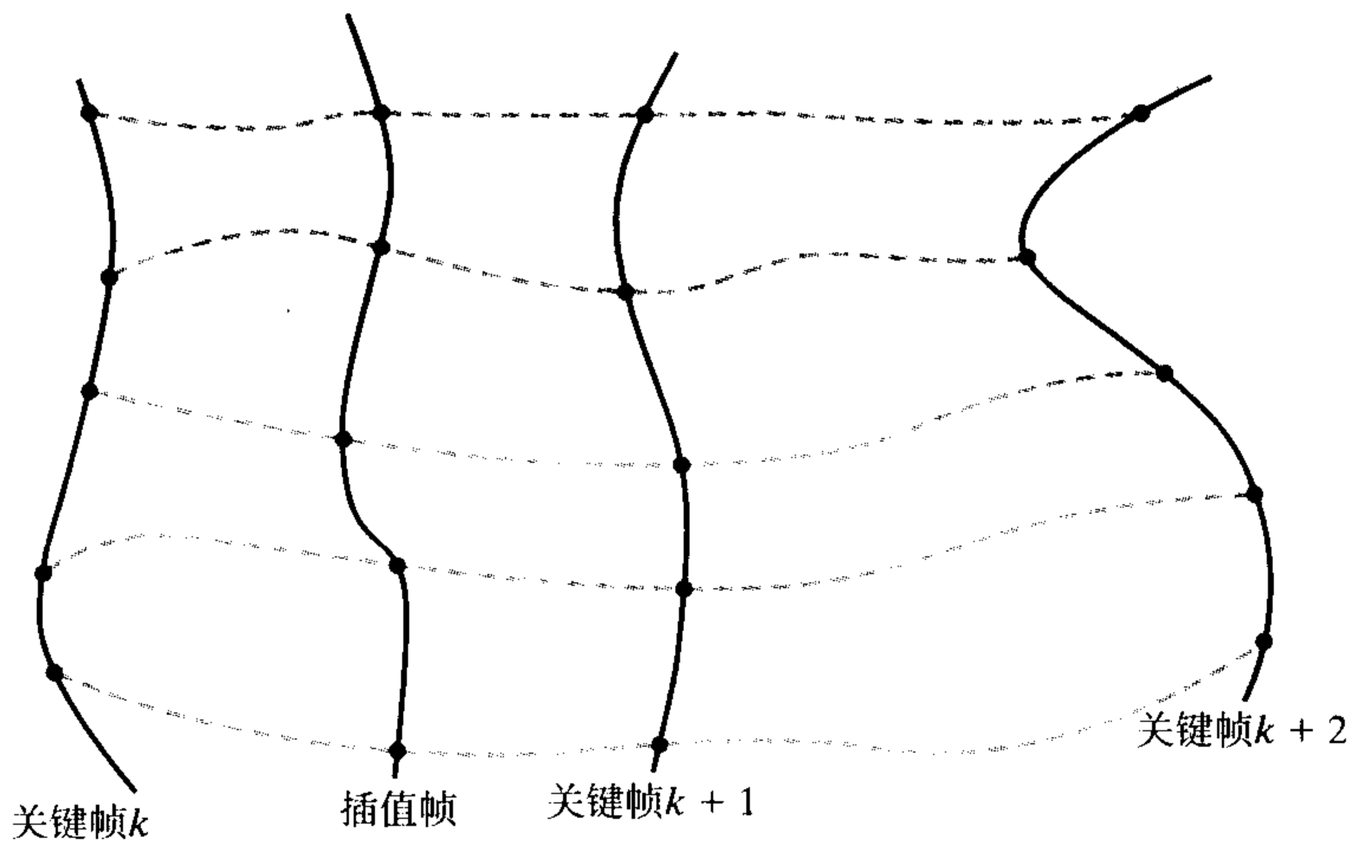


图 13.13　使用非线性样条拟合关键帧的顶点位置

对于恒定速度（加速度为0）的运动，我们使用等时间间隔的插值帧。假定我们需要在时间t_1和t_2的关键帧中插入n帧（参见图13.14），关键帧之间的时间段则分成n+1个子段，给出插值帧的间隔：

$$\Delta t = \frac{t_2 - t_1}{n+1} \tag{13.5}$$

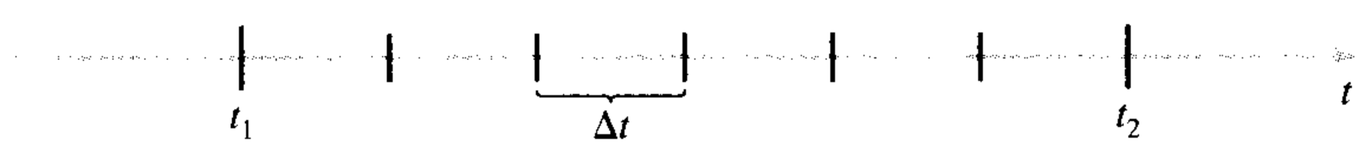


图 13.14　匀速运动的插值位置

计算第j个插值帧的时刻：

$$tB_j = t_1 + j\Delta t, \qquad j = 1, 2, \ldots, n \tag{13.6}$$

并利用这个时间值确定此运动帧的坐标位置、颜色值以及其他的物理参数。

速度变化（非零加速度）在动画电影或卡通片中是必需的，尤其是在运动序列的开始和结束。我们可以使用样条曲线或三角函数来给出动画路径中的启动和减速部分的模型。抛物线和三次时间函数应用于加速度建模。动画软件包一般提供三角函数来模拟加速度。

为了模拟加速（正向加速度），我们让帧之间的时间间隔增加，使得在对象移动加快时有较大的位置变化。我们可以利用下列函数得到增加的间隔：

$$1 - \cos\theta, \qquad 0 < \theta < \pi/2$$

对于n个插值帧，第j个插值帧的时刻可以计算如下：

$$tB_j = t_1 + \Delta t\left[1 - \cos\frac{j\pi}{2(n+1)}\right], \qquad j = 1, 2, \ldots, n \tag{13.7}$$

其中，Δt是两个关键帧之间的时间差。图13.15给出了$n=5$时，三角加速度函数和插值间隔的图示。

我们可以利用$\sin\theta$在$0 < \theta < \pi/2$范围内来模拟降低速度（减速度）的情况。一个插值帧的时间位置现在定义为

$$tB_j = t_1 + \Delta t \sin\frac{j\pi}{2(n+1)}, \qquad j = 1, 2, \ldots, n \tag{13.8}$$

图13.16给出了有5个插值帧时该函数的图示，以及时间间隔的减小程度。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

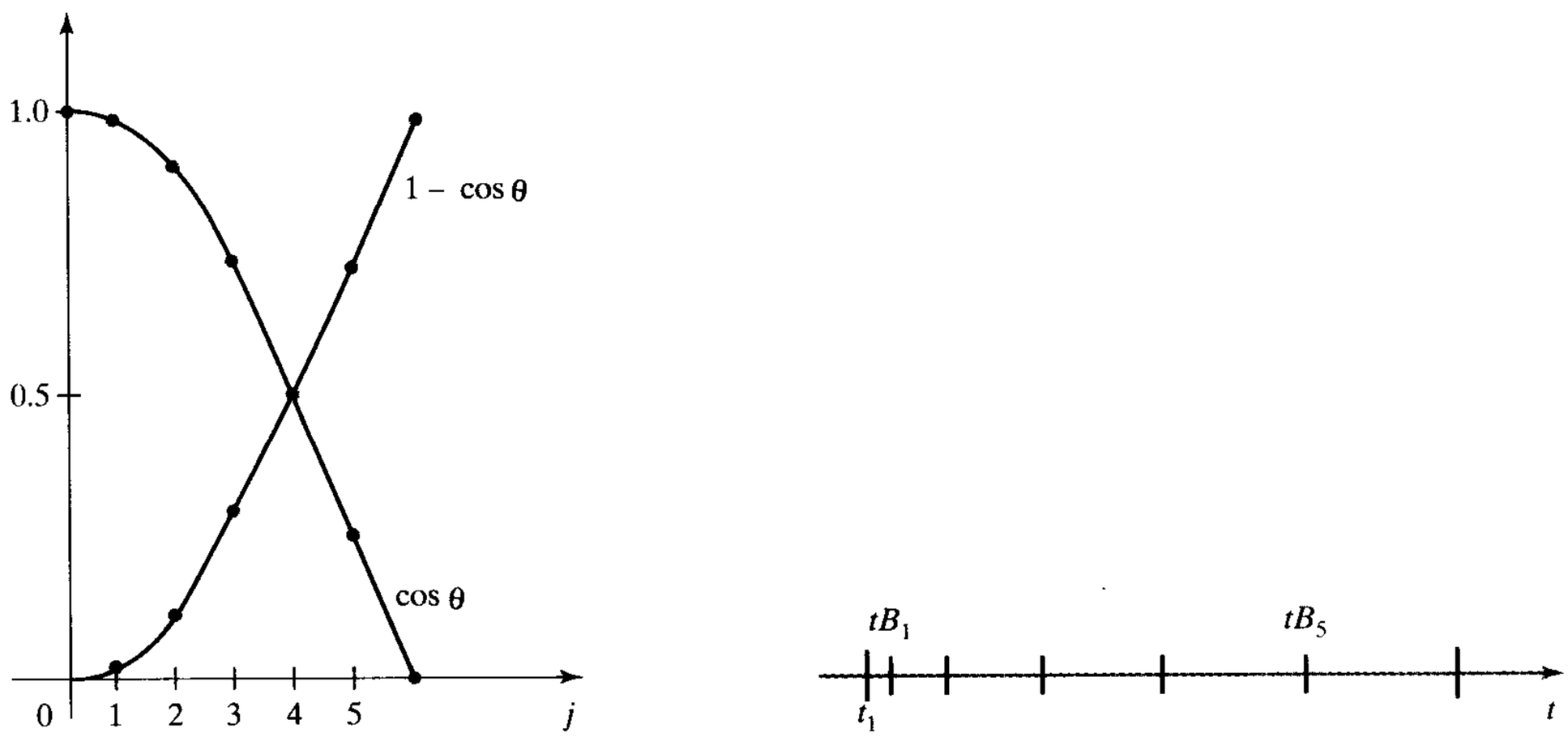


图 13.15　等式（13.7）中，$n=5$ 且 $\theta=j\pi/12$ 时，三角加速度函数和对应的插值间隔，当对象移经每一时间间隔时，产生递增的坐标变化

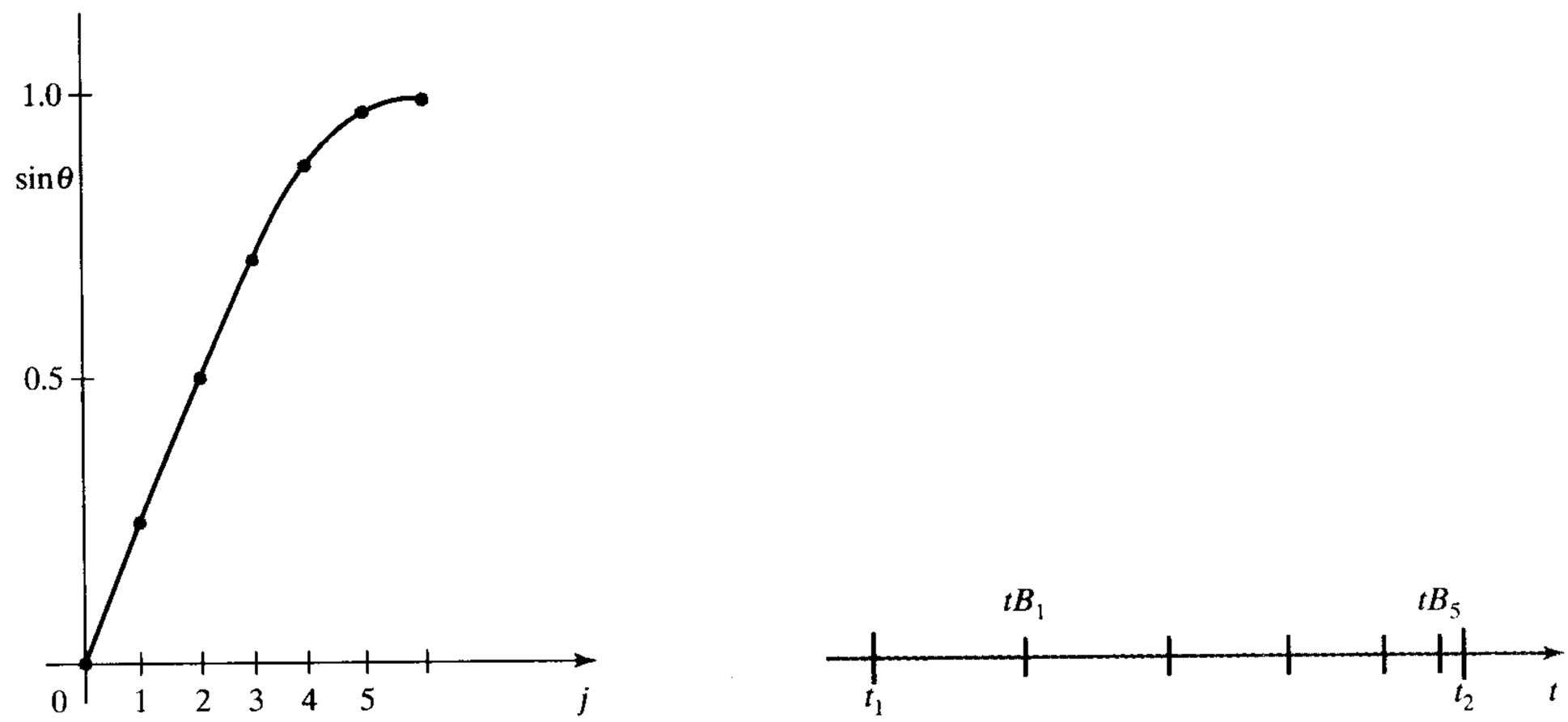


图 13.16　等式（13.8）中，$n=5$ 且 $\theta=j\pi/12$ 时，三角减速度函数和对应的插值间隔，在对象移经每一时间间隔时，产生递减的坐标变化

运动常包含加速和减速，我们可以通过先增加插值时间间隔后，再减小该间隔的办法来实现加减速度的混合建模。实现该时间变化的函数是

$$\frac{1}{2}(1-\cos\theta), \qquad 0<\theta<\pi/2$$

现在第 j 个插值帧的时刻计算如下：

$$tB_j = t_1 + \Delta t\left\{\frac{1-\cos[j\pi/(n+1)]}{2}\right\}, \qquad j=1,2,\ldots,n \tag{13.9}$$

Δt 表示两个关键帧的时间差。运动对象的时间间隔如图 13.17 所示先增加然后减少。

通过使用“骨架”（线框图）来对对象进行初始建模，可以简化处理插值帧。这样可以交互地调整运动序列。在动画序列完全定义好以后，就能完整地绘制对象。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

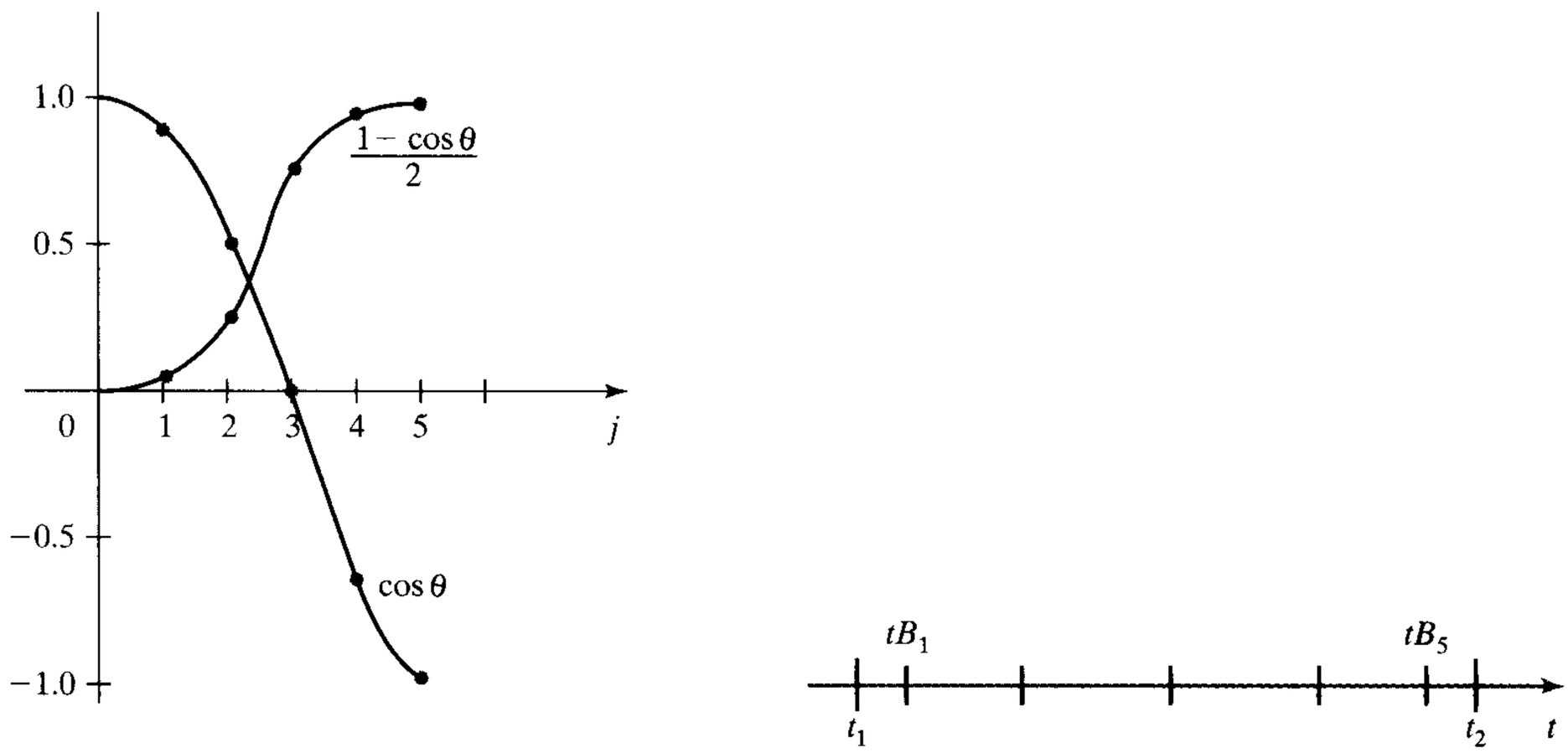


图 13.17　等式（13.9）中，$n=5$ 时，三角加速－减速函数和$(1-\cos\theta)/2$ 对应的插值间隔

13.7　运动的描述

描述动画序列一般可采用包括从显式指定运动路径到描述产生运动的相互作用的一系列方法。因此，我们可以通过给定变换参数、运动路径参数、作用在物体上的力或物体之间相互作用产生运动的细节来定义运动将如何发生。

13.7.1　直接运动描述

定义运动序列的最直接的方法是几何变换参数的直接运动描述法（direct motion specification）。这里，我们显式地给出旋转角度和平移向量，然后使用几何变换矩阵来对坐标位置进行变换。我们也可以使用一个包含这些参数的近似等式来描述某种运动。例如，可以使用衰减的、校正的正弦曲线（参见图 13.18），从而近似地给出跳跃的球的路径：

$$y(x) = A|\sin(\omega x + \theta_0)|e^{-kx} \tag{13.10}$$

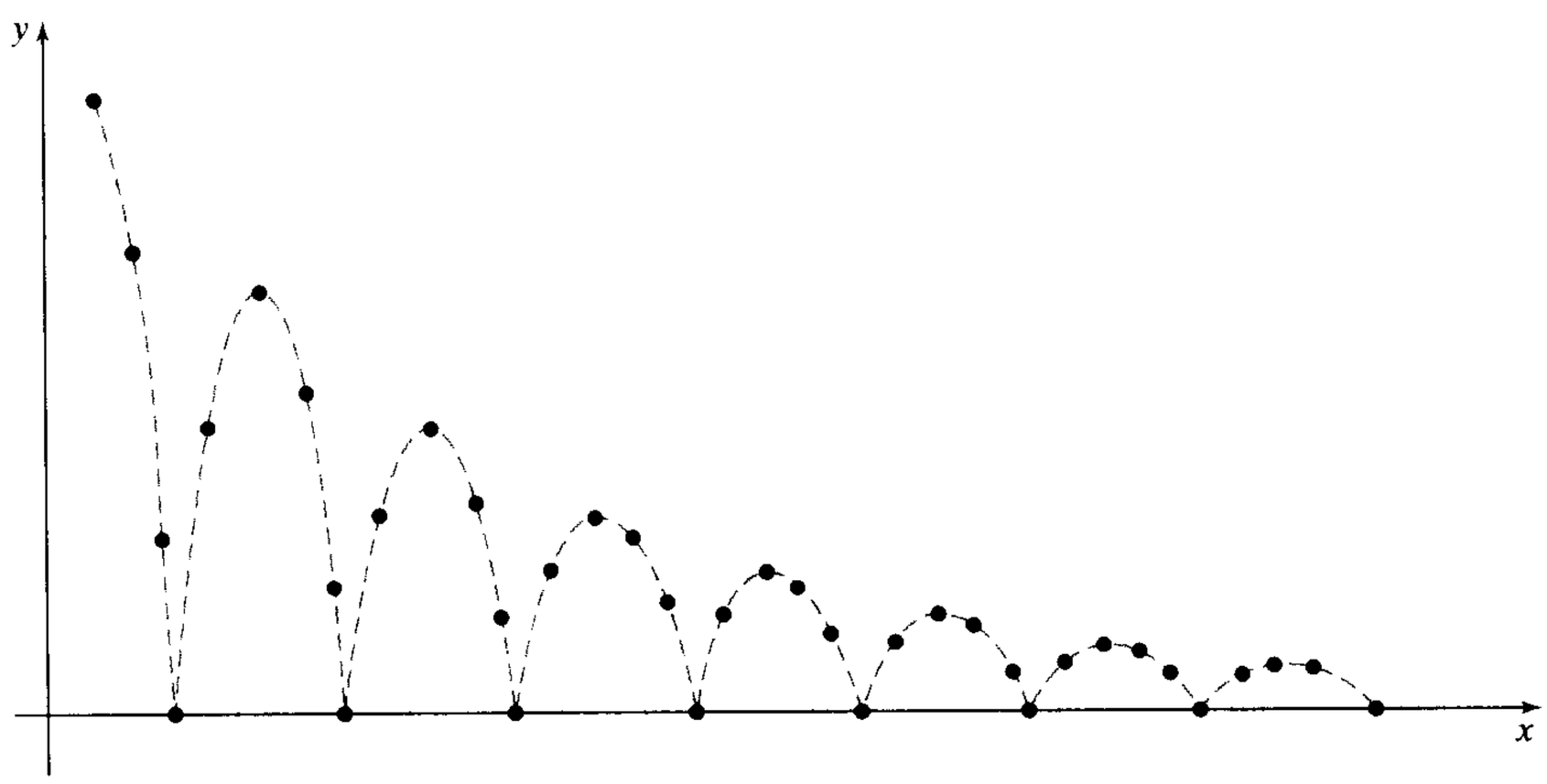


图 13.18　利用衰减的正弦函数式（13.10）来近似给出弹球的运动

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

这里 A 是初始振幅（球高于地面的高度），ω 是角频率，θ_0 是相位角，k 是衰减常数，这种运动描述方法可以用于简单的用户编程的动画序列。

13.7.2　目标导向系统

另一方面，我们可以利用抽象地描述动作的最终结果来概括地描述即将发生的运动。也就是说，通过运动的最终状态来描述动画。由于这些系统根据动画的目标确定出运动参数，因此将其称为目标导向系统。例如，可以说明要把一个对象“走”或“跑”到一个特定位置，或者说明要求一个对象“拾取”某一另外的对象。输入命令将解释成部件运动以实现选择的任务。例如，人体运动可定义成躯干、四肢和其他部分子运动的层次结构。因而，当类似“走到门口”的目标给出后，需要完成此动作的躯干和四肢的运动就会被计算出来。

13.7.3　运动学和动力学

我们还可以使用运动学（kinematics）或动力学（dynamics）来描述构造动画序列。在使用运动学描述时，通过给出运动参数（位置、速度和加速度）来指定动画，而并不引入引起运动的力。对于匀速（零加速度）情况，通过为每一对象给出初始位置和速度向量来指定场景中每一刚体的运动。例如，如果速度指定为(3, 0, –4) km/s，那么该向量给出了直线运动路径的方向且速度（速度的量）是 5 km/s。如果我们指定加速度（速度变化率），就能生成加速、减速和曲线运动路径。我们还可以通过简单地叙述运动路径来给出运动的运动学描述。这通常使用样条曲线来实现。

一种替代方法是使用反向运动学（inverse kinematics）。这里，我们指定在确定时间内对象的初始和最终位置，然后由系统计算移动参数。例如，假定为零加速度，可以确定实现一个对象从初始位置到最终位置运动的匀速度。该方法常用于复杂对象，给定该对象末端节点（如手或脚）的位置和方向，由系统确定实现所需运动的其他节点的运动参数。

另一方面，动力学描述要求产生速度和加速度的力的描述。在该力作用下的对象行为的描述，通常称为基于物理的建模（参见第 8 章）。影响对象运动的力的例子有电磁力、重力、摩擦力和其他机械力。

对象的运动可由描述物理定律的力学公式而得到。例如，重力和摩擦力运动的牛顿定律，描述流体的 Euler 或 Navier-Stokes 公式，以及描述电磁力的 Maxwell 公式。例如质量 m 的牛顿第二定律的一般形式是

$$\mathbf{F} = \frac{d}{dt}(m\mathbf{v}) \tag{13.11}$$

其中，$\mathbf{F}$ 是力向量，$\mathbf{v}$ 是速度向量。如果质量是常数，我们得到等式 $\mathbf{F} = m\mathbf{a}$，这儿 $\mathbf{a}$ 是加速度向量。否则，质量是时间的函数，正如在相对运动中或在单位时间内消耗可计量燃料的宇宙飞船的运动中一样。我们还可以给定对象的初始和最终位置及运动类型，利用反向动力学（inverse dynamics）来获得力。

基于物理模型的应用包括复杂的刚体系统和类似布和塑料材料的非刚体系统。一般情况下，数值方法较多地用于从使用初始条件或边界值的动态方程中获得运动参数。

13.8　关节链形体动画

制作人、动物、昆虫和其他生物的动画的一种基本技术是利用**关节链形体**（articulated figure）建模。关节链形体是由一组通过旋转节点连接的刚性连杆组成的层次结构（参见图 13.19）。这表示

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

将动画对象建模为移动的棍状形体或简化了的骨架。其后再用表示皮肤、毛发、羽毛、衣服或其他遮盖物的曲面包裹骨架。

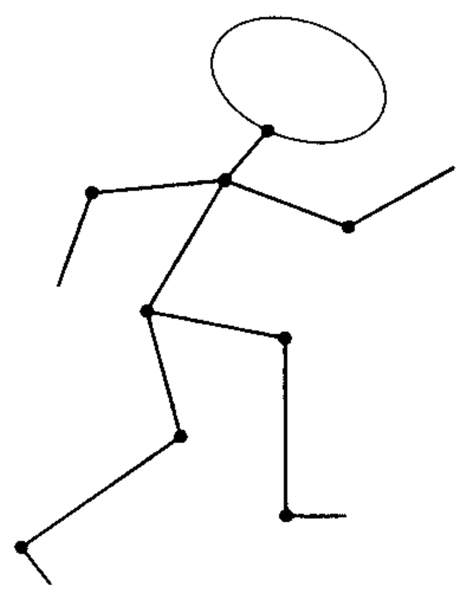


图 13.19　一个有 9 个节点和 12 个连杆的简单关节链形体，椭圆形的头不计在内

关节链形体的连接点（或铰链）位于肩、臀、膝和其他骨骼关节。当身体移动时，这些关节按特定的路径运动。例如，指定对象的运动，肩部就自动按一定的方式移动。并且，肩部移动，手臂也跟着移动。我们定义不同类型的移动（例如走、跑、跳）并将它们与节点和连杆的特定运动关联起来。

例如，图 13.20 定义了一系列走动的腿的运动。臀部节点沿水平线向前平移，而连杆进行一系列围绕臀、膝和踝节点的运动。从直腿开始（参见图 13.20(a)），第一个运动是当臀部前移时膝部弯曲（参见图 13.20(b)）。然后腿向前摆动，回到垂直位置，再向后摆动，如图 13.20(c) ~ 图 13.20(e) 所示。最后的运动是腿向后大幅摆动然后直腿回到垂直位置，如图 13.20(f)和图 13.20 (g)所示。在整个动画过程中，上述运动周期重复进行，直到形体移动指定的距离或时间。

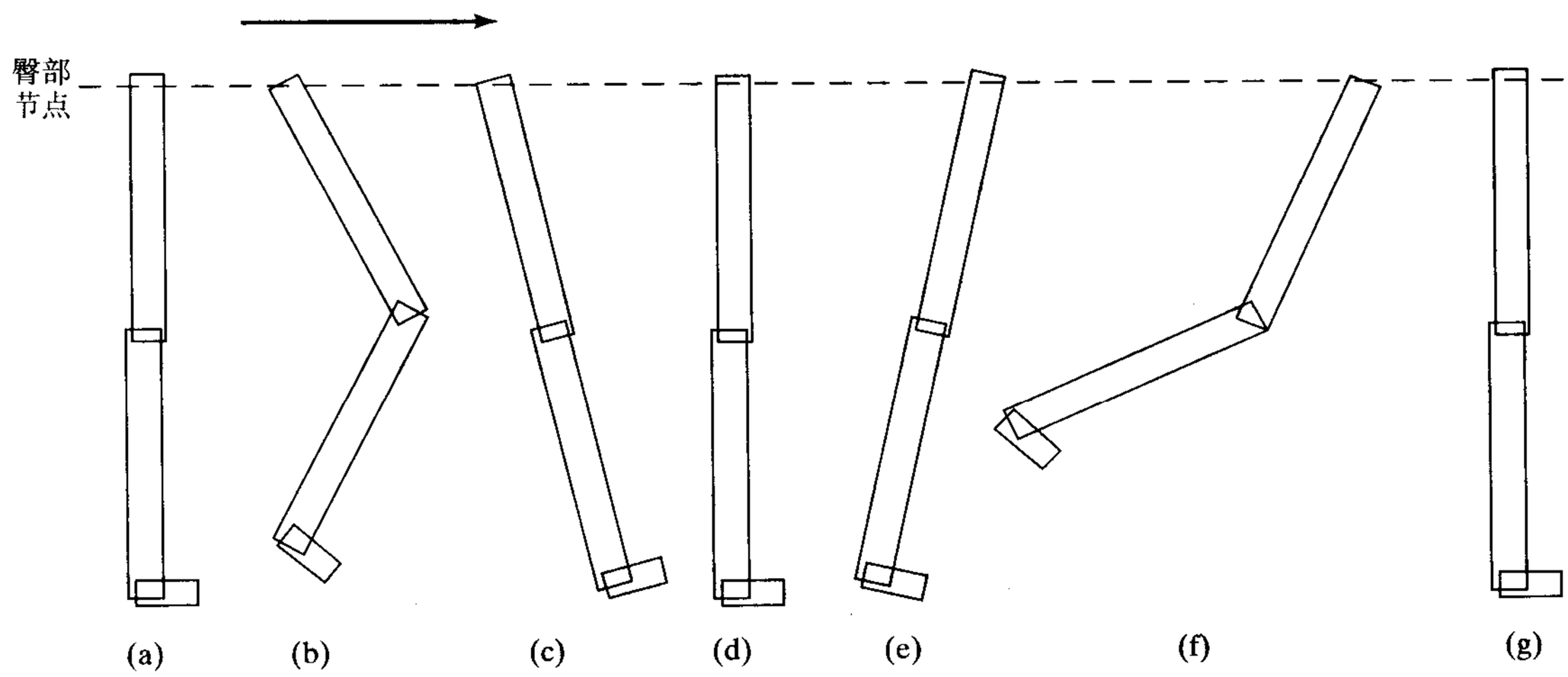


图 13.20　表示正在走路的腿的一组互连连杆的可能运动

当形体移动时，各个关节上还可以加入其他运动。通常，具有变化振幅的正弦曲线运动可以加入到臀关节使其绕躯干运动。类似地，滚动或摇动可以加到肩部，并且头部也可以上下摆动。

形体动画既使用运动学运动描述也使用反向运动学运动描述。指定关节运动一般比较容易，但是在任意的地面上生成简单运动就需要使用反向运动学。对于复杂的形体，反向运动学不保证产生惟一的动画序列。给定一组初始和终止条件，可能存在多种不同的旋转运动。在这种情况下，向系统中添加更多的约束（例如动量守恒）可以产生惟一的结果。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

13.9　周期性运动

当用重复的运动模式（例如一个旋转物体）创建动画时，我们需要确信运动采样（参见4.17节）足够频繁以正确表现运动。也就是说，运动必须与生成帧的速率同步，从而在每个周期中显示足够多的帧来展示真实的运动。否则，动画的显示会不正确。

周期性运动采样过低的典型例子是西部影片中的马车轮倒转现象。图13.21显示了带有一根红色辐条的马车轮旋转的一个完整周期。车轮每秒顺时针旋转18周。如果用标准的电影放映速率（24帧/秒）录制到胶片上，前5帧的情况如图13.22所示。因为每1/24秒车轮只完成3/4转，每个周期只生成一幅动画帧，因此看起来像是朝相反的方向（逆时针反向）旋转。

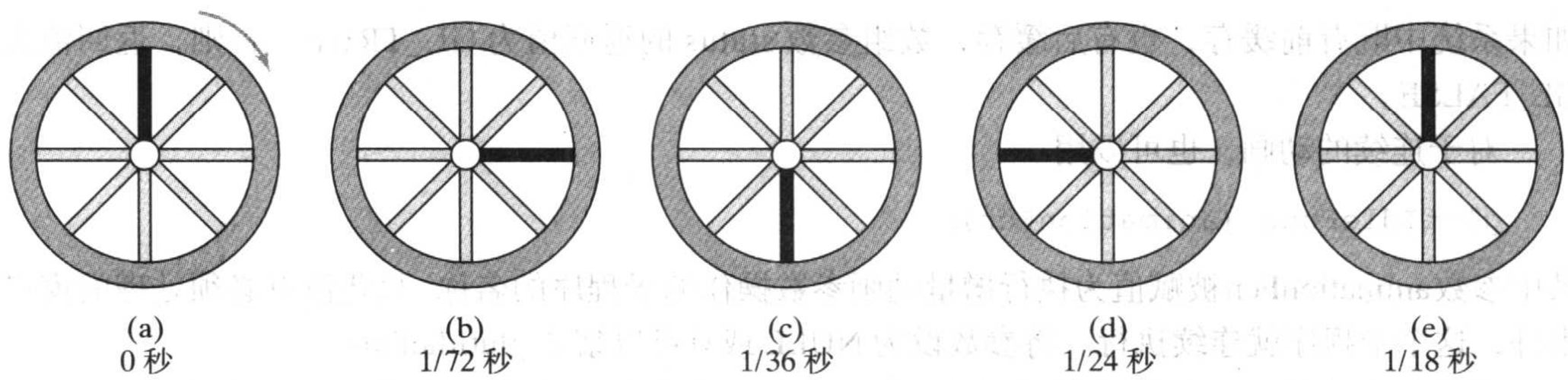


图 13.21　在车轮运动的一个周期内辐条的 5 个位置，车轮每秒转 18 周

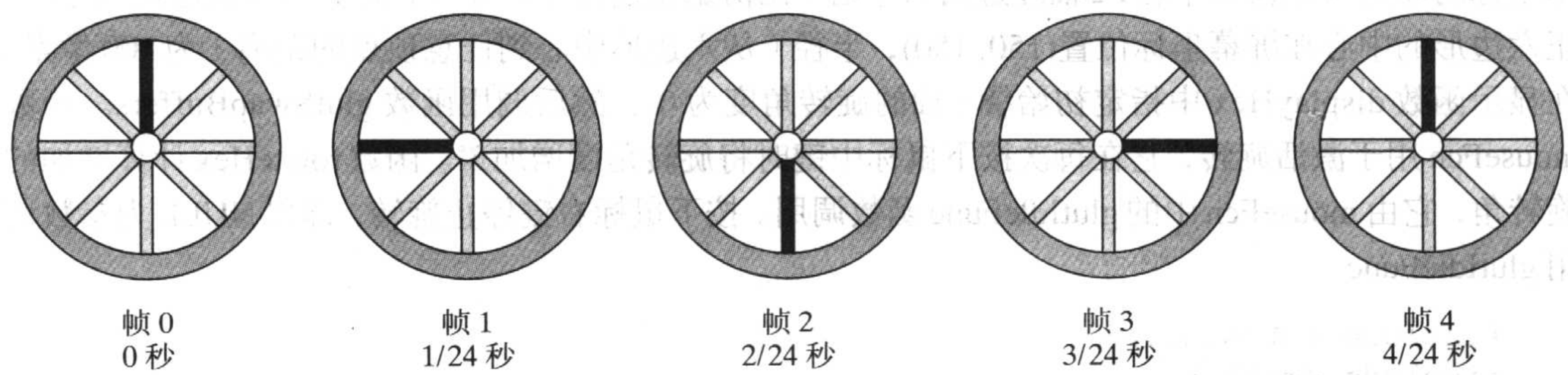


图 13.22　以每秒 24 帧采样的图 13.21 中旋转的车轮的前 5 个胶片帧

在计算机动画中，我们可以通过调节运动参数来控制对周期性运动的采样率。例如，可以设置旋转物体运动的角增量使每秒可以生成多帧。3° 的角增量在旋转一周中产生 120 个运动位置，4° 的角增量产生 90 个位置。对于更快的运动，可以使用更大的旋转步长，只要每个周期的采样次数不太小，并且可以清楚地显示运动。当制作复杂对象的动画时，我们还必须考虑创建帧的时间对刷新率的影响（参见 13.1 节）。如果创建每帧的时间过长，复杂对象的运动会比我们期望的更慢。

显示重复运动还需要考虑的另一个因素是计算运动参数时取整的影响。如5.4节解释的，我们可以周期性地重置参数值以防止由累计误差导致的不稳定运动。对于连续的旋转，可以每个周期（360°）重置参数值。

13.10　OpenGL 动画子程序

OpenGL 核心库中包含光栅操作（参见 5.7 节）和颜色索引分配函数。GLUT（参见 4.3 节）提供改变颜色表值的子程序。其他光栅操作仅作为 GLUT 的例程，因为它们在使用中依赖于窗口系统。而且，某些硬件系统可能不包含类似双缓存等计算机动画的特性。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如果双缓存可用，我们用以下 GLUT 命令来激活双缓存：

```
glutInitDisplayMode (GLUT_DOUBLE);
```

这产生两个交替刷新屏幕的缓存，分别称为前缓存（front buffer）和后缓存（back buffer）。当一个缓存作为当前显示窗口的刷新缓存时，在另一个缓存中创建动画的下一帧。以下函数用于互换两个缓存的角色：

```
glutSwapBuffers ( );
```

我们可以用下列查询来确定一个系统是否支持双缓存操作：

```
glGetBooleanv (GL_DOUBLEBUFFER, status);
```

如果系统中既有前缓存，也有后缓存，数组参数 status 的返回值为 GL_TRUE。否则，返回值为 GL_FALSE。

对于连续的动画，也可以用

```
glutIdleFunc (animationFcn);
```

其中参数animationFcn被赋值为执行增量动画参数操作的子程序的名称。只要没有必须处理的窗口事件，这个子程序就连续执行。将参数设为 NULL 或 0 可以禁止 glutIdleFunc。

下列代码给出一个动画程序的例子。该例子在 xy 平面上绕 z 轴连续旋转一个正六边形。三维屏幕坐标的原点放在窗口中心，z 轴经过窗口中心。在初始化过程中，用一个显示列表描述正六边形。正六边形的中心在屏幕坐标位置(150, 150)，半径（从六边形中心到任意顶点的距离）为 100 像素。在显示函数 displayHex 中指定初始绕 z 轴的旋转角度为 0°，然后调用函数 glutSwapBuffers。函数 mouseFcn 用于激活旋转，它在每次按下鼠标中键时将旋转角度增加 3°。函数 rotateHex 计算增加的旋转角，它由 mouseFcn 中的 glutIdleFunc 函数调用。按下鼠标右键停止旋转，即以 NULL 为参数调用 glutIdleFunc。

```
#include <GL/glut.h>
#include <math.h>
#include <stdlib.h>

const double TWO_PI = 6.2831853;

GLsizei winWidth = 500, winHeight = 500; // Initial display window size.
GLuint regHex;                           // Define name for display list.
static GLfloat rotTheta = 0.0;

class scrPt {
public:
   GLint x, y;
};

static void init (void)
{
   scrPt hexVertex;
   GLdouble hexTheta;
   GLint k;

   glClearColor (1.0, 1.0, 1.0, 0.0);

   /* Set up a display list for a red regular hexagon.
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
 * Vertices for the hexagon are six equally spaced
 * points around the circumference of a circle.
 */
regHex = glGenLists (1);
glNewList (regHex, GL_COMPILE);
   glColor3f (1.0, 0.0, 0.0);
   glBegin (GL_POLYGON);
      for (k = 0; k < 6; k++) {
         hexTheta = TWO_PI * k / 6;
         hexVertex.x = 150 + 100 * cos (hexTheta);
         hexVertex.y = 150 + 100 * sin (hexTheta);
         glVertex2i (hexVertex.x, hexVertex.y);
      }
   glEnd ( );
glEndList ( );
}

void displayHex (void)
{
   glClear (GL_COLOR_BUFFER_BIT);

   glPushMatrix ( );
   glRotatef (rotTheta, 0.0, 0.0, 1.0);
   glCallList (regHex);
   glPopMatrix ( );

   glutSwapBuffers ( );

   glFlush ( );
}

void rotateHex (void)
{
   rotTheta += 3.0;
   if (rotTheta > 360.0)
      rotTheta -= 360.0;

   glutPostRedisplay ( );
}

void winReshapeFcn (GLint newWidth, GLint newHeight)
{
   glViewport (0, 0, (GLsizei) newWidth, (GLsizei) newHeight);

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ( );
   gluOrtho2D (-320.0, 320.0, -320.0, 320.0);

   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ( );

   glClear (GL_COLOR_BUFFER_BIT);
}

void mouseFcn (GLint button, GLint action, GLint x, GLint y)
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com


```
{
   switch (button) {
      case GLUT_MIDDLE_BUTTON:          // Start the rotation.
         if (action == GLUT_DOWN)
            glutIdleFunc (rotateHex);
         break;
      case GLUT_RIGHT_BUTTON:           // Stop the rotation.
      if (action == GLUT_DOWN)
            glutIdleFunc (NULL);
         break;
      default:
         break;
   }
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowPosition (150, 150);
   glutInitWindowSize (winWidth, winHeight);
   glutCreateWindow ("Animation Example");

   init ( );
   glutDisplayFunc (displayHex);
   glutReshapeFunc (winReshapeFcn);
   glutMouseFunc (mouseFcn);

   glutMainLoop ( );
}
```

13.11 小结

动画序列可以逐帧创建，也可以实时创建。当动画是逐帧创建并存储时，创建之后可以将帧转换为胶片或者在视频监视器上快速连续地显示。包含复杂场景的动画通常以逐帧方式创建，而简单的运动序列可以实时显示。

在光栅系统中可以用双缓存方法来加快运动显示。一个缓存用于刷新屏幕，另一个则装入下一帧的显示内容。然后，通常在刷新周期的末期互换两个缓存的角色。

另一种光栅方法用像素值的块转移来执行运动序列。通过简单地将帧缓存中一矩形块中的像素颜色从一个位置移动到另一个位置可以实现平移。90°的旋转可以用平移与像素阵列的行列互换的组合来实现。

颜色表方法可以实现简单的光栅动画。它将一个对象的图像用不同的颜色表值存储在帧缓存的多个位置上。其中一个位置的图像用前景色，而其他位置的图像用背景色。通过快速交换颜色表中的前景色和背景色，就可以将对象显示在不同的屏幕位置上。

生成动画要经过三个开发阶段：情节串联、定义对象和指定关键帧。情节板是动作的概述，而关键帧定义动画中选定位置的对象运动的细节。一旦建立了关键帧，就可以生成一系列插值帧来构成从一个关键帧到下一个的平滑运动。计算机动画可以包括照相机的运动描述和场景中对象的运动路径。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

已经存在多种模拟和强调运动效果的方法。挤压和拉伸效果是标准的强调加速的方法，也可以调节运动帧之间的定时来产生速度变化。其他方法还有加入预备动作、完结动作和聚焦重要动作的分级方法。当描述加速运动时，通常使用三角函数来生成插值帧的时间间隔。

计算机动画可以用专用软件或通用图形软件包生成。计算机动画系统有关键帧系统、参数系统和脚本系统。

许多动画使用变形效果，即将一个对象形体变成另一个。变形效果通过在一个对象的点和线与另一个对象的点和线之间插入过渡的插值帧来实现。

动画中的运动描述可以用直接运动描述或目标导向描述。动画可以用平移和旋转参数来定义。或者可以用方程、运动学参数或动力学参数描述运动。运动学用位置、速度和加速度来描述运动。动力学利用作用在场景中对象上的力来描述运动。

关节链形体常用于为人和动物的运动建模。通过旋转节点连接的刚性连杆定义为一个层次式结构。当对象进行一个运动时，每个部分按特定的路线移动。移动路线根据对象的整体运动编程确定。

为了正确显示动画中的周期性运动，采样率必须保证在每个周期内产生足够的帧。否则，将会产生不稳定或误导的动画效果。

除了光栅操作和颜色表方法，OpenGL Utility Toolkit（GLUT）中也提供了一些开发动画程序的函数，包括双缓存操作和空闲时刻的增量运动参数。表 13.1 列出在 OpenGL 程序中产生动画的 GLUT 函数。

表 13.1　OpenGL 动画函数总结

函数	描述
glutInitDisplayMode (GLUT_DOUBLE)	激活双缓存操作
glutSwapBuffers	互换前后刷新缓存
glGetBooleanv (GL_DOUBLEBUFFER, status)	查询系统是否支持双缓存
glutIdleFunc	指定用于增量动画参数的函数

参考文献

关于计算机动画系统的更多的信息请参见Thalmann and Thalmann(1985)、Watt and Watt(1992)、O' Rourke(1998)、Maestri(1999 and 2002)、Kerlow(2000)、Gooch and Gooch(2001)、Parent(2002)、Pocock and Rosebush(2002)和Strothotte and Schlechtweg(2002)。传统动画技术请参见Lasseter(1987)、Thomas, Johnston, and Johnston(1995)和Thomas and Lefkon(1997)。变形技术在Hughes(1992)、Kent, Carlson, and Parent(1992)以及Sederberg and Greenwood(1992)和Gomes, Darsa, Costa, and Velho(1999)中给出。

动画应用算法在Glassner(1990)、Arvo(1991)、Kirk(1992)、Gascuel(1993)、Snyder, Woodbury, Fleischer, Currin, and Barr(1993)和Paeth(1995)中给出。关于OpenGL动画技术的讨论请参见Woo, Neider, Davis, and Shreiner(1999)。

练习题

13.1　为一个如图 13.19 所示的简单棍状形体的动画设计一个情节板及其关键帧。

13.2　使用线性插值法编写一个程序，生成练习题 13.1 中指定的关键帧的插值帧。

13.3　将练习题 13.1 中的动画序列扩充到两个或多个运动对象。

13.4　使用线性插值法编写一个程序，生成练习题 13.3 中的关键帧插值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

13.5 编写一个变形程序，用五个插值帧将任意给定多边形变换成另一个指定的多边形。

13.6 编写一个变形程序，用五个插值帧将一个球变换成一个指定的多面体。

13.7 建立一个引入加速度并实现等式（13.7）的动画描述。

13.8 建立一个引入加速度和减速度并实现等式（13.7）和等式（13.8）中给出的插值间隔计算的动画描述。

13.9 建立一个实现等式（13.9）中的加速度－减速度计算的动画描述。

13.10 编写一个程序，模拟在给定矩形区域中的一个填充圆的二维线性运动。给定圆的初始位置，且该圆碰到墙时按反射角等于入射角的原则弹出。

13.11 通过使用能沿着矩形边来回移动的短线段代替矩形一边，将练习题13.10的程序转换成一个板球游戏。交互控制短线段的移动来模拟用板来阻止圆离开矩形内部。当圆离开矩形内部时游戏结束。初始输入参数包括圆的位置、方向和速度。游戏得分应包括圆被板接挡的次数。

13.12 修改练习题13.11使球速可变。经过一定间隔，例如反弹5次后，就可以增加球速。

13.13 将练习题13.10扩充成模拟一个球在平行管道中进行三维运动。可以交互设定观察参数，以从不同方向观察运动。

13.14 编写一个程序，使用等式（13.10）实现弹球的运动模拟。

13.15 扩展上一练习题中的程序，使之包含挤压和拉伸效果。

13.16 编写一个程序，用动力学实现弹球的运动。弹球的运动受向下重力和地面摩擦力的支配。启动时，该球按一给定速度向量发射到空间中。

13.17 编写一实现动态运动描述的程序。描述有2到3个对象的场景、初始运动参数及指定力，然后从力的方程解中生成动画（例如，对象可以是地球、月亮和太阳，其引力与质量成正比，与距离平方成反比）。

13.18 修改旋转的六边形程序，允许用户从菜单项交互选择一个要旋转的对象。

13.19 修改旋转的六边形程序，使其绕椭圆形路径旋转。

13.20 修改旋转的六边形程序，允许交换改变旋转速度。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第14章 层次建模

定义复杂对象或系统的最容易的方法是先指定各个局部，然后描述如何拼装这些局部成为完整的对象或系统。例如，一部自行车可以用框架、轮子、挡板、手把、座位、链条、踏板以及定位这些部件形成自行车的规则来描述。可以用树结构给出这种层次式的描述，其中局部作为树节点，而构造规则作为树枝。

在建筑和工程系统中，建筑布局、汽车设计、电子电路设计和家电设计普遍使用计算机辅助设计方法。图形设计方法也可以用来表示财政、经济、组织、科学、社会和环境系统。我们经常创建仿真系统来研究系统在不同条件下的行为。仿真的结果可以作为指导性的工具或作为系统决策的基础。图形设计软件包通常提供创建和管理层次建模的子程序。有些软件包还包含预定义的形状，例如轮子、门、齿轮、轴和电子电路部件。

14.1 基本建模概念

创立和管理一个系统的表示称为**建模**（modeling）。系统的一种表示称为系统的一个**模型**（model）。系统的模型可以使用图形定义，也可以完全是描述性的定义，例如，系统参数之间的关系可以使用一组方程来定义。图形模型又称为**几何模型**（geometric model），因为系统的组成部分可以使用几何对象（如直线段、多边形、多面体、圆柱或球）来表示。这里我们仅仅考虑图形应用，因此使用模型这个术语来说明一个系统的计算机几何表示。

14.1.1 系统表示

图 14.1 是一个逻辑电路的图形表示，显示了许多系统模型的共同特征。系统的各个组成部分使用称为**符号**（symbol）的几何结构进行显示。符号之间的关系使用连线构成的网络来表示。这里使用了三个标准符号来表示布尔运算中的逻辑操作：与、或、非。连线则定义了系统的组成部分之间的输入输出流关系（从左向右）。与门的符号显示在逻辑电路的两个不同位置上。在建立复杂的模型时，常常对一些基本符号重复放置。模型中，符号的每一次出现称为该符号的一个**实例**（instance）。在图 14.1 中，“或”和“非”的符号各有一个实例，而“与”的符号有两个实例。

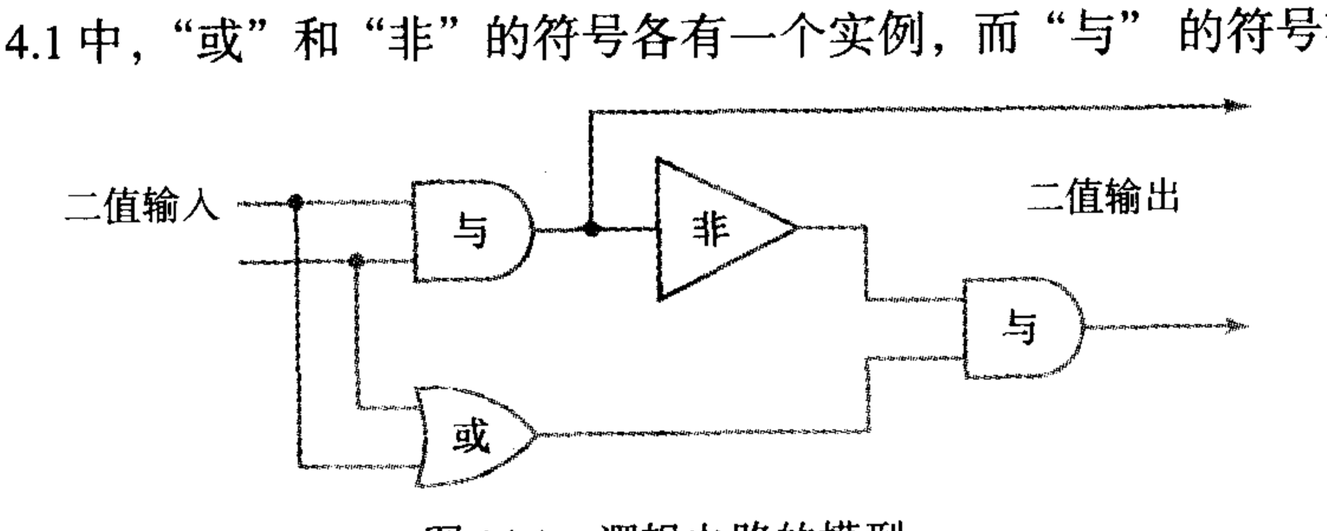


图 14.1 逻辑电路的模型

在很多情况下，表示系统各个部分的特定图形符号由系统描述进行指定。电路模型使用标准电路符号或者逻辑符号。对于表示抽象概念的模型，如政治、经济、财政系统，可以采用任何方便的几何图案作为符号。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

描述模型的信息通常是几何数据和非几何数据的组合。几何信息包括定位部件的坐标值、输出图元和定义部件结构的属性函数，以及构造部件之间联系的数据。非几何信息包括文字标识、模型操作的算法描述和判别部件之间的关系和连接的规则（如果这些信息没有使用几何数据描述）。

指定构造和操作模型所需信息的方法有两种。一种方法是将信息存储在类似列表和链表的数据结构中。另一种方法是在过程中描述信息。通常，模型描述包括数据结构和过程，当然有些模型完全使用数据结构定义而其他一些模型仅使用过程定义。物体的实体造型中大部分的信息是从定义坐标位置的数据结构中提取出来的，而很少使用过程。另一方面，对于天气模型，主要使用过程来计算温度和压强变化图。

作为一个说明如何组合使用数据结构和过程的例子，考虑如图 14.1 所示的逻辑电路的几种模型描述。一种方法是定义数据表中的逻辑部件（如表 14.1 所示），说明如何进行网络连接和电路操作的处理过程。在该表中的几何数据包括坐标值，以及绘制和定位门电路的必要参数。这些符号可以是多边形，或是直线段、椭圆弧的合成。各个部件的标识也包含在表中，而在显示为大家所公认的符号时也可以省略标识。然后，按照门电路的坐标位置和连接门电路的顺序，使用一个过程来显示门电路和构造连接线。再使用另外的一个过程，在给定输入时产生电路输出（二进制值）。该过程可以设置为仅仅显示最后的输出，也可以设置为显示中间值以说明电路的内部功能。

另一方面，可以在一个数据结构中指定电路模型的图形信息。在列出电路的每条线段端点的数据表中定义了连接线和门电路。然后使用一个简单的过程显示电路和计算输出。作为另一种极端情况，可以完全在过程中定义模型而不使用外部的数据结构。

表 14.1　定义图 14.1 电路中各个门的结构和位置的数据表

符号码	几何描述	标识符
Gate 1	（坐标和其他参数）	and
Gate 2	…	or
Gate 3	…	not
Gate 4	…	and

14.1.2　符号层次

很多模型可以使用符号的层次结构来组织。模型的基本元素可以使用适合于构造这种模型的简单几何形状进行定义。这些简单符号可用来形成称为**模块**（module）的组合对象。模块又可以形成更高层次的模块，从而形成模型的各个部分。最简单的情况中，可以使用一层部件来描述模型，如图 14.2 所示。在电路的例子中，假设将门电路定位，并按照门电路描述中的连接规则对门进行连接。在这个层次描述中的基本符号是逻辑门。虽然门本身也可以描述为直线、椭圆弧和文本形成的层次结构，但是这种描述不能方便地构造逻辑电路，因为逻辑电路中最简单的构造块为门。而在设计不同几何形状的应用中，基本符号可以定义为直线段和弧。

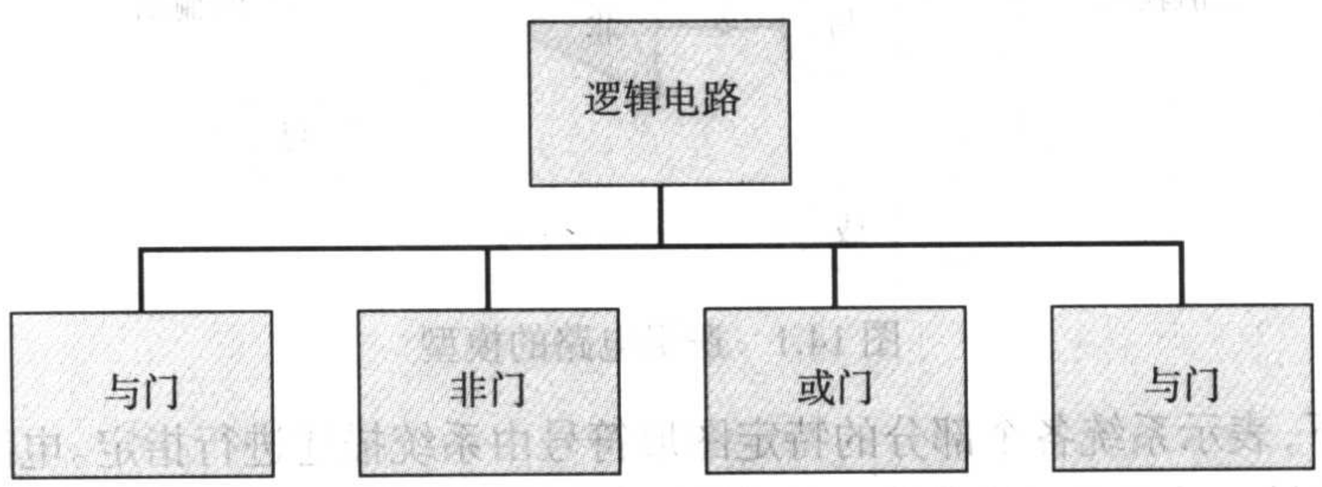


图 14.2　使用逻辑门构成的电路的单层次描述

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

图 14.3 给出了两层符号的层次。设备的布局就是工作区的安排。每个工作区都配置了一套家具。基本的符号是各类家具：办公桌、椅子、书架、文件柜等。更高层的对象是配有不同家具结构的工作区。基本符号的引用由工作区中该符号的尺寸、位置和方向进行定义。位置由工作区中的坐标给出，方向则根据符号面向方向的旋转来指定。第二层的各个工作区由设备布局中具体的尺寸、位置和方向进行定义。每个工作区的边界配有分割栏，而且提供了设备之间的通道。

通过在每个更高层次上重复对符号簇进行组合，可以形成更复杂的符号层次。图 14.3 中的布局可以扩展到包含符号簇以形成一个大楼的不同房间、不同楼层、一个楼群的不同大楼以及不同地点的不同楼群。

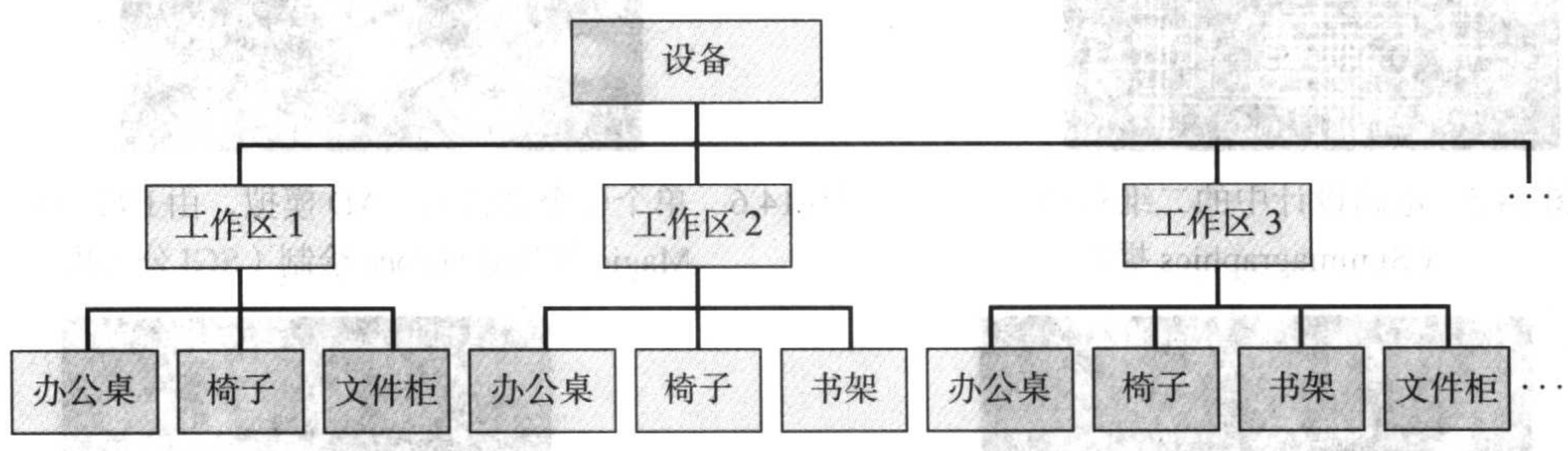


图 14.3　设备布局的两层次描述

14.2　建模软件包

虽然通用的图形系统可以用来设计和操纵系统模型，对于特殊应用的建模使用专用建模系统则更为方便。建模系统则根据符号层次提供了定义和重新调整模型表示的方法，图形程序处理符号的层次是为了显示。通用图形系统并不是针对广泛的建模应用进行设计，而有些图形软件包（如 PHIGS 和 GL）将建模功能和图形功能集成在一个软件包内。图 14.4 给出一个 PHIGS 结构层次的例子。该例子由曼彻斯特大学使用 PHIGS 工具包软件开发，为 PHIGS 应用程序提供编辑窗口、菜单和其他界面工具。

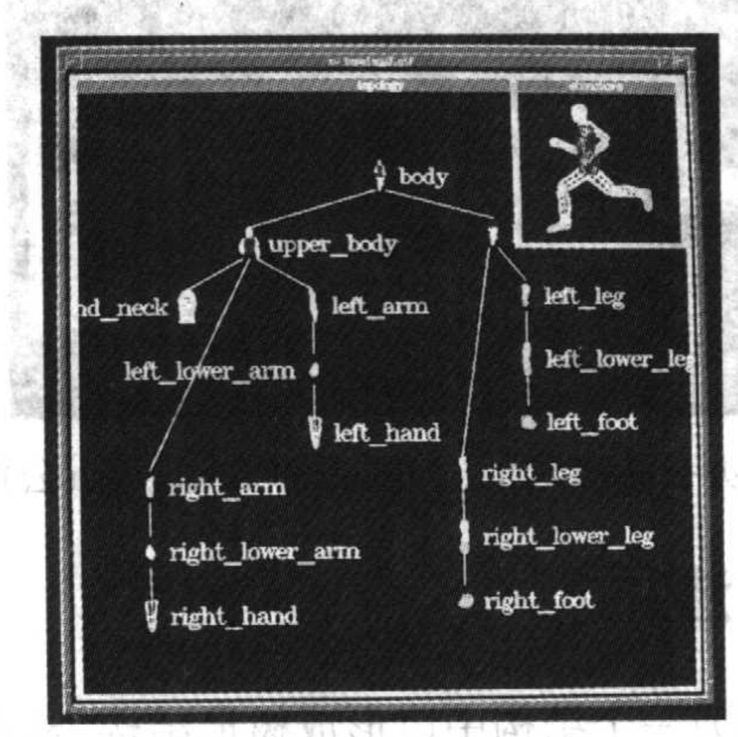


图 14.4　使用 PHIGS 工具包表示物体的层次，由曼彻斯特大学研制。图示物体树本身是一个 PHIGS 结构（曼彻斯特大学计算机科学系的 T. L. J. Howard、J. G. Williams 和 W. T. Hewitt 提供）

如果图形库不包含建模功能，可以用建模软件包与图形子程序建立接口。或者，可以使用图形库中提供的几何变换和其他功能创建自己的建模子程序。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

专用建模软件包（如某些CAD系统）按照应用的类型进行定义和构造。软件包为针对的应用提供符号形状的菜单和函数，并且可以支持二维或三维建模。建模软件包可以设计为二维或者三维的。图14.5给出了由CAD软件包生成的电路设计中的二维布局。图14.6展示了三维CAD应用。图14.7和图14.8给出了三维分子建模的例子。图14.9显示了设备布局的三维模型。

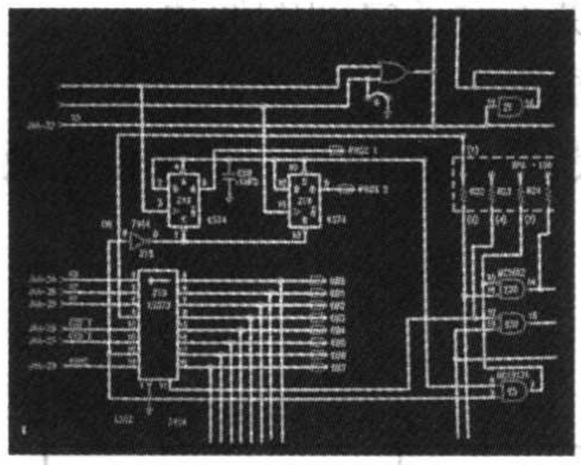


图14.5 电路设计中的二维布局（Summagraphics提供）

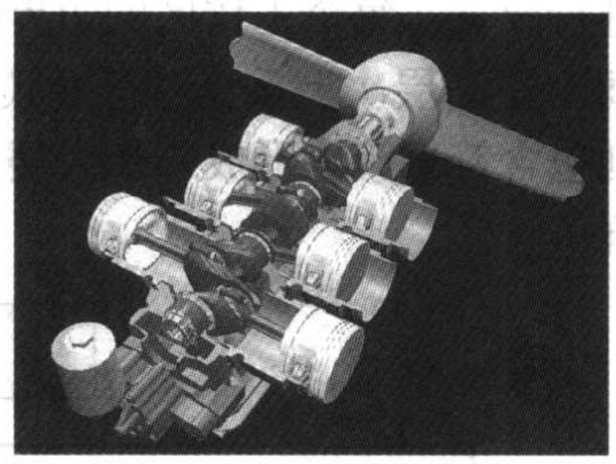


图14.6 单个引擎部件的CAD模拟，由FTI/3D-Magic的Ted Malone绘制（SGI公司提供）

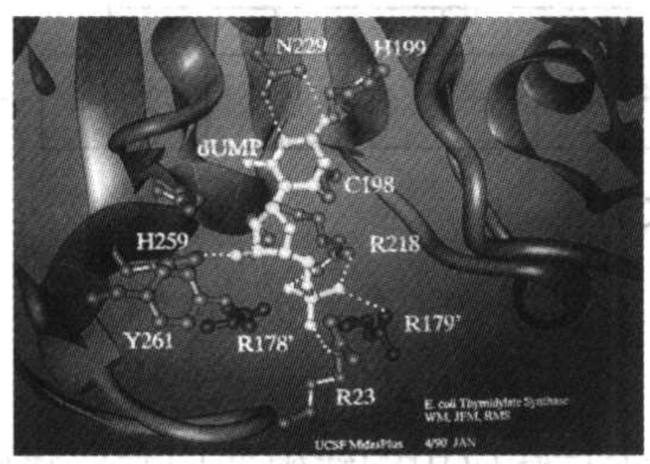


图14.7 核氨基酸余渣与Thymidylate Synthase的自然酶作用物相互作用的圆球和枝条表示，由UCSF计算机图形实验室的Julie Newdoll建模并绘制（SGI公司提供）

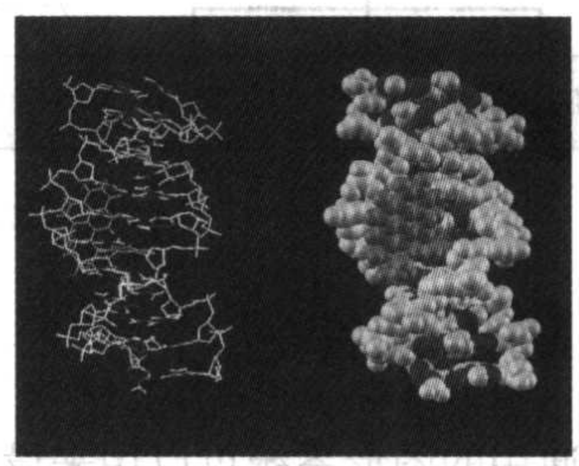


图14.8 表示DNA的三维分子模型的一对立体图像。Rutgers大学的Tamar Schlick、NYU和Wilma K. Olson提供数据，SDSC的Jerry Greenberg将其可视化（圣迭戈超级计算机中心的Stephanie Sides提供）

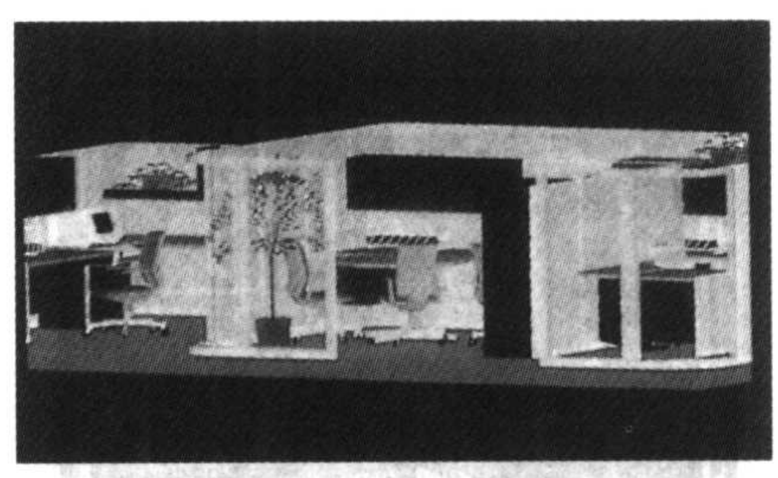


图14.9 办公室布局的三维视图（Intergraph公司提供）

14.3 通用层次建模方法

可以通过将一个结构嵌套到另一个结构中以形成树形结构，从而创建系统的层次式模型。在将每个结构放进层次中时，将进行适当的变换以适合整体模型。例如，先将家具安排在各个办公室和工作区内，然后办公室和工作区可以组成不同的部门，这样由下而上形成层次。图14.10给出了多个坐标系和三维物体分层建模的一个应用实例。此图表示了拖拉机运动的模拟。当拖拉机运动时，拖拉机坐标系和前轮坐标系在世界坐标系中移动。当拖拉机转弯时，前轮在车轮系统中旋转，而车轮系统在拖拉机系统中旋转。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

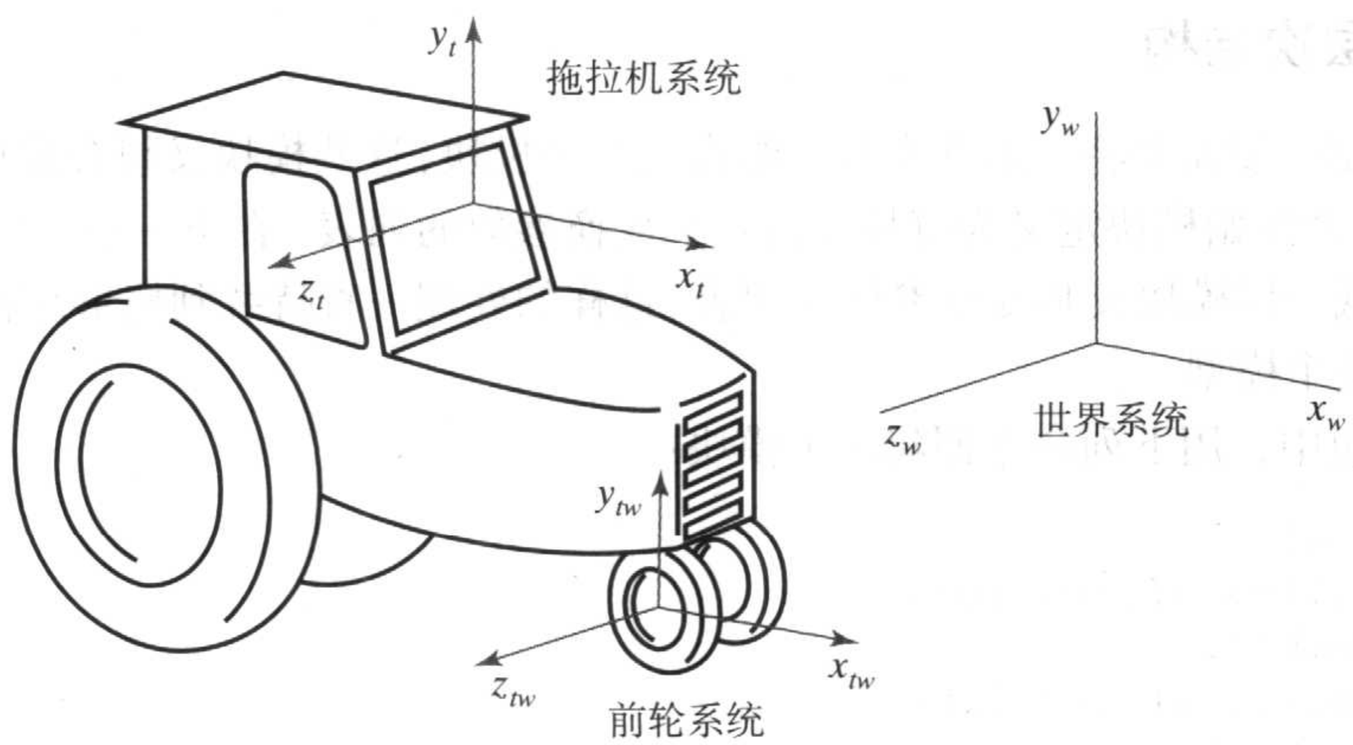


图 14.10　模拟拖拉机运动可能使用的坐标系统。前轮系统的旋转导致拖拉机转弯。车轮和拖拉机参照系都在世界坐标系中运动

14.3.1　局部坐标

在许多设计应用中，通过基本符号集中的几何形状实例（变换后的副本）来构造模型。实例按适当的方向放置在模型的世界坐标参照系中。应用中使用的不同图形符号分别关于世界坐标参照系独立定义，该坐标系称为局部坐标系（local coordinate system）。局部坐标也称为建模坐标（modeling coordinate）或主坐标（master coordinate）。图 14.11 给出了两个符号（用于二维设备的布局应用）的局部坐标定义。

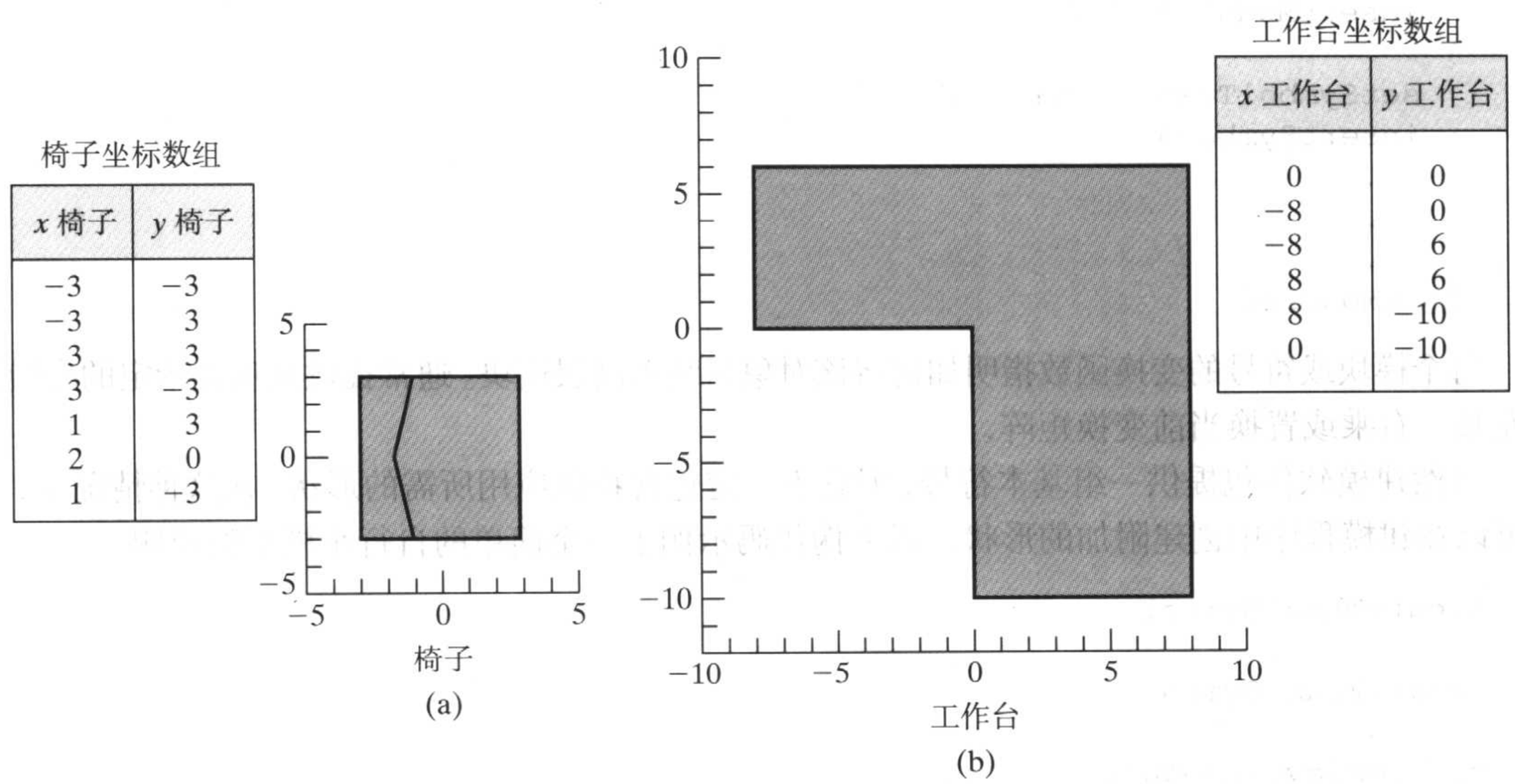


图 14.11　局部坐标系中定义的对象

14.3.2　建模变换

如果要构造图形模型的各个部分，可以对符号的局部坐标进行变换，从而生成世界坐标系中符号的实例。作用于符号的建模坐标定义的变换称为建模变换（modeling transformation），建模变换给出符号在模型中的具体位置和方向。建模软件包中典型的建模变换包括平移、旋转和缩放，但是在某些应用中还有其他的变换。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

14.3.3 创建层次结构

层次建模的第一步需要将一组基本符号组合成一个模块，这些模块又组合成更高层次的模块，等等。首先，将一个初始模块定义为符号实例及其变换参数的列表。在下一层，将高一层的模块定义为符号实例和低一层模块及其变换参数的列表。这样的处理一直持续到树形结构的根，根代表了世界坐标系中的整个模型。

在建模软件包中，用下列命令创建一个模块：

```
createModule1
   setSymbolTransformation1
   insertSymbol1
   setSymbolTransformation2
   insertSymbol2
      .
      .
      .
closeModule1
```

为构成模块，每个基本符号的实例被赋予一组变换参数。同样，以下函数可以将模块组合成更高层的模块：

```
createModule6
   setModuleTransformation1
   insertModule1
   setModuleTransformation2
   insertModule2

   setSymbolTransformation5
   insertSymbol5
      .
      .
      .
closeModule6
```

每个模块或符号的变换函数指明如何将该对象组装入高层模块。通常提供选项将指定的变换矩阵左乘、右乘或置换当前变换矩阵。

虽然建模软件包提供一组基本符号，但它不一定包含特殊应用所需的形状。在这种情况下，我们可以在建模程序中创建附加的形状。以下伪代码示例了一个简单的自行车模型的说明。

```
createWheelSymbol

createFrameSymbol

createBicycleModule
   setFrameTransformation
   insertFrameSymbol
   setFrontWheelTransformation
   insertWheelSymbol

   setBackWheelTransformation
   insertWheelSymbol
closeBicycleModule
   .
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

层次建模系统通常还包含许多其他建模子程序。可以有选择地显示模块或者暂时将模块从系统表示中除去。这些功能使设计者可以用不同的形状和结构进行试验。在设计过程中还可以突出显示和移动选中的模块。

14.4　使用 OpenGL 显示列表的层次建模

我们可以用OpenGL的嵌套显示列表形成层次模型来描述复杂对象。模型的每个符号和模块用glNewList函数创建。在高阶显示列表的定义中可以用glCallList函数将一个显示列表插入到另一个显示列表中。与每个插入对象关联的几何变换参数指定了该对象在上一层模块中的位置、方向和尺寸。例如，下列代码用来描述一部仅由框架和两个相同的轮子组成的自行车。

```
glNewList (bicycle, GL_COMPILE);
   glCallList (frame);

   glTranslatef (tx1, ty1, tz1);
   glCallList (wheel);

   glTranslatef (tx2, ty2, tz2);
   glCallList (wheel);
glEndList ( );
```

类似地，框架的显示列表自身还可以包括描述手把、链条、踏板和其他部件的显示列表。

14.5　小结

计算机图形应用中的术语模型是指某系统的图形表示。系统中的部件使用局部坐标参照系（也称为建模坐标系或主坐标系）中的符号表示。很多模型（例如电路）可以通过在选定的位置和指定的方向放置符号的实例而构成。

很多模型使用多层符号进行构造。我们可以嵌套模块来构造层次模型，其中模块由符号的实例和其他模块组成，然后继续向下嵌套直到成为仅包括输出图元（和属性）的符号。因为每个符号或模块嵌套在上层模块中，在嵌套结构中必须指定它们的建模变换参数。

使用OpenGL的显示列表也可以设置层次模型。glNewList函数用于定义系统的整体结构及其成员模块。在指定了插入对象的一系列指定位置、方向和尺寸的变换后，用glCallList将各个符号和模块插入上一层模块。

参考文献

使用 OpenGL 进行建模的例子请参见 Woo, Neider, Davis, and Shreiner(1999)。

练习题

14.1　讨论几种不同系统的模型表示和各系统的图形表示。同时讨论对于每个系统如何实现图形表示。

14.2　设计一个二维设备布局软件包。为设计者提供设备形状的菜单，可在一个房间的任何地方放置物体（一个层次）。实例的变换限于平移和旋转。

14.3　扩充上一练习题，使家具的尺寸可以缩放。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

14.4 设计一个有设备形状菜单的二维设备布局软件包。使用两个层次的结构，从而使家具放在不同的工作区中，而工作区可以放置在更大的区域中。实例的变换限于平移和旋转。

14.5 扩充上一练习题，使家具尺寸可以缩放。

14.6 编写一组子程序用于构造和显示逻辑电路中采用的标准逻辑符号。符号集至少包括图 14.1 中的与门、或门和非门。

14.7 开发一个用户在电路网络中定位电子符号的逻辑电路设计软件包。使用上一练习题中的符号集，并且仅使用平移将菜单中形状的实例放置于网络中。一旦将组件放入网络，就使用直线段与其他指定元件相连。

14.8 编写一组子程序用于编辑已经在应用程序中创建的模块。这些子程序必须提供下列编辑操作：追加、插入、置换和删除模块元素。

14.9 给定一个模型中所有显示对象的坐标范围，编写一个子程序删除任意选定的对象。

14.10 编写一个程序来显示和删除一个模型中的指定模块。

14.11 编写一个子程序，可以有选择地将模块从一个模型的显示中除去或放回。

14.12 编写一个程序来用某种方式突出显示选定的模块。例如，选定模块可以用不同的颜色显示，或者用一个矩形框包围选定模块。

14.13 编写一个程序，用闪烁方式突出显示一个模型中选定的模块。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

第15章　图形文件格式

任何存储的图画的表示称为**图形文件**（graphics file）或**图像文件**（image file）。在光栅图形系统中，彩色屏幕显示在帧缓存中表现为一组RGB像素值。如2.1节所述，帧缓存（或其中的矩形区域）中的内容称为像素图（pixmap）。虽然二值图像可以用位图（bitmap，每个像素一个比特位）形式存储，但是大多数光栅图像用像素图存储。通常，任何图画的光栅表示都可以称做**光栅文件**（raster file）。已经存在多种文件格式，可以用各种各样的方式来组织图像文件中的信息。因为全彩色的光栅文件非常大，大多数文件格式采用某种压缩方法来减小存储和传输的文件尺寸。此外，当在颜色显示能力有限的系统中显示图像时，或是用不支持24位颜色深度的文件格式存储图像时，必须减少全彩色图像文件中的颜色数量。本章对图形文件格式和常用的减小图像文件尺寸和减少待显示图像颜色数量的方法做简要介绍。

15.1　图像文件结构

光栅图像文件中的像素颜色值通常用非负整数存储。颜色值的取值范围取决于每个像素可用的比特位数。对于全彩色（每个像素24位）RGB图像，每个颜色分量的值用一个字节存储。R、G和B的取值范围是0～255。使用RGB颜色表示、未压缩的光栅图像文件也称为**原始数据**（raw data）或**原始光栅文件**（raw raster file）。压缩图像文件格式使用其他颜色模型，包括HSV、HSB和YC_rC_b。每个像素可用的比特位数取决于文件格式。

文件格式通常包括一个说明文件结构信息的**文件头**（header）。压缩文件的文件头还包含用于解码和显示压缩图像的表。文件头包含的信息有：文件尺寸（水平扫描线的数量和每条扫描线上的像素数），分配给每个像素的比特数或像素数，用于减小文件尺寸的压缩方法，像素值的颜色取值范围，以及图像的背景颜色。

光栅图像文件的另一个特征是文件中的字节排序。大多数计算机处理器按最重要字节在前的顺序存储多字节整数，用术语“**高位在前**”（big endian）表示。而有些处理器按最不重要字节在前的顺序，用术语“**低位在前**”（little endian）表示。

有些文件格式采用**几何表示**（geometric representation），即通过一列坐标位置和描述直线段、填充区域、圆弧、样条曲线和其他图形元素的信息来存储图画。几何表示也包含属性信息和观察参数。这种图像表示通常称为向量格式（vector format），即使并非所有几何结构都用直线段表示。术语“向量”文件原来是用来描述一组在向量系统（随机扫描系统）上显示的直线段。虽然向量系统已经被光栅系统取代，并且向量文件中也加入了非线性对象，但是对于使用图的几何表示的文件仍然沿用“向量格式”的名称。既支持几何表示又支持光栅表示的文件格式称为**混合格式**（hybrid format）或**元文件**（metafile）。

科学可视化应用通常使用由一组数据形成的图像文件，数据通过仪器测量或者计算机数值模拟而得到。特定数据的可视化可以采用多种方法，包括伪彩色显示、等值面表示或体绘制。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

15.2 降色方法

有多种方法可以减少显示图像时所使用的颜色数量。最常用的方法是产生能够较好地近似原始颜色的颜色采样。

有时降色方法也称为“量化”。量化是物理和数学领域使用的术语，例如在量子力学和采样理论中，用于描述从一个连续的分布中产生一组离散的值的过程。然而，光栅图像文件包含一组有限、离散的颜色值，而不是一个连续分布。因而，降色方法只是用一组更少的离散颜色来替代原先的一组离散颜色。而且，常用的降色方法产生的颜色并不是某个选定颜色值的倍数，也就是说，降色并不产生量化了的颜色。

15.2.1 均匀降色法

减少光栅文件中颜色数量的简单方法是分别将 R、G 和 B 分量除以一个整数然后取整。例如除以 2，就可以将全彩色表示的 R、G 和 B 分量减少为 128 级。如图 15.1 所示，均匀降色方法用一个颜色级来取代原先的一组相邻的颜色级。

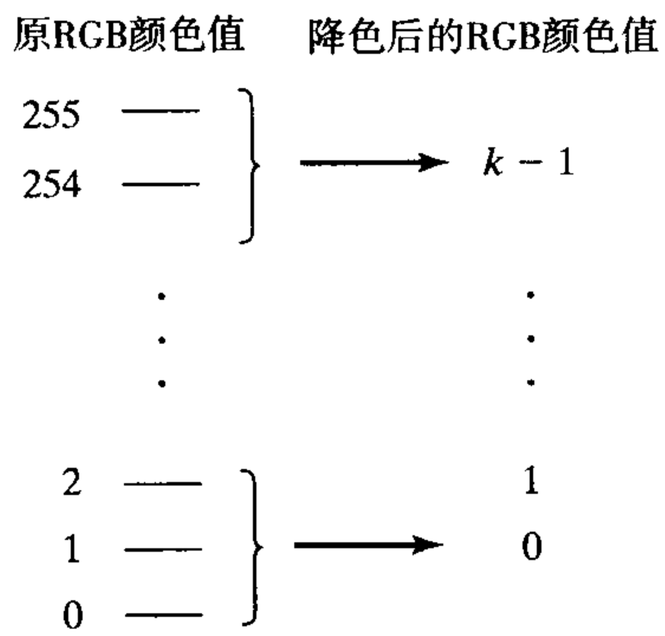


图 15.1 将全彩色图像中的 RGB 值均匀降色到 k 级

另一种方法是用一组像素值的中值代替这组像素值，或者用一组颜色值中的平均值代替这组像素值。

通常我们可以认为在一个图像文件中，RGB 各分量的所有 256 个值不会全部出现。因此，可以在文件中出现的颜色值的最大、最小值直接进行均匀量化。

也可以对不同的 RGB 分量使用不同的降色规则。例如，降色一幅全彩色图像时，我们可以用 3 个比特位（8 级）来表示红色和绿色分量，而用 2 个比特位（4 级）来表示蓝色分量。

15.2.2 基于出现频率的降色法

另一种降色方法只保留在图像中出现最频繁的那些颜色值。首先，减少图像中表示 RGB 颜色分量的比特位数，然后扫描更改后的颜色集合，产生每个 RGB 颜色分量值的出现频率的计数（或直方图）。如果要产生包含 k 种颜色的彩色文件，我们就选择在图像中出现最频繁的 k 种颜色。

15.2.3 中值切割降色法

这种方法将图像中的颜色空间分为 k 个子区域，并且计算每个子区域的平均值。生成子区域的算法如下：首先，确定图像中 RGB 颜色分量的最大、最小值，记为 R_{min}、R_{max}、G_{min}、G_{max}、B_{min} 和 B_{max}。这六个值确定了在RGB颜色立方体中该图像使用的颜色块的边界。对于三个颜色分量中间隔最大的一个，将位于此间隔中的颜色值按递增或递减排序，然后根据位于序列中间位置的颜色值将

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

当前颜色块分为两个更小的颜色块。如图 15.2 所示，红色分量的范围最大，计算红色范围的中值（记为 R_{median}）使一半像素的红色分量大于它，另一半小于它。然后，在 R_{median} 处将图像的颜色块切割成两个子块。每个子块再用同样的方法进行处理，直到将原始图像的颜色块分为 k 个子块。每一步处理最大的子块。按要求的精度计算每个子块的平均颜色，最后用子块的平均色替换所有位于子块内的像素颜色。

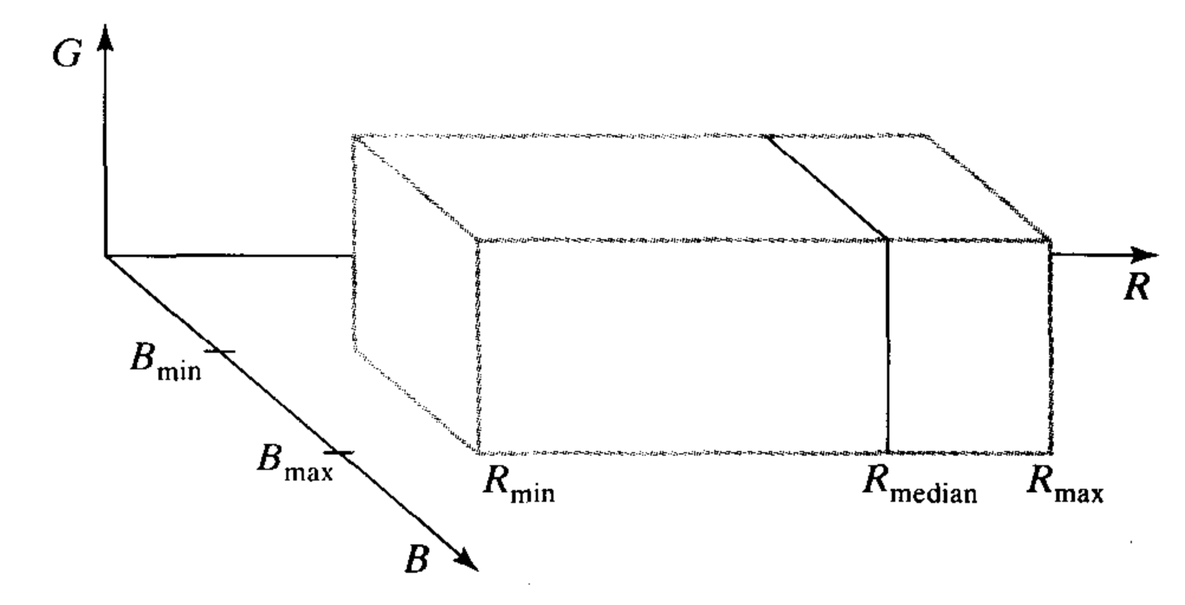


图 15.2　在红色分量的中值处对一个图像颜色块进行切割

15.3　文件压缩技术

有多种压缩技术可以减少图像文件的字节数，但特定压缩方法的效果取决于图像的类型。在图像文件中寻找模式的简单方法最适用于包含大范围单色区域的几何设计，而更复杂的压缩方案处理真实感计算机图像和数字照片的效果更好。减小图形文件尺寸的一般技术使用占据更少字节的编码来代替颜色值。此外，压缩文件中还包含用于指明诸如扫描线末尾或图像文件末尾的代码。

有些压缩算法使用浮点运算，这会引入取整误差。使用拟合的方法也会改变图像的颜色。因此，从压缩文件中解码产生的文件通常包含与原始图像不完全一致的颜色值。例如，原始输入图像中的整数 RGB 颜色值(247, 108, 175)在压缩和解码后可能会变成(242, 111, 177)。但是这些颜色变化是可以容忍的，因为人眼对微小的颜色差别并不敏感。

不改变图像文件的值的压缩方法称为**无损压缩**（lossless compression）方法，而产生颜色变化的方法称为**有损压缩**（lossy compression）方法。大多数情况下，有损压缩方法可以产生大得多的压缩比（原始文件的字节数与压缩文件的字节数的比值）。

15.3.1　行程编码

此方法在图像文件中搜索连续、重复的值，然后将一串重复的值存储为重复的次数和该值，从而减小文件尺寸。例如，如果 125 在一条扫描线上连续重复 8 次，在压缩文件中存储两个值：8 和 125。这将原先需要 8 个字节存储减为仅需两个字节。对于包含大范围单色区域的图像，这种编码方法非常有效。但是在如数字照片等具有频繁颜色变换、很少连续重复值的图像中，很多颜色值的重复次数只有 1。

有多种对基本行程编码方法的改进方法可以提高编码效率。例如，可以用负数来指出一串不重复的值，而不再为其存储重复次数 1。因此，以下序列：

```
{20, 20, 20, 20, 99, 68, 31, 40, 40, 40, 40, 40, 40, 40, 40, ...}
```

编码为

```
{4, 20, -3, 99, 68, 31, 8, 40, ...}
```

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

上述编码表示20出现4次，然后是3个不重复的值99、68和31，接着是40出现8次。此例中，原先15个字节的输入被压缩为8个字节。

15.3.2 LZW编码

由Lempel、Ziv和Welch开发的LZW算法是早期的模式识别算法LZ、LZ77和LZ78的改进。LZW算法用一个代码来替换图像文件中的重复模式。例如，在下列12个值的序列中模式{128, 96}和{200, 30, 10}各出现两次：

```
{128, 96, 200, 30, 10, 128, 96, 50, 240, 200, 30, 10, ...}
```

如果用代码$c1$和$c2$分别代替这两个模式，并用代码$c3$代替剩下的模式{50, 240}，就可以将原来12个字节的输入编码为5个字节：

```
{c1, c2, c1, c3, c2, ...}
```

或者，不重复的模式，如{50, 240}，可以不用代码而直接存储在压缩文件中。

LZW算法搜索重复序列，并构造一张这些序列与其对应的代码的表，根据码表解码压缩文件。因此，这种编码方法称为替代算法（substitutional algorithm）和基于字典的算法（dictionary-based algorithm）。

15.3.3 其他模式识别压缩方法

我们可以使用模式识别方法在图像文件中查找某种黑白颜色或RGB颜色组合的重复，也可以检测和编码重复的扫描线或其他模式来进一步减小图像文件的尺寸。此外，自相似的颜色集合还可以使用分形方法进行编码。

15.3.4 霍夫曼编码

霍夫曼（Huffman）方法对图像文件中的值进行变长编码来实现压缩。霍夫曼编码方法为最频繁出现的值分配最短的代码，而为最少出现的值分配最长的代码。

霍夫曼算法的基本思想与摩尔斯（Morse）电码相同，即为字母表中的字母分配不同长度的字符代码。摩尔斯编码方案为高频字母分配一个字符长度的代码，而为最低频率的字母分配四个字符长度的代码。例如，字母‘E’的代码是一点（·），字母‘T’的代码是一横（–），而字母‘Q’的代码是一点三横(––·–)共四个字符。与摩尔斯电码不同，霍夫曼编码为图像中的值分配变长比特代码，而不是字符代码，这样可以得到更大的压缩率。

霍夫曼算法的第一步是统计输入图像文件中的每个值出现的次数。然后，根据频率计数为每个值分配比特代码。一种分配方法是构造一个二叉树，使高频值靠近树根，而最低频的值作为树的叶子。从低频值开始，自低向上创建子树。为每个子树的根分配一个数值标记，该标记等于它的两个子结点的频率计数的和。当树构造完整后，所有的左子树用二进制0标记，所有的右子树用二进制1标记。每个值的比特代码就是通过拼接从树顶到该值所在结点位置的分支路径上的比特标记而形成。

表15.1用一组六个值来示例创建树的一般步骤。这组值表示一个包含21个像素的小图像文件，其中96出现8次，177出现4次，其他4个值出现的次数请参见表15.1。

表15.1 一个小例子文件中的值的频率计数

文件中的值	频率计数
96	8
177	4

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

（续表）

文件中的值	频率计数
141	3
85	3
210	2
43	1
文件中像素总数	21

表中 210 和 43 的频率计数最低，因此从它们开始创建第一棵子树（参见图 15.3）。该子树的根的结点标记等于它的两个后代的频率计数之和：3 = 2 + 1。然后，从活动列表中删除这两个值（210 和 43）。此时，最低的频率计数是 3。刚刚创建的子树的根结点标记也是 3。接着，可以用三个计数为 3 的项中的任意两个形成下一棵子树。图 15.4 显示了使用两个文件中的值创建的子树，并且从活动列表中删除了 141 和 85。如图 15.5 所示，下一棵子树由频率计数为 4 的 177 和根结点标记为 3 的子树创建。从活动列表中删除 177 和标记为 3 的根结点，则列表中最低的两项都是子树。如图 15.6 所示，这两个子树合并生成一棵新的子树。最后，结合 96 和最新的子树就完成了二叉树的创建，如图 15.7 所示。最终的树根结点的标记为 21，等于图像中所有值的频率计数总和。

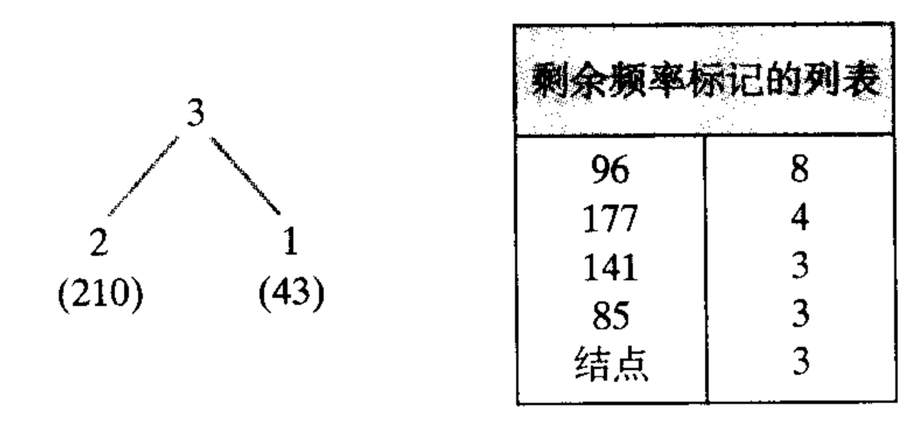


图 15.3　用文件中的值 210 和 43 形成霍夫曼子树

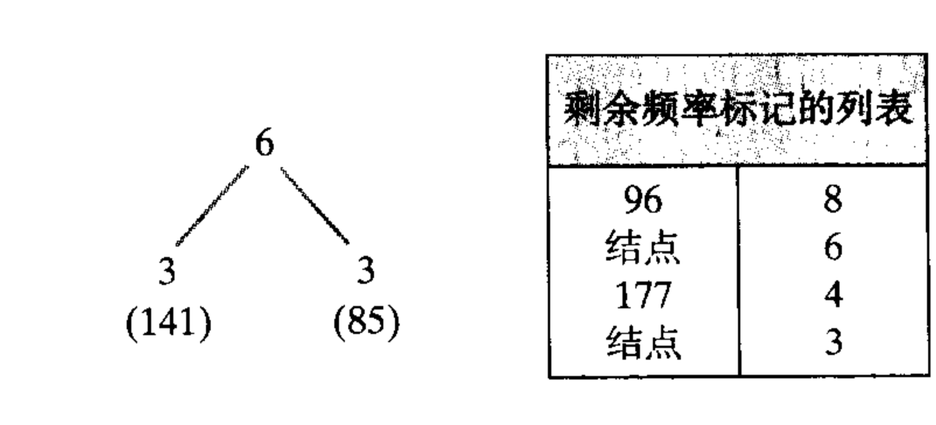


图 15.4　用文件中的值 141 和 85 形成霍夫曼子树

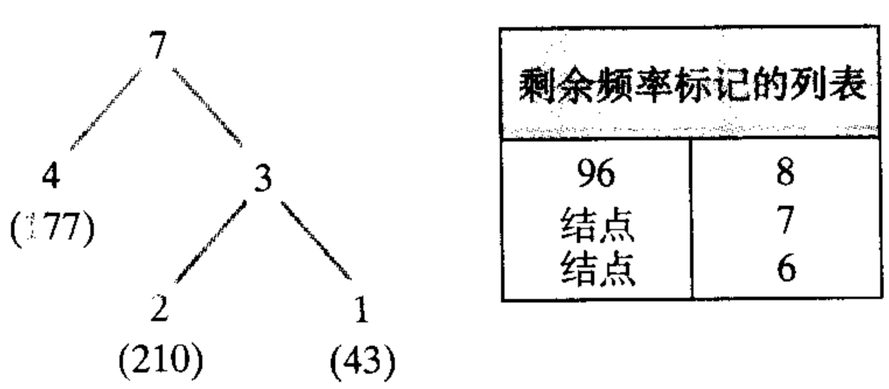


图 15.5　用文件中的值 177 和前面创建的子树形成霍夫曼子树

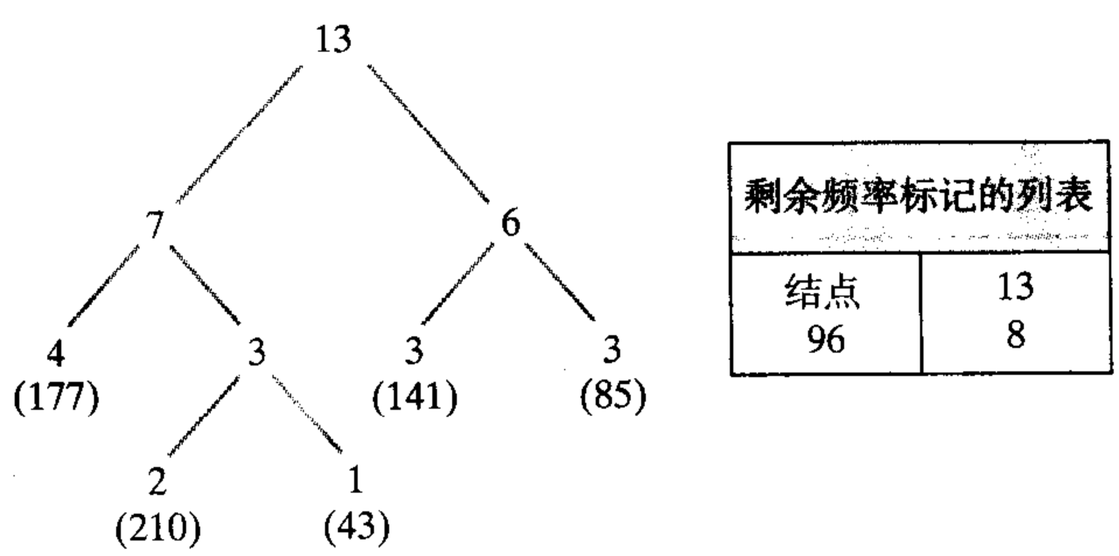


图 15.6　用两棵前面创建的子树形成霍夫曼子树

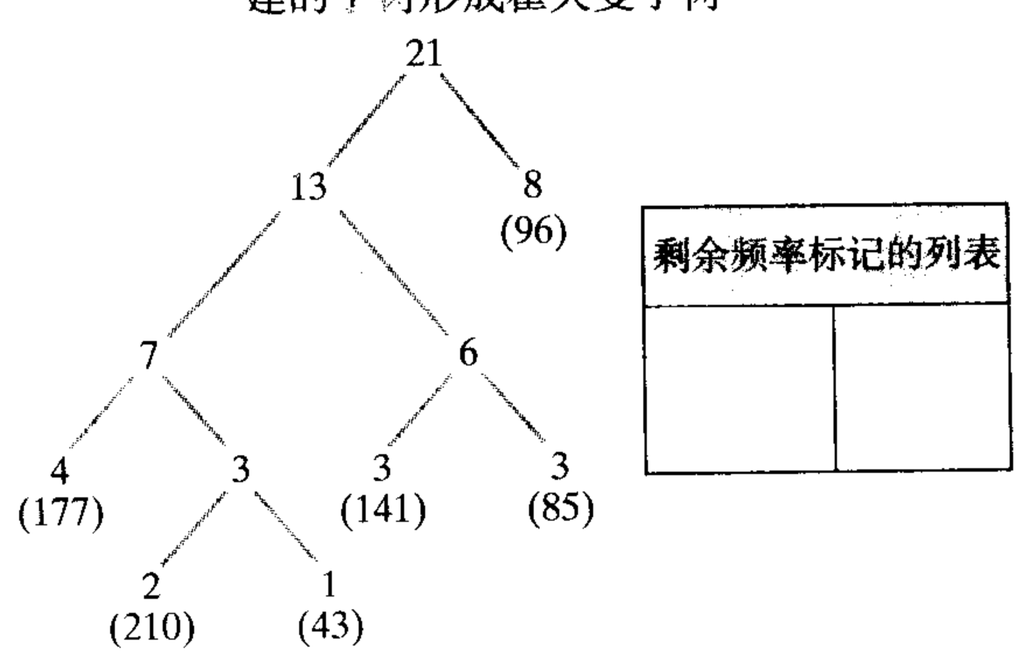


图 15.7　表 15.1 中的值的完整霍夫曼二叉树

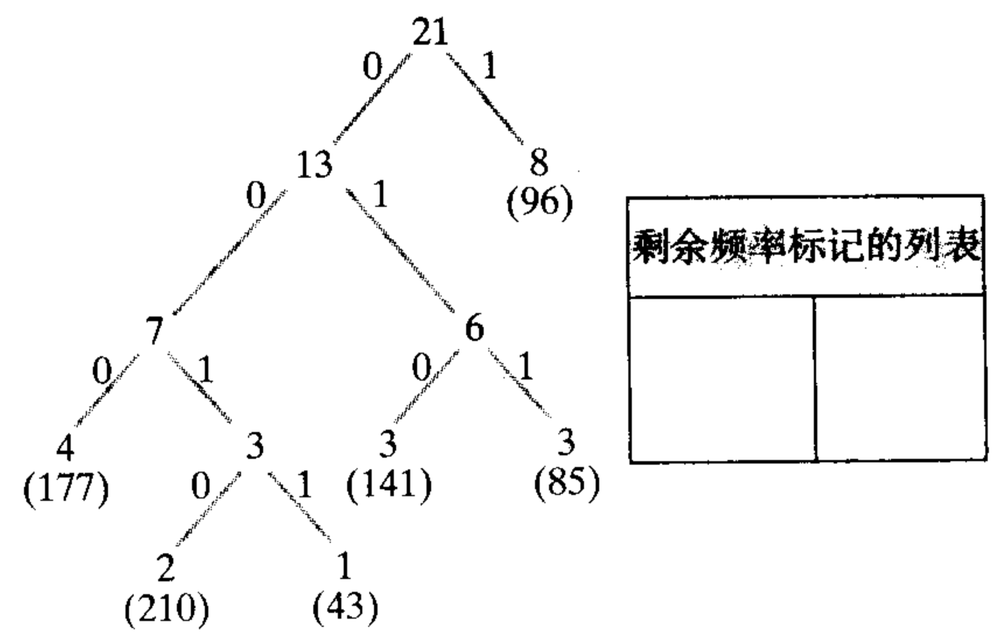


图 15.8　带有分支标记的完整霍夫曼二叉树

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

现在，文件中的所有值都放入了二叉树中。如图15.8所示，将左分支用二进制0标记，右分支用二进制1标记。从树根开始，可以向下拼接分支标记直到叶结点。这形成了每个文件值的变长二进制代码。据此生成的码表（参见表15.2）将存储在压缩文件中。在此例中，有一个值用一位二进制代码，三个值用三位二进制代码，另外两个值用四位二进制代码。低频值的代码较长，而高频值的代码较短。

表15.2 示例文件的霍夫曼编码

索引	文件中的值	二进制代码
1	96	1
2	177	000
3	141	010
4	85	011
5	210	0010
6	43	0011

霍夫曼编码的一个重要特点是没有一个代码是另一个代码的前缀，因此可以利用表15.3中的信息来辅助表15.2进行解码。假设压缩文件包含比特流{100100100...}。第一个比特是1。因为1不可能是任何其他代码的前缀，所以它一定代表原文件中的96。下一个比特是0。因为除了1之外，没有一个比特或者两个比特的代码，因此下一个代码一定是001或者0010。查表15.2可知，值210的代码是0010，因此不可能有其他值的代码是001。至此，两个值96和210已经被解码。下一个代码是010或0100。因为141的代码是010，因此不可能有其他值的代码是0100。这样就解码出第三个值141。如此继续分析比特流直到完全解码。

表15.3 比特代码参考表

比特代码长度	最小代码值	最大代码值	首索引
1	1	1	1
3	000	011	2
4	0010	0011	5

我们也可以使用其他方法产生和分配霍夫曼比特代码。当得到频率计数后，为每个值分配一个代码长度。根据代码长度和频率计数，就可以用列表合并算法来设计比特代码。可以用一组预定义代码为文件中的值分配代码，这样可以避免在压缩文件中存储代码。

15.3.5 算术编码

算术编码方法根据文件中的频率计数产生值序列的数值代码。它首先计算每个值在文件中所占的比例，这在从0.0到1.0的单位区间内产生一组子区间。然后，反复映射每个文件片段到这些子区间上，从而建立不同文件值组合的数值区间。子区间的数值边界就用来编码这些值的组合。

考虑一个具有80个序列但只有3个不同的值的文件。表15.4给出了这三个值对应的频率计数和在文件中的比例。值V_1对应于从0.00到0.20的子区间，值V_2对应于从0.20到0.50的子区间，值V_3对应于从0.50到1.00的子区间。也就是说，20%的单位区间与V_1关联，30%的单位区间与V_2关联，50%的单位区间与V_3关联。如果把V_1映射到V_3的子区间，将占单位区间上半段的20%。这个从0.50到0.60的子区间代表了序列V_3V_1。类似可得将V_2和V_3映射到V_3的结果，如表15.5所示。如此继续将两个值的序列的区间映射到其他区间上，从而得到更长的值序列。子区间的边界值用于编、解码值序列。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

表 15.4　示例文件中值的频率计数和出现比例

文件值	频率计数	出现比例	单位区间上的范围
V_1	16	0.20	0.00 ~ 0.20
V_2	24	0.30	0.20 ~ 0.50
V_3	40	0.50	0.50 ~ 1.00
总计	80	1.00	

表 15.5　以 V_3 开始的所有两个值的序列在单位区间上的范围

比特代码长度	单位区间上的范围
V_3V_1	0.50 ~ 0.60
V_3V_2	0.60 ~ 0.75
V_3V_3	0.75 ~ 1.00

有多种方法可以终止单位区间的细分，并且为值的组合分配数值代码。算术编码方法通常用二进制数实现，而不是用单位区间内的浮点值。因此，压缩文件是一个二进制数的序列。

15.3.6　离散余弦变换

文件压缩方法中应用了包括傅里叶（Fourier）变换和哈达玛（Hadamard）变换在内的许多数值变换方法，其中离散余弦变换（Discrete Cosine Transform，DCT）是最常用的。DCT 的高效实现算法可以更快、更逼真地重现用高压缩比进行压缩的图像。

对于一列 n 个数值 V_k（$k = 0, 1, ..., n-1$），DCT 产生下列一组变换值：

$$V_j' = c_j \sum_{k=0}^{n-1} V_k \cos\left[\frac{(2k+1)j\pi}{2n}\right], \qquad j = 0, 1, \ldots, n-1 \tag{15.1}$$

其中

$$c_j = \begin{cases} \dfrac{1}{\sqrt{n}}, & j = 0 \\ \sqrt{\dfrac{2}{n}}, & j \neq 0 \end{cases}$$

这种变换方法计算频率递增、幅度正比于输入值的余弦项的离散和。用逆变换可以恢复原始值，除了可能存在的取整误差：

$$V_k = \sum_{j=0}^{n-1} c_j V_j' \cos\left[\frac{(2k+1)j\pi}{2n}\right], \qquad k = 0, 1, \ldots, n-1 \tag{15.2}$$

变换值 V_j' 常称为逆变换方程中余弦函数的“系数”。这其实是一个不正确的术语，因为在求和过程中余弦项的系数是 c_jV_j'。

下面举例说明 DCT 方法。考虑下列包含 8 个输入值的序列：

{215, 209, 211, 207, 192, 148, 88, 63}

其变换值（精确到两位小数）为

{471.29, 143.81, −67.76, 16.33, 7.42, −4.73, 5.49, 0.05}

我们注意到变换值的幅度显著下降，说明高频余弦项对于恢复原始值的作用很小。这是 DCT 的基本性质。它使我们可以用前面的几个变换值来比较精确地逼近原始值。因此，我们可以计算并存储

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

大约前一半变换值来压缩图像文件。表15.6显示了当使用4个、5个或全部8个变换值来计算输入值时，方程（15.2）的结果。表中所有计算的结果保留两位小数。

表 15.6 逆 DCT 计算

输入值	215	209	211	207	192	148	88	63
参与求和项数				逆 DCT 值：				
4	212.63	211.85	211.53	207.42	188.43	147.65	95.47	58.02
5	215.26	209.23	208.91	210.04	191.06	145.02	92.84	60.64
8	215.00	209.00	211.00	207.00	192.00	148.00	88.00	63.00

我们可以通过变换一个矩形块的输入值，而不是沿扫描线的一行输入值来提高压缩的效率。一个 $n \times n$ 的正方形输入值的变换值计算如下：

$$V'_{lm} = c_{lm} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} V_{jk} \cos\left[\frac{(2k+1)l\pi}{2n}\right] \cos\left[\frac{(2j+1)m\pi}{2n}\right] \tag{15.3}$$

其中

$$l, m = 0, 1, \ldots, n-1$$

并且

$$c_{lm} = \begin{cases} \dfrac{1}{n}, & l = 0 \quad 或 \quad m = 0 \\ \dfrac{2}{n}, & l \neq 0 \quad 和 \quad m \neq 0 \end{cases}$$

同样，用逆变换恢复 $n \times n$ 的输入值：

$$V_{jk} = \sum_{l=0}^{n-1} \sum_{m=0}^{n-1} c_j V'_j \cos\left[\frac{(2j+1)l\pi}{2n}\right] \cos\left[\frac{(2k+1)m\pi}{2n}\right] \tag{15.4}$$

其中

$$j, k = 0, 1, \ldots, n-1$$

通常用 8×8 的输入值进行变换和逆变换，因而要同时处理8条扫描线上的颜色值。

15.4 主流文件格式的结构

在不同的系统、不同的环境中有数百种表示图形数据的文件格式。例如，操作系统的各种系统处理子程序中使用许多种专门设计的文件格式。有些特殊的格式是为了特定的应用而存在的，例如三维建模、动画、图形用户界面、光线追踪软件、视频录制、科学计算可视化软件、绘图程序、字处理系统、电子制表软件包、互联网通信、电视广播和传真。此外，ISO和ANSI标准委员会提出了一些通用的文件格式和压缩系统。

多数光栅文件格式可以处理彩色图像，但也有一些格式只能用于二值位图（bitmap）。然而，格式的名称常常会引起误导，因为术语“位图”常用来指彩色图像（pixmap，像素图文件）。这是延续使用光栅文件的旧称“位图”的结果。在开发彩色显示器之前，所有的光栅文件都存储为表示黑白图案的位图（每个像素一个比特位）。彩色显示技术发展之后，像素图文件（每个像素多个比特位）取代了位图。但这些文件仍然被称为位图。于是，很多彩色图像编码格式现在还被称为位图格式，即使它们实际上是像素图格式。通过参考这些格式的文档可以确定分配给每个像素的比特位数。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

绝大部分本节中描述的文件格式都不是静态的。它们不断地被修改和更新，并且一种格式通常存在多种变体。

15.4.1　JPEG：联合图像专家组

JPEG 格式是 Joint Photographic Experts Group（联合图像专家组）的缩写，由国际标准化组织（ISO）开发。JPEG 的基本格式包含大量文件压缩选项，是一个广泛应用的复杂系统。JPEG 定义中给出了从简单的无损压缩算法到很高压缩比的有损压缩算法的二十几种变体，因此可以用多种不同的方式实现。但是基本 JPEG 定义中并未完全说明如何组织压缩文件，使其可以在不同的计算机系统上或被不同的应用程序使用。例如，没有指定文件头信息的格式，也没有指定对压缩文件中使用的颜色模型的说明。

JPEG 标准定义了四种一般的文件压缩模式，分别称为无损模式、顺序（sequential）模式、渐进（progressive）模式和层次（hierarchical）模式。**JPEG 无损模式**（JPEG lossless mode）将模式识别方案与霍夫曼编码方法或者算术编码方法相结合。然而，原先的 JPEG 无损模式的效率不如其他无损压缩格式，因此很少被应用。**JPEG 基线顺序模式**（JPEG baseline sequential mode）是最常使用的 JPEG 版本，它用 8 个比特存储图像中颜色分量的数值，并且压缩算法结合了 DCT 变换和霍夫曼编码或算术编码。扩展顺序模式（extended sequential mode）比基线顺序模式定义了更多的选项，其中可以用 16 个比特来指定颜色分量。**JPEG 渐进模式**（JPEG progressive mode）通过几轮处理来压缩图像文件，从而生成不同分辨率的图像的多层表示。这种常被称为渐进式 JPEG（progressive JPEG）的模式在互联网应用中相当流行，因为在下载完整个图像文件之前就可以看见图像的概况。**JPEG 层次模式**（JPEG hierarchical mode）包含另外一组用于得到图像的逐渐精化的版本的子程序。它将图像分割为一组子图像，使得可以逐步构造图像的选定部分。由于其复杂性，层次式 JPEG 的使用并不广泛。

大规模的JPEG实现可以提供选择压缩模式和压缩参数的选项，如在逆DCT变换的求和计算中使用的余弦项的个数。同时，JPEG 压缩标准还说明霍夫曼编码或者算术编码都可以与 DCT 变换进行组合。但 JPEG 的实现从不使用算术编码，因为这些算法受专利保护，需要支付许可证费用。

尽管 JPEG 标准没有定义压缩图像文件的明确结构，但是现在的 JPEG 实现都使用由 C-Cube Microsystems 公司的 Eric Hamilton 提出的、基于许多 JPEG 用户建议的 **JPEG 文件交换格式**（JPEG File Interchange Format，JFIF）。JFIF 格式的文件头中包含一个像素的 JFIF 标识符（称为文件“签名”）、用来设置文件的 JFIF 版本、图像尺寸（以每厘米像素数或每英寸像素数度量）、可选的 RGB 预览图像（称为“thumbnail”）的宽、高以及 RGB 值。压缩文件中的像素值用 YC_rC_b 颜色模型存储，其中颜色分量按 Y、C_b、C_r 的顺序存储。灰度图像中只使用了 Y 分量。文件中的其他信息包括压缩算法使用的表。JPEG 文件中的整数用“高位在前”的顺序存储。

JPEG/JFIF 的基线顺序编码模式通常包括以下操作：

(1) **颜色转换**（Color Conversion）：图像像素的 RGB 颜色值转换为 YC_rC_b 颜色值。

(2) **颜色采样**（Color Sampling）：仅使用选定像素的颜色或平均相邻颜色组的颜色分量可以减少文件中颜色值的数量。简单的采样操作可以从每两个、三个或四个像素中选取一个颜色值。对于不同的颜色分量通常使用不同的采样频率，从而选取较多的亮度值（Y 分量）。因为只存储较少的色度值（C_r 和 C_b 分量），所以可以得到更大的压缩比。

(3) **DCT 变换**（Discrete Cosine Transform）：用方程（15.3）将 8 × 8 的像素颜色值转换为 DCT 值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

(4) **变换值缩减**（Reduction of Transformed Value）：为进一步压缩编码图像文件，仅存储变换值的缩减集合（参见 15.3 节）。缩减集合中值的数量可以是固定的，也可以用算法进行计算来确定不同变换项的影响。

(5) **霍夫曼编码**（霍夫曼 Encoding）：最后一个压缩操作是将 DCT 值转换成霍夫曼代码（参见 15.3 节）。

由 Eric Hamilton 和 ISO JPEG 委员会开发的静止图像交换文件格式（Still-Picture Interchange File Format，SPIFF）是 JFIF 的扩展。SPIFF 包含比 JFIF 更多的特性和选项。预期 SPIFF 将最终取代 JIFF 作为 JPEG 的实现。然而，像 JPEG 一样，SPIFF 包含了比一个实现所实际需要的多得多的选项。例如，JFIF 只使用一个颜色模型（YC_rC_b），而 SPIFF 支持 13 种不同的颜色模型。

对于真实感计算机图像和数字照片，现在的 JPEG 实现提供比其他系统更高的压缩比。但是，对于包含大范围单色区域的简单图像，其他格式在不损失颜色信息的条件下也可以提供同等的压缩比。

15.4.2 CGM

CGM 是 Computer-Graphics Metafile Format（计算机图形元文件格式）的缩写，它也是 ISO 和 ANSI 为任何计算机系统和任何计算机图形领域开发的标准，包括科学计算可视化、CAD、图形艺术、商业图形、电子出版以及使用 GKS 或 PHIGS 图形库的任何应用。因此，CGM 支持多种特性和选项。

正如名称“元文件”所指，CGM 允许图像说明以像素图给出，或者以一组几何定义给出，包括诸如线宽、线型、填充类型和字符串说明。图像文件中也可以包含其他各种参数，例如颜色分量的最大值、彩色表的大小、文件中使用的字体列表和裁剪窗口的边界。

CGM 采用字符编码方案来减小文件尺寸，并且为快速编、解码优化了二进制数值代码。可以用多种颜色方案（如 RGB、CMYK、YC_rC_b、CIE 和彩色表）给出像素值。此外，还可以使用行程编码的变体和霍夫曼编码来压缩像素图文件。

15.4.3 TIFF

TIFF 是 Tag Image-File Format（标记图像文件格式）的缩写，它是由 Aldus 公司作为主席的计算机公司协会开发的一种为了在不同的应用和计算机系统之间传送光栅图像的高效格式。虽然它相当复杂，但 TIFF 是一种最通用的格式，并且可以为个别应用进行定制。TIFF 被广泛使用在不同的领域，包括医学成像、桌面出版、图形用户界面、卫星图像存储和传真。

TIFF 格式可以用于二值、灰度和全彩色图像，并且 TIFF 可以存储多幅光栅图像。像素颜色可以用 RGB 分量或者颜色表给出。TIFF 提供比其他任何系统都多的压缩方案选择，包括行程编码的组合、LZW 编码、霍夫曼编码和一套 JPEG 方法。

15.4.4 PNG

PNG 是 Portable Network-Graphics Format（可移植网络图形格式）的缩写，它是由一组独立的开发人员设计的。PNG 提供高效的图像无损压缩方案。PNG 的压缩算法包括霍夫曼编码和 LZ 编码的变体。这种格式正在互联网的图像存储和传输中逐渐流行。它对于暂时存储图像以进行反复修改也很有用。对于简单的计算机图形组成的图画，PNG 可以得到可与 JPEG 压缩文件相比的、很高的压缩比。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

PNG 中的整数值按“高位在前”的顺序存储，颜色分量可以用最多 16 位精度指定。PNG 支持许多选项，包括 RGB 颜色分量、XYZ 颜色分量、灰度、颜色表和透明信息的 alpha 值。

15.4.5　XBM 和 XPM

XBM 和 XPM 分别是 X Window System Bitmap Format（X Window 系统位图格式）和 X Window System Pixmap Format（X Window 系统像素图格式）的缩写。与其他格式不同，它们用 C 和 C++代码存储在使用 X Window 系统的工作站上处理的图画信息。像素值用阵列表示，按扫描线顺序从左向右存储。XBM 是位图（每个像素一个比特）的格式，XPM 是像素图（每个像素多个比特）的格式。大多数 Web 浏览器支持这两种格式。

XBM 和 XPM 格式不包含压缩算法，但可以使用专门设计的压缩程序来减小文件的尺寸。它们用#define 预处理指示来代替文件头，用来指定扫描行数目及每一扫描行的像素数目等。在 XBM 格式中，位值为 1 表示当前的前景色，而位值为 0 表示当前的背景色。在 XPM 格式中，像素值可用 RGB 或 HSV 分量来存储。

15.4.6　Adobe Photoshop 格式

Adobe Photoshop 是广泛使用的图像处理软件。Adobe Photoshop 格式为快速存取大尺寸、全彩色光栅图像而优化。相反，Photoshop 仅使用行程编码进行少许压缩，而早期的 Photoshop 根本不包含压缩方法。

Photoshop 按“高位在前”的顺序存储整数，并提供许多选项。它支持像素图、位图和灰度图像。颜色可以用 RGB 颜色分量、CMYK 颜色分量或颜色表存储。它还提供多种方案用于表示每个像素多种颜色、半色调图像和透明参数。

15.4.7　MacPaint 绘图格式

MacPaint 是 Apple 公司的一个产品。MacPaint 绘图格式是所有 Macintosh 应用程序的标准格式。这种格式的图像文件是位图，其中 0 代表白色，1 代表黑色。MacPaint 格式通常用于文本、线条图和剪贴画。

MacPaint 按“高位在前”的顺序存储整数。它总是包含 720 条水平扫描线，并且每条扫描线上有 576 个像素。它用一种行程编码方案来压缩图像。

15.4.8　PICT

PICT 是 Picture Data Format（图画数据格式）的缩写。它是 Apple 公司为 Macintosh 应用程序设计的一种混合格式。它支持用位图、像素图或几何表示来指定图像。用几何表示的 PICT 文件包含一组 Macintosh QuickDraw 函数。这组函数将一幅图定义为线段、多边形、圆弧、位图、其他对象、裁剪参数、属性和其他状态参数。

可以用单色位图、RGB 颜色分量或颜色表来指定图像。光栅文件可以用行程编码算法进行压缩。

15.4.9　BMP

BMP 是 Bitmap Format（位图格式）的缩写。虽然名为位图，BMP 实际上支持每个像素多比特颜色的图像文件。这种格式由 Microsoft 公司为 Windows 操作系统的应用程序开发。另一种也称为 BMP 的类似格式应用在 IBM OS/2 操作系统上。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

BMP文件中的整数按“低位在前”的顺序存储。每个像素可以用1、2、4、8、16、24或32个比特表示。颜色值可以用RGB颜色分量或者颜色表指定。像素扫描线按自底向上的顺序存储，并且坐标原点在像素图空间的左下角。BMP文件通常不压缩，但行程编码算法可以应用在像素图上，使得用4个或8个比特表示每个像素。

15.4.10 PCX

PCX是PC Paintbrush文件格式。它由ZSoft公司开发，是Windows操作系统上使用的另一种像素图格式。PCX文件中的一个像素可以用1到24个比特表示，且颜色值可以用RGB颜色分量或者颜色表指定。整数按“低位在前”的顺序存储。像素扫描线按自顶向下的顺序存储。光栅文件可以用行程编码算法压缩。

15.4.11 TGA

TGA是Truevision Graphics-Adapter Format（Truevision图形适配器格式）的缩写。它由Truevision公司开发，用于Targa和Vista图形适配器。TGA像素图格式也称为**Targa格式**（Targa format）。这种格式在视频编辑应用中很流行。

TGA格式按“低位在前”的顺序存储像素值。每个像素可以用8、16、24或32比特表示。像素颜色用RGB颜色分量或者两种颜色表指定。可以用一个单独的RGB颜色表，也可以用三个分开的表分别表示R、G、B分量。TGA格式通常不压缩，但对大文件也可以使用行程编码算法。

15.4.12 GIF

GIF是Graphics Interchange Format（图形交换格式）的缩写。它由CompuServe公司设计用于高效地电传光栅图像。GIF采用LZW算法，对于简单的计算机图形组成的图画可以得到相当好的压缩比。但是GIF对于真实感图像的压缩能力不如JPEG或PNG。虽然GIF已经被许多应用程序使用，但是，由于LZW压缩算法的专利问题，它的流行程度正在急剧下降。

GIF可以处理单色或多色图像，但每个像素只能用1到8个像素表示，也就是最多256种颜色。像素值用RGB颜色表，按“低位在前”的顺序存储。

15.5 小结

光栅图形系统的典型的图像文件是RGB像素图，也常称为原始光栅文件。RGB颜色值用整数存储，其最小值为0，最大值则取决于可用来表示每个像素的最多比特数。图画也可以用图画组件（例如线段、填充区域或样条曲线）的几何描述来表示。

当光栅文件在系统之间传送或者以某种形式存储时，可能需要减少图像中使用的颜色数量。我们可以通过多种方式（如平均）合并颜色，从而均匀地减少颜色数量。基于出现频率的降色方法选取出现最频繁的那些颜色值。中值切割降色法将颜色空间细分为一组块，然后用每个块中的平均颜色取代所有位于该块中的颜色。

有多种格式可以在特定应用或特定系统中方便地存储图像文件。这些格式的不同之处在于文件头、整数的字节顺序（高位在前或低位在前）和压缩方法。文件压缩方法的效率用压缩比（原始文件的字节数与压缩文件的字节数的比值）来衡量。改变颜色值的压缩方法称为有损压缩方法，而精确恢复颜色值的方法称为无损压缩方法。有些文件格式还使用降色方法。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

行程编码是常用的压缩方法，它将一列重复的像素值替换为一个值和一个行程长度。LZW 压缩方法是行程编码方法的一个变体，它用一个代码来替换一种重复出现的模式。其他基于模式识别的压缩方法包括扫描线比较方法和寻找自相似像素组的分形方法。霍夫曼编码方法为颜色值分配变长代码，使得最频繁出现的值具有最短的代码。算术编码用颜色值在图像文件中的频率计数来在从 0.0 到 1.0 的单位区间中创建子区间，然后用每个子区间的边界来编码对应于该子区间的像素值序列。离散余弦变换方法用频率递增的余弦项与像素值相乘，然后计算乘积的和。求和操作将一组像素值转换为一组变换值，然后通过消除某些变换值进行图像的有损压缩。

在不同的计算机系统上的各种图形应用程序使用各种各样的文件格式。有些格式由标准化组织 ISO 和 ANSI 开发，有些由计算机软件或硬件公司开发，还有一些来自独立团体。广泛使用的文件格式包括 JPEG、TIFF、PNG 和那些在 X Windows 系统、Apple Macintosh 计算机和 Windows 操作系统上使用的格式。

参考文献

Heckbert(1982 and 1994)、Glassner(1990)、Arvo(1991)和 Kirk(1992)给出了降色方法。Gonzalez and Wintz(1987)概括讨论了变换方法和图像处理技术。Huffman(1952)、Ziv and Lempel(1977 and 1978)、Welch(1984)、Rao and Yip(1990)、Arvo(1991)和 Barnsley and Hurd(1993)详细描述了各种文件压缩方法。

图形文件格式的一般信息请参见 Brown and Shepherd(1995)和 Miano(1999)。JPEG 的其他信息请参见 Taubman and Marcellin(2001)。Henderson and Mumford(1993)详细说明了 CGM 文件格式的标准。

练习题

15.1　编写一个程序，在全彩色系统中对所有颜色级进行均匀降色，其中每个 RGB 颜色分量的整数取值范围都是从0到255。输入任意一个整数作为每个颜色分量的除法因子 d，输出是一组减少了的颜色级。

15.2　修改上一练习题中的程序，使输入整数 k 作为输出的减少后的颜色级数。

15.3　修改上一练习题中的程序，使 R、G、B 分量减少不同的颜色级数。减少量可以通过指定每个分量的整数范围或者指定比特位数。

15.4　编写一个程序，用基于频率的降色方法将输入图像降为 k 种颜色。程序的输入是一个像素颜色的数组和数组的大小。数组的大小用扫描线数和每条线上的像素数来指定。

15.5　编写一个程序实现中值切割降色方法。将一个包含 n 种 RGB 颜色的图像降为 k 种颜色。

15.6　编写一个程序，对一条包含 1024 个整数值（0 ~ 255）的扫描线进行行程编码。

15.7　修改上一练习题中的程序，使输入文件包含 n 条扫描线。

15.8　编写一个程序，对一条包含 1024 个整数值（0 ~ 255）的扫描线进行简化的 LZ 编码。程序仅搜索由 3 个元素组成的 RGB 颜色重复的模式。对模式使用整数代码。

15.9　扩展上一练习题中的程序，使输入文件包含 n 条扫描线。

15.10　给定一个包含 n 条扫描线、每条扫描线上 m 个 RGB 像素的输入文件，编写一个程序输出像素颜色的频率计数表。

15.11　使用上一练习题中的频率计数表，编写一个程序用霍夫曼编码压缩文件。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

15.12 使用练习题 15.10 中的频率计数表，编写一个程序用算术编码压缩文件。

15.13 给定一个 32 个像素颜色（RGB）的列表，编写一个程序计算列表中每一组连续 8 个像素的离散余弦变换值（参见等式（15.1））。

15.14 用方程（15.2）和上一练习题中产生的变换值，编写一个程序计算原先（恢复）的 32 个像素颜色。

15.15 修改上一练习题，计算每组 8 个像素的逆变换值。使用任意 n 个选定的变换值，其中 n 的取值可以从 1 到 8。

15.16 给定一个 32×32 像素颜色（RGB）的图像文件，编写一个程序计算文件中每一组连续 8×8 个像素的离散余弦变换值（参见等式（15.3））。

15.17 用方程（15.4）和上一练习题中产生的变换值，编写一个程序计算原先（恢复）的 32×32 个像素颜色。

15.18 修改上一练习题，计算每组 8×8 个像素的逆变换值。使用任意 $n \times m$ 个选定的变换值，其中 n 和 m 的取值都可以从 1 到 8。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

附录 A　计算机图形学的数学基础

计算机图形学算法要用到许多数学概念和技术。在此，我们就解析几何、线性代数、向量分析、张量分析、复数、四元数、微积分、数值分析等方面以及本书讨论的图形算法所涉及的其他领域而提供简明的参考。

A.1　坐标参照系

笛卡儿坐标系和非笛卡儿坐标系在计算机图形应用中都经常用到。图形程序一般使用笛卡儿参照系来指定坐标，但对场景的初始描述常常会以非笛卡儿参照系的坐标给出。球面的、圆柱的或其他对称坐标系常被用来简化描述或处理对象的表达。

A.1.1　二维笛卡儿屏幕坐标系

对于图形软件包中的设备无关的命令，屏幕坐标位置在坐标原点位于屏幕的左下角的二维笛卡儿参照系的第一象限内定义，如图 A.1(a)所示。然而，由于屏幕的扫描线是自顶向下从 0 开始编号的，因而屏幕坐标的内部表示是以屏幕的左上角作为原点的。设备相关的命令（例如交互式输入和窗口操作）通常采用反转的笛卡儿参照系来定义屏幕坐标，如图 A.1(b)所示。这两个参照系的 x 坐标是相同的，反转的 y 坐标转换到笛卡儿坐标系的 y 坐标的方程如下：

$$y = y_{max} - y_{invert} \tag{A.1}$$

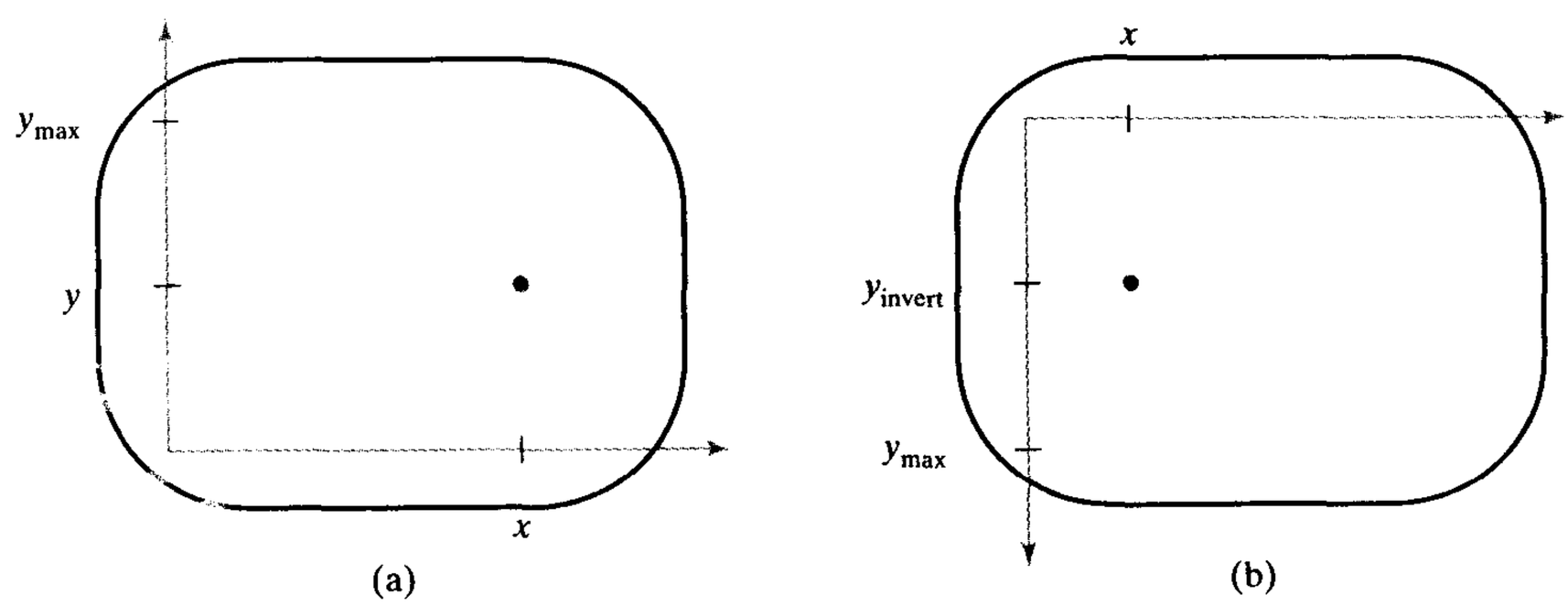


图 A.1　屏幕笛卡儿参照系：(a)坐标原点在屏幕左下角；(b)坐标原点在左上角

在某些软件包中，屏幕坐标原点可以放置在任何位置，例如屏幕中心。

A.1.2　标准二维笛卡儿参照系

我们使用标准笛卡儿参照系来描述世界坐标、观察坐标和其他二维观察流水线中的坐标定义。参照系中的坐标可以是任意的正值或负值。为显示一幅图，我们要指定一个裁剪窗口和一个视口来将图的一部分映射到屏幕坐标。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

A.1.3 *xy* 平面的极坐标

常用的非笛卡儿系是极坐标系（参见图 A.2）。其中，坐标位置由离坐标原点的极半径距离 r 和距水平轴的角位移量 θ 指定。正的角位移量是逆时针的，而负的角位移量是顺时针的。笛卡儿和极坐标之间的关系示例于图 A.3。考虑图 A.4 的直角三角形，并利用三角函数的定义，可以从极坐标变换成笛卡儿坐标，使用表达式：

$$x = r\cos\theta, \qquad y = r\sin\theta \tag{A.2}$$

从笛卡儿到极坐标的逆变换为

$$r = \sqrt{x^2 + y^2}, \qquad \theta = \tan^{-1}\left(\frac{y}{x}\right) \tag{A.3}$$

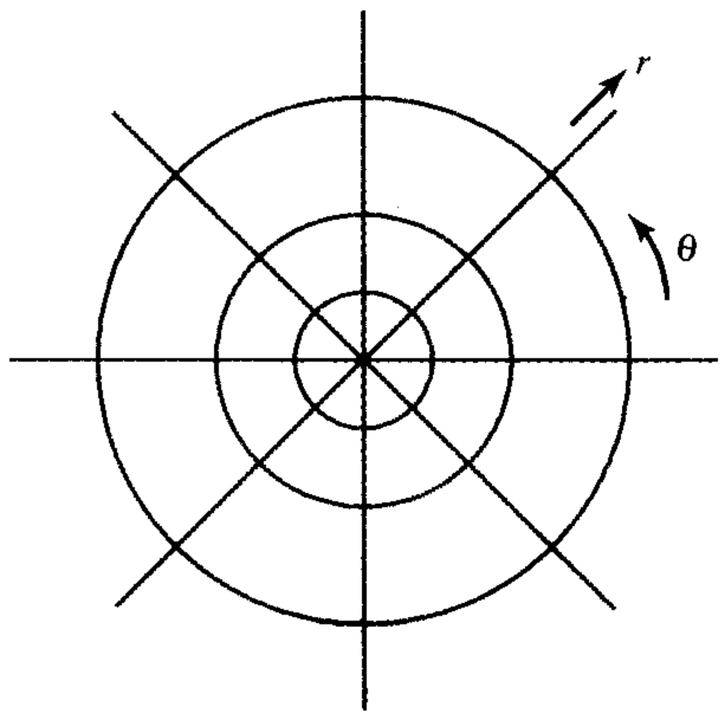


图 A.2 极坐标参照系，由同心圆和辐射线形成

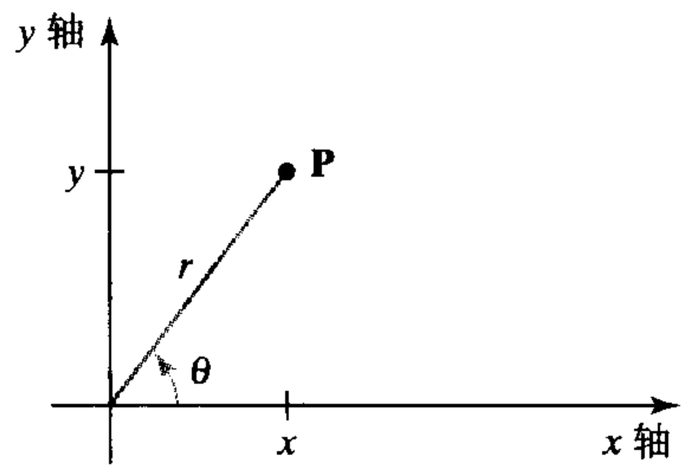


图 A.3 极坐标和笛卡儿坐标的相互关系

角度值可由指定度数或者以非维单位（弧度）给出。1 弧度是指长度等于圆的半径的圆弧所对应的圆心角的大小。图 A.5 给出平面上两交叉线和以交叉点 **P** 为圆心的圆。任何以 **P** 为中心的圆的 θ 角的弧度值计算如下：

$$\theta = \frac{s}{r} \qquad \text{（弧度）} \tag{A.4}$$

其中，s 是对应 θ 的弧长，而 r 为圆半径。绕点 **P** 的总角距为圆周长（$2\pi r$）除以 r，或 2π 弧度。如果用角度来表示，整个圆周被等分为 360 段圆弧，每个圆弧对应的圆心角为 1 度。因此，$360° = 2\pi$ 弧度。

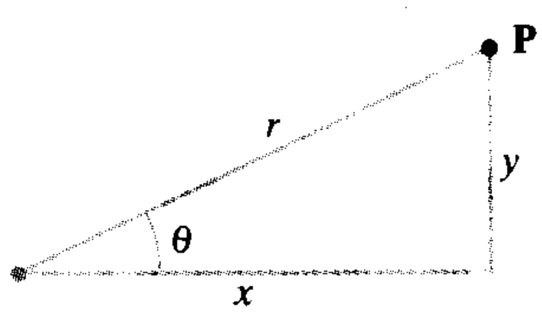


图 A.4 带斜边 r 和边 x、y 的直角三角形

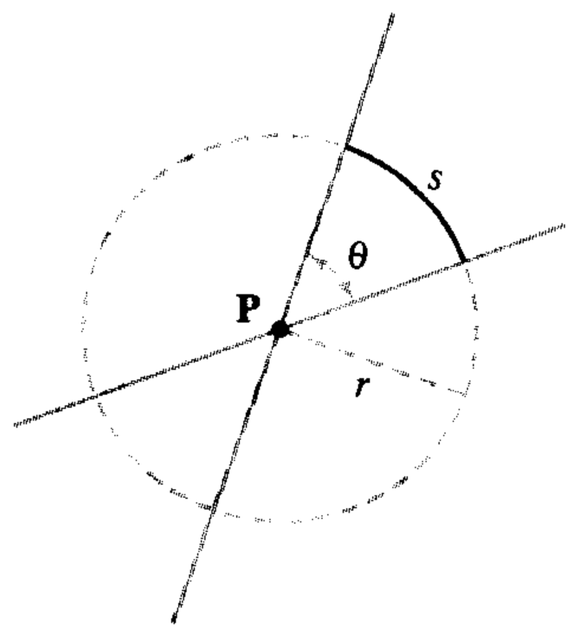


图 A.5 由弧长 s 和半径 r 构成的 θ 角

除了圆以外，其他圆锥形也可用来指定坐标位置。例如，使用同心椭圆而不是圆，我们可以在椭圆坐标系给出坐标位置。类似地，可以使用的其他类型的对称系统有双曲线或抛物线平面坐标。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

A.1.4　标准三维笛卡儿参照系

图 A.6(a)给出了在三维笛卡儿参照系中坐标轴的常规定向。称之为右手系是因为，当我们用右手握住 z 轴时，手指从正 x 轴向正 y 轴环绕（通过 90°），右手拇指指向 z 轴正向，如图 A.6(b)所示。多数计算机图形软件包需要使用右手笛卡儿坐标来描述和处理对象。除非特别指明，本书从头到尾（包括附录）的所有笛卡儿参照系都是右手的。

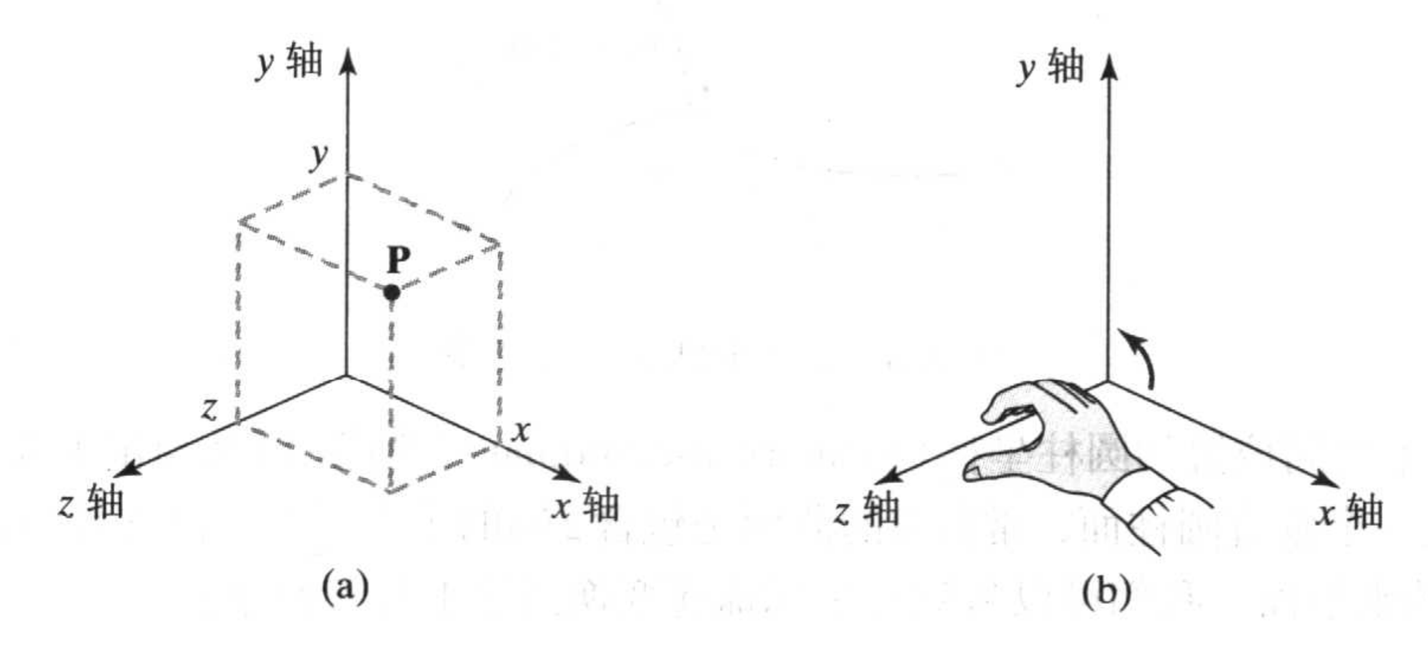


图 A.6　在右手笛卡儿参照系中，位于(x, y, z)的点 **P** 的坐标表示

笛卡儿参照系是**正交坐标系**（orthogonal coordinate system），也就是说，所有的坐标轴两两相互垂直。并且，笛卡儿参照系的坐标轴都是直线。但是在许多应用中曲线坐标轴非常有用。多数曲线坐标系仍然是正交的，也就是说，在坐标空间内的任何一点上坐标轴的方向都是互相垂直的。

A.1.5　三维笛卡儿屏幕坐标系

当在视频监视器上显示三维场景的一个视图时，屏幕上的每个位置都包含深度信息。对应屏幕点的三维位置通常用左手系来定义，如图 A.7 所示。对于这种坐标系，当我们用左手握住 z 轴时，左手手指从正 x 轴向正 y 轴旋转 90°，左手拇指则指向正 z 轴方向。轴的这种定向有时便于描述对象相对于屏幕的深度。对于 xy 平面上的每个点，正 z 值指向屏幕背后的位置，并且沿正 z 轴较大的值表明离观察者较远。

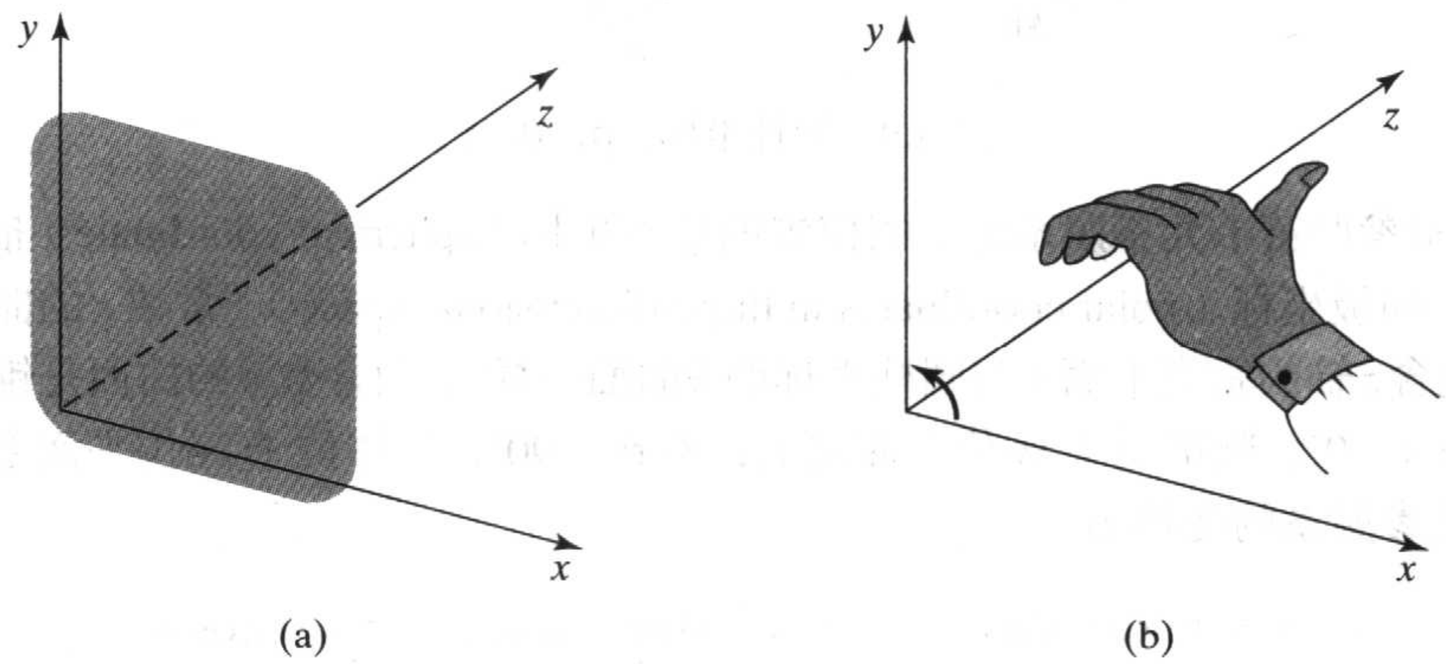


图 A.7　在视频监视器上设置的左手笛卡儿坐标系

A.1.6　三维曲线坐标系

任何非笛卡儿参照系则看成是**曲线坐标系**（curvilinear-coordinate system）。对于特殊图形应用的坐标系的选择，取决于诸多因素，如对称性、计算的容易程度及形象化的好处。图 A.8 表示由三

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

个坐标曲面形成的通用曲线坐标参照系，其中每个曲面有一个坐标保持常数。例如，x_1x_2 曲面由 x_3 保持常数来定义。在任意参照系中的坐标轴是坐标曲面之间的交叉曲线。如果坐标曲面以直角交叉，我们就有**正交曲线坐标系**（orthogonal curvilinear-coordinate system）。非正交参照系用于特定空间，如受一般相对性定律控制的运动的可视化，但通常这种坐标系在图形应用中比正交系用得少。

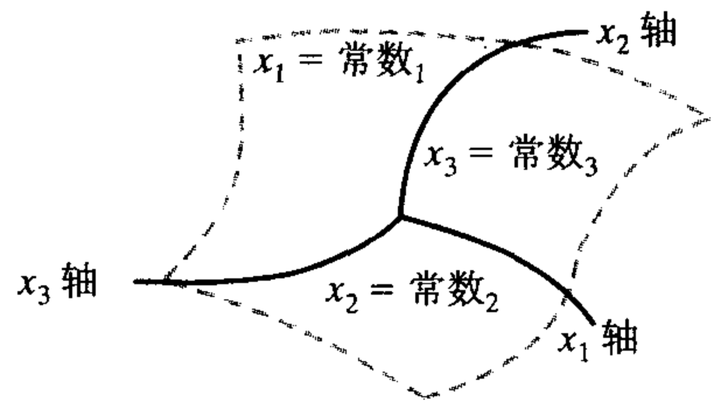


图 A.8 通用曲线坐标参照系

图A.9给出了空间位置的圆柱坐标（cylindrical-coordinate）描述，以及与笛卡儿参照系的关系。常数 ρ 的曲面是一个垂直圆柱面，常数 θ 的曲面是包含 z 轴的垂直平面，而常数 z 的曲面是平行于笛卡儿 xy 平面的水平面。我们可以从圆柱坐标描述变换到笛卡儿参照系：

$$x = \rho\cos\theta, \qquad y = \rho\sin\theta, \qquad z = z \tag{A.5}$$

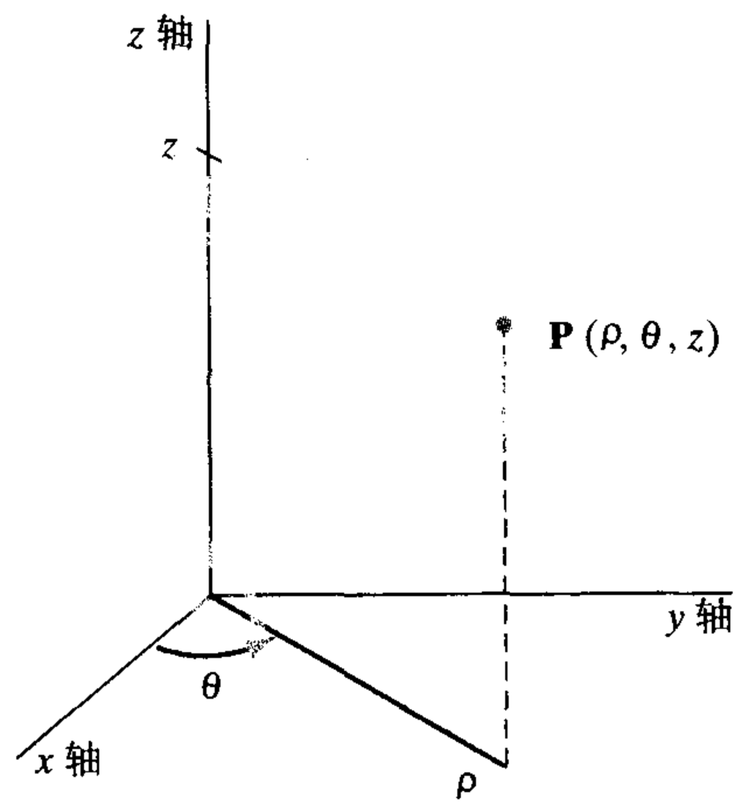


图 A.9 圆柱坐标：ρ，θ，z

图 A.10 表示参照笛卡儿参照系的空间位置的球面坐标（spherical-coordinate）描述。球面坐标有时看成三维空间极坐标（polar coordinates in three-dimensional space）。常数 r 的曲面是球面，常数 θ 的曲面是包含 z 轴的垂直平面（与圆柱坐标的 θ 曲面一样），而常数 ϕ 的曲面是锥顶在坐标原点的圆锥。如果 $\phi < 90°$，圆锥位于 xy 位平面之上；若 $\phi > 90°$，锥体位于 xy 位平面之下。从球面坐标描述到笛卡儿参照系的变换为

$$x = r\cos\theta\sin\phi, \qquad y = r\sin\theta\sin\phi, \qquad z = r\cos\phi \tag{A.6}$$

A.1.7 立体角

我们使用与二维角 θ 由两交叉线定义（方程 A.4）相类似的方法来定义立体角 ω。代之以圆，我们考虑中心位置在 **P** 点的任意球面（参见图 A.11），顶点在 **P** 的锥形区域内的立体角 ω 定义为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\omega = \frac{A}{r^2} \tag{A.7}$$

其中，A 是圆锥所切割球表面的面积，而 r 是球半径。

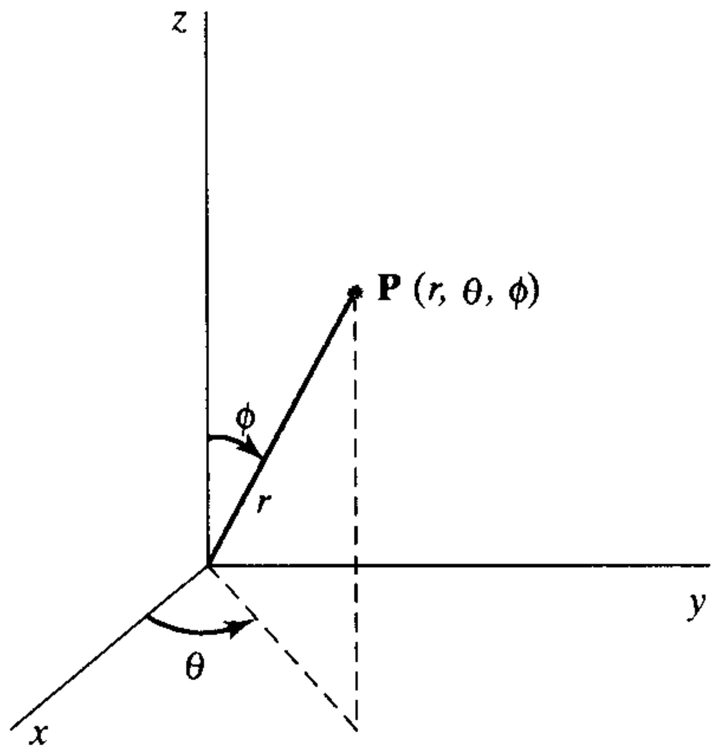


图 A.10　球面坐标：r，θ，ϕ

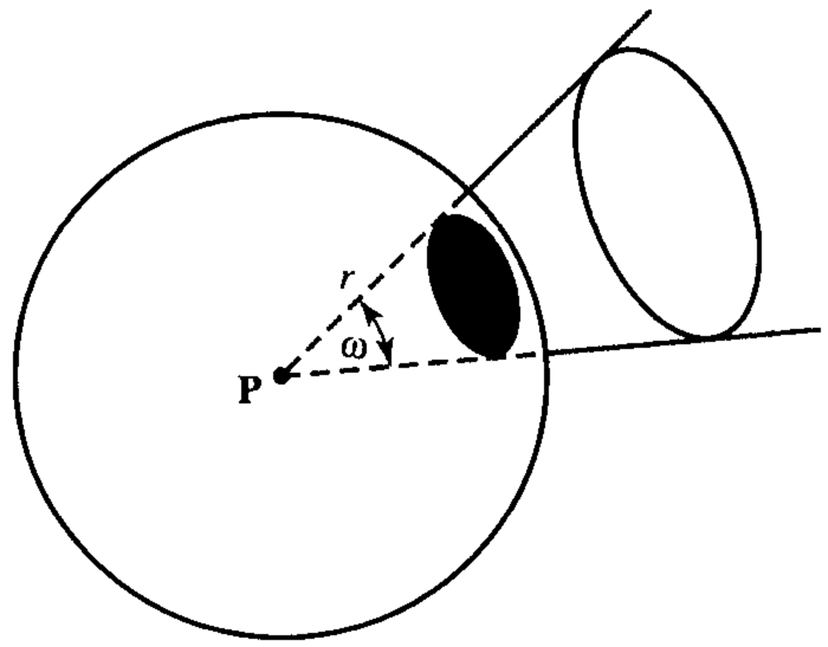


图 A.11　由球表面面积 A 和半径 r 确定的立体角 ω

同样，与二维极坐标类似，立体角的非维单位称**立体弧度**（steradian）。关于一点的总立体角是球表面的总面积（$4\pi r^2$）除以 r^2，或 4π 立体弧度。

A.2　点与向量

点与向量的概念之间有着根本的差别。点是在某参照系以坐标值指定的位置。因此点的坐标和其他性质依赖于参照系的选择。然而，向量的性质是与特定的坐标系无关的。

A.2.1　点的性质

图 A.12 给出了二维点 **P** 在两个参照系中的坐标描述。在 A 坐标系中，该点坐标由一个有序对 (x, y)指定，它到坐标原点的距离为$\sqrt{x^2+y^2}$；在 B 坐标系中，同一点的坐标为(0，0)，而到 B 坐标系原点的距离为 0。

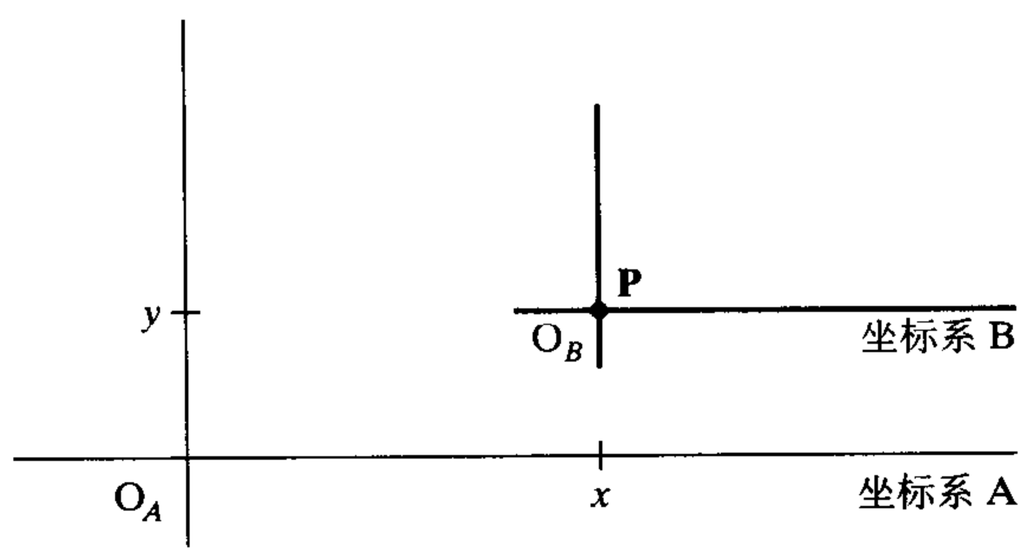


图 A.12　涉及两个不同笛卡儿参照系的 **P** 点位置

A.2.2　向量的性质

在一个选定的坐标系中，一个向量定义为两个点位置之间的差。因此，对于图 A.13 中的二维点位置 $\mathbf{P}_1$ 和 $\mathbf{P}_2$，我们可以指定一个向量 **V** 如下：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\begin{aligned}\mathbf{V} &= \mathbf{P}_2 - \mathbf{P}_1 \\ &= (x_2 - x_1,\ y_2 - y_1) \\ &= (V_x,\ V_y)\end{aligned} \tag{A.8}$$

其中，笛卡儿分量 V_x 和 V_y 是 **V** 分别在 x 轴和 y 轴上的投影。在这个坐标参照系中，我们还可以通过其他两个点位置得到同样的向量。实际上，存在无穷多个点对可以生成同样的向量分量，并且一个向量常常用相对于当前坐标参照系的一个点位置来定义。因此，向量在一个坐标系中没有固定位置。如果将 **V** 的表示变换到其他参照系中，点位置 $\mathbf{P}_1$ 和 $\mathbf{P}_2$ 会改变，但向量的基本性质保持不变。

我们可以将向量描述为一个有向线段（directed line segment），它有两个基本特性：大小和方向。对于图 A.13 所示的二维向量，我们用 Pythagorean 理论计算向量大小，即向量的两个端点之间沿向量方向的直线距离：

$$|\mathbf{V}| = \sqrt{V_x^2 + V_y^2} \tag{A.9}$$

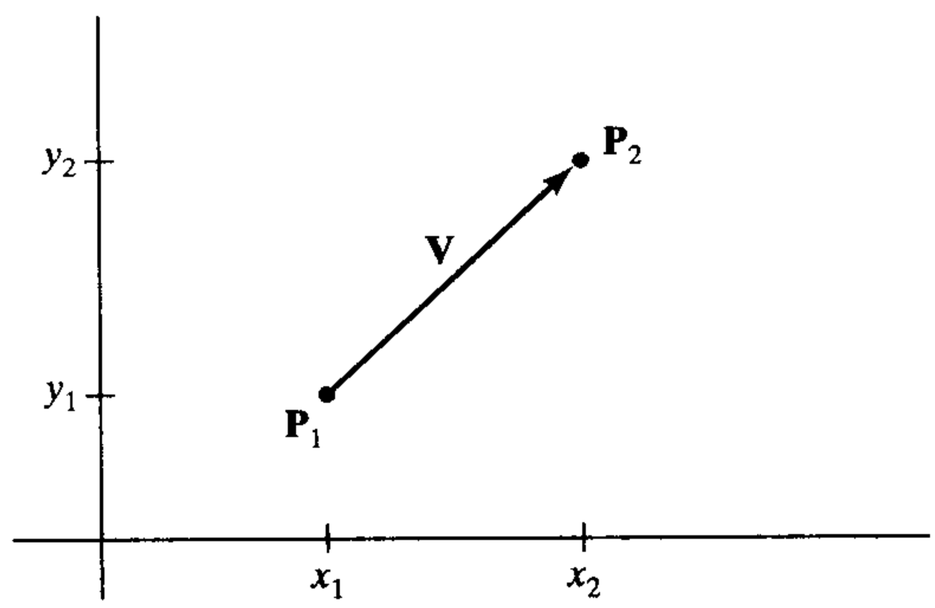


图 A.13　在笛卡儿坐标系 xy 平面的向量 **V**

可以用多种方式指定向量的方向。例如，二维向量的方向可以根据离 x 轴的角位移量而给出：

$$\alpha = \tan^{-1}\left(\frac{V_y}{V_x}\right) \tag{A.10}$$

向量具有同样的特性（量值和方向），而不管我们把向量在单一坐标系内定位在何处。并且，向量量值是独立于坐标表示的。当然，如果我们将向量变换到另外一个参照系，向量分量的值和向量的方向也可能改变。例如，我们可以将上述向量变换到一个旋转过的笛卡儿参照系中，使向量的方向沿着新的 y 轴方向。

对于三维笛卡儿向量表示 $\mathbf{V} = (V_x, V_y, V_z)$，我们计算向量的大小为

$$|\mathbf{V}| = \sqrt{V_x^2 + V_y^2 + V_z^2} \tag{A.11}$$

向量方向由方向角 α、β 和 γ 给出，它们分别是该向量与各坐标轴形成的角（参见图 A.14）。向量与每个正向坐标轴形成的方向角为正。我们计算这些角为

$$\cos\alpha = \frac{V_x}{|\mathbf{V}|}, \qquad \cos\beta = \frac{V_y}{|\mathbf{V}|}, \qquad \cos\gamma = \frac{V_z}{|\mathbf{V}|} \tag{A.12}$$

$\cos\alpha$、$\cos\beta$ 和 $\cos\gamma$ 称为向量的方向余弦。实际上，为给定 **V** 的方向，只需要指定两个方向余弦。因为

$$\cos^2\alpha + \cos^2\beta + \cos^2\gamma = 1 \tag{A.13}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

向量用来表示具有大小和方向特性的任意量，两个普通的例子就是力和速度（参见图 A.15）。力可以设想为在特定方向的推或拉的确定量。速度向量规定一物体在某个方向移动的快慢。

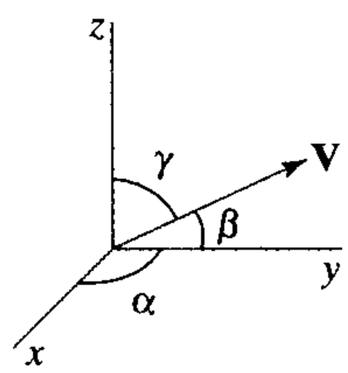


图 A.14　方向角 α、β 和 γ

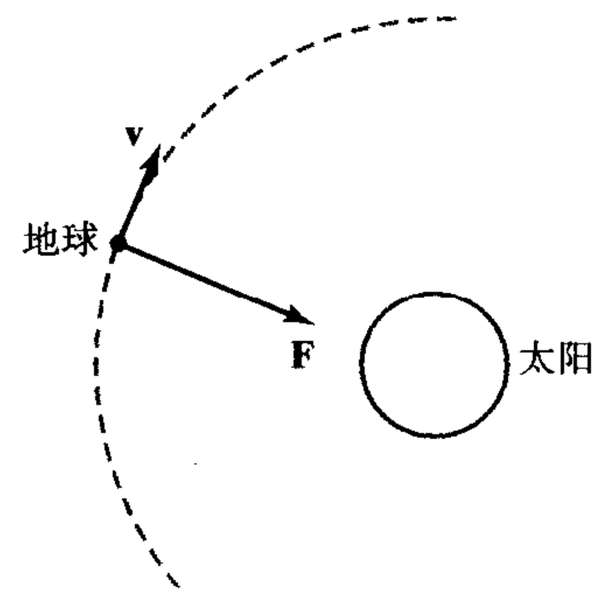


图 A.15　万有引力向量 **F** 和速度向量 **v**

A.2.3　向量加与标量乘

按定义，两向量之和由其对应分量相加得到：

$$\mathbf{V}_1 + \mathbf{V}_2 = (V_{1x} + V_{2x}, V_{1y} + V_{2y}, V_{1z} + V_{2z}) \tag{A.14}$$

二维向量相加的几何说明示于图 A.16。将一向量的起始位置放在另一向量顶部，并按图 A.16 那样从前者的起始位置到后者的顶部画出和向量。向量和标量的加法是没有定义的，因为标量总是只有一个数值，而一个 n 维空间的向量有 n 个数值分量。

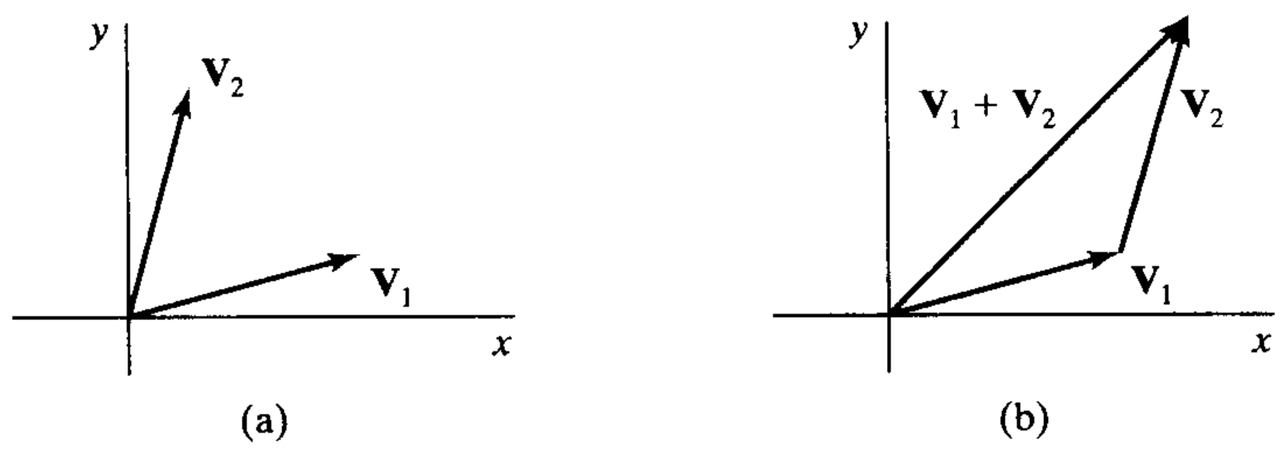


图 A.16　(a)两向量可通过端点接端点定位两向量而实现几何相加；(b)从第一向量起点到第二向量终点画出结果向量

三维向量与标量的乘法定义为

$$s\mathbf{V} = (sV_x, sV_y, sV_z) \tag{A.15}$$

例如，若标量参数 s 取值 2，则 **V** 的每个分量加倍。

也可用多种乘法运算来结合两个向量。一种最常用的方法是将两个向量的大小相乘。进行乘法后或者得到另一个向量，或者得到一个标量。

A.2.4　两向量的标量积

产生标量的向量乘法定义为

$$\mathbf{V}_1 \cdot \mathbf{V}_2 = |\mathbf{V}_1|\,|\mathbf{V}_2| \cos\theta, \qquad 0 \leqslant \theta \leqslant \pi \tag{A.16}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

其中，θ是两向量之间的角（参见图 A.17）。这个积称为两向量的**标量积**（scalar product）或**点积**（dot product）。也可以将其看作为内积（inner product），特别是在张量分析中讨论标量积时。方程（A.16）对任何坐标表示都有效，并可解释为两向量平行分量的积，其中$|\mathbf{V}_2|\cos\theta$是向量$\mathbf{V}_2$在$\mathbf{V}_1$方向上的投影。

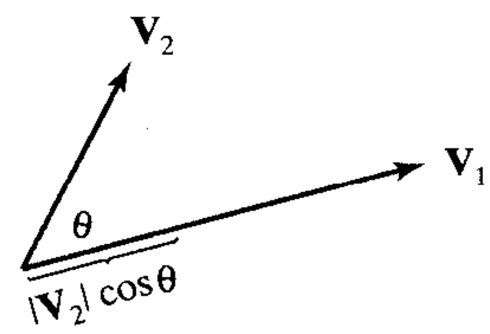


图 A.17　两向量的点积，由平行分量相乘而得

除了不依赖于坐标的标量积形式之外，我们可以在指定的坐标表示下表达这个积。对于笛卡儿参照系，标量积计算为

$$\mathbf{V}_1 \cdot \mathbf{V}_2 = V_{1x}V_{2x} + V_{1y}V_{2y} + V_{1z}V_{2z} \tag{A.17}$$

标量积是Pythagorean定理的一种推广，并且一个向量与其自身的标量积产生向量大小的平方。两向量的标量积为 0，当且仅当这两向量是互相垂直的（正交）。

点积是可交换的：

$$\mathbf{V}_1 \cdot \mathbf{V}_2 = \mathbf{V}_2 \cdot \mathbf{V}_1 \tag{A.18}$$

因为这种操作产生标量，并且点积关于向量加法是可分配的：

$$\mathbf{V}_1 \cdot (\mathbf{V}_2 + \mathbf{V}_3) = \mathbf{V}_1 \cdot \mathbf{V}_2 + \mathbf{V}_1 \cdot \mathbf{V}_3 \tag{A.19}$$

A.2.5　两向量的向量积

能产生另一向量的两向量乘法定义为

$$\mathbf{V}_1 \times \mathbf{V}_2 = \mathbf{u}|\mathbf{V}_1|\,|\mathbf{V}_2|\sin\theta, \qquad 0 \leqslant \theta \leqslant \pi \tag{A.20}$$

其中，$\mathbf{u}$是垂直于$\mathbf{V}_1$和$\mathbf{V}_2$的单位向量（量值为 1，参见图 A.18）。$\mathbf{u}$的方向则由右手定律决定：握住垂直于$\mathbf{V}_1$和$\mathbf{V}_2$所在平面的轴，使右手手指从$\mathbf{V}_1$环绕$\mathbf{V}_2$，右手拇指则指向$\mathbf{u}$方向。这个积称为两向量的**向量积**（vector product）或**叉积**（cross product），而等式（A.20）在任何坐标表示下都有效。两向量的叉积是垂直于两向量所在平面的向量，大小等于这两个向量形成的平行四边形的面积。

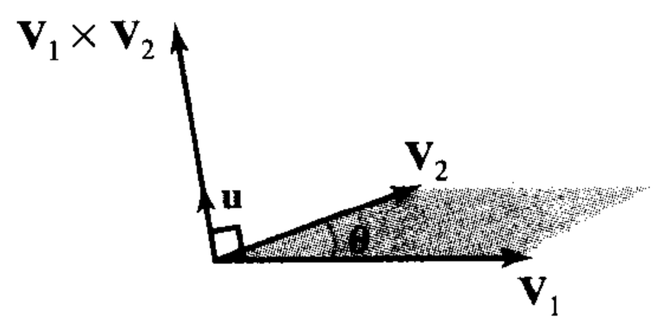


图 A.18　两向量的叉积是一向量，其方向垂直于两原始向量，而大小等于投影的平行四边形的面积

我们也可以借助指定参照系的向量分量来表达叉积。在笛卡儿坐标系，计算叉积的分量为

$$\mathbf{V}_1 \times \mathbf{V}_2 = (V_{1y}V_{2z} - V_{1z}V_{2y},\ V_{1z}V_{2x} - V_{1x}V_{2z},\ V_{1x}V_{2y} - V_{1y}V_{2x}) \tag{A.21}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

如果令 $\mathbf{u}_x$、$\mathbf{u}_y$ 和 $\mathbf{u}_z$ 表示沿 x、y 和 z 轴的单位向量（量值为1），可以借助笛卡儿分量以行列式格式写出叉积（参见附录A.5节）：

$$\mathbf{V}_1 \times \mathbf{V}_2 = \begin{vmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z \\ V_{1x} & V_{1y} & V_{1z} \\ V_{2x} & V_{2y} & V_{2z} \end{vmatrix} \tag{A.22}$$

任何两个平行向量的叉积为0。即一向量与其自身的叉积是0。同样，叉积是不满足交换律的，它是反交换的：

$$\mathbf{V}_1 \times \mathbf{V}_2 = -(\mathbf{V}_2 \times \mathbf{V}_1) \tag{A.23}$$

而且，叉积也不满足结合律：

$$\mathbf{V}_1 \times (\mathbf{V}_2 \times \mathbf{V}_3) \neq (\mathbf{V}_1 \times \mathbf{V}_2) \times \mathbf{V}_3 \tag{A.24}$$

但是，叉积关于向量加法是可分配的，即

$$\mathbf{V}_1 \times (\mathbf{V}_2 + \mathbf{V}_3) = (\mathbf{V}_1 \times \mathbf{V}_2) + (\mathbf{V}_1 \times \mathbf{V}_3) \tag{A.25}$$

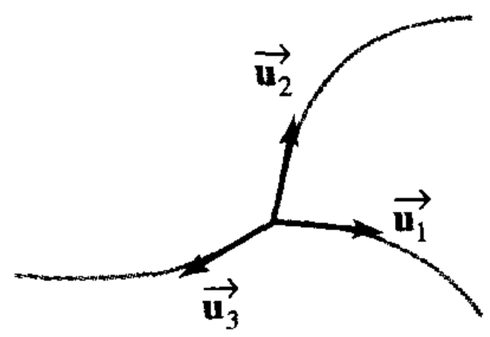
A.3 张量

张量是向量概念的推广。张量的正式定义为一个具有指定阶的量，它在从一种坐标表示转换成另一种时具有确定的转换性质。对于正交坐标系，张量的转换性质同向量一样直接。物体的多种物理性质都是张量，例如压力、张力和导电性。

张量的阶和定义张量的空间的维数决定了张量中分量（也称为元素或系数）的数量。标量和向量是张量的特例。标量是零阶张量，向量是一阶张量。基本上，张量的阶指定用于标明张量元素的下标的数量，而空间维数决定各个下标可以取值的个数。因此，标量有零个下标，向量有一个下标。有时具有一个下标的参数被误标为“一维”；具有两个下标的参数被误标为“二维”。然而，一个量的**维数**（dimension）取决于它的空间表示，而不是下标的个数。在二维空间中，单个向量下标可以取两个值；二维向量有两个分量。在三维空间中，单个向量下标可以取三个值；三维向量有三个分量。同样，一个二阶张量具有两个下标，它在三维空间中有九个分量（每个下标可以取三个值）。

A.4 基向量和度量张量

我们可以在一个 n 维参照系中用一组轴向量（记做 $\vec{\mathbf{u}}_k$，$k=1,2,...,n$）指定坐标方向，如图A.19所示。图中的轴向量位于三维曲线坐标系的原点。每个坐标轴向量在沿轴的任一点给出轴的方向。这些轴的切线向量形成线性独立的向量组，即任一轴向量不能写成其余两向量的线性组合。同时，该空间的任何其他向量均可写成轴向量的线性组合。此轴向量组称为该空间的**基**（basis），或**基向量**（base vector）组。通常，将该空间看成向量空间，而基包含最少个数的一组向量数，使该空间的任何其他向量表示为该组向量的线性组合。



图A.19 曲线坐标轴向量

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

A.4.1 决定一个坐标空间的基向量

任意空间的基向量由**位置向量**（position vector）$\vec{\mathbf{r}}$决定。位置向量是任意空间位置的向量表示。例如，在三维笛卡儿空间，任意一点(x, y, z)的位置向量是

$$\vec{\mathbf{r}} = x\mathbf{u}_x + y\mathbf{u}_y + z\mathbf{u}_z \tag{A.26}$$

其中$\mathbf{u}_x$、$\mathbf{u}_y$和$\mathbf{u}_z$是x、y和z轴的单位基向量。不同于其他坐标表示，笛卡儿基向量是与空间坐标无关的常量。因此，我们有

$$\mathbf{u}_x = \frac{\partial \vec{\mathbf{r}}}{\partial x}, \qquad \mathbf{u}_y = \frac{\partial \vec{\mathbf{r}}}{\partial y}, \qquad \mathbf{u}_z = \frac{\partial \vec{\mathbf{r}}}{\partial z} \tag{A.27}$$

类似地，对于任意其他三维空间，我们可以根据空间坐标将位置向量的表示公式化为$\vec{\mathbf{r}}(x_1, x_2, x_3)$。那么，基向量由下式确定：

$$\vec{\mathbf{u}}_k = \frac{\partial \vec{\mathbf{r}}}{\partial x_k}, \qquad k = 1, 2, 3 \tag{A.28}$$

通常，基向量$\vec{\mathbf{u}}_k$既不是常量也不是单位向量。它们是空间坐标的函数。

二维极坐标空间的位置向量是

$$\vec{\mathbf{r}} = r\cos\theta\,\mathbf{u}_x + r\sin\theta\,\mathbf{u}_y \tag{A.29}$$

极坐标基向量是

$$\begin{aligned} \vec{\mathbf{u}}_r &= \frac{\partial \vec{\mathbf{r}}}{\partial r} = \cos\theta\,\mathbf{u}_x + \sin\theta\,\mathbf{u}_y \\ \vec{\mathbf{u}}_\theta &= \frac{\partial \vec{\mathbf{r}}}{\partial \theta} = -r\sin\theta\,\mathbf{u}_x + r\cos\theta\,\mathbf{u}_y \end{aligned} \tag{A.30}$$

在这个空间，单位向量$\vec{\mathbf{u}}_r$是θ的函数。但是$\vec{\mathbf{u}}_\theta$是r和θ的函数，不是单位向量。

A.4.2 正交基

通常，基中的向量是规范化的，即每个向量的大小为1。我们用下式计算出任意三维空间的单位基向量：

$$\mathbf{u}_k = \frac{\vec{\mathbf{u}}_k}{|\vec{\mathbf{u}}_k|}, \qquad k = 1, 2, 3 \tag{A.31}$$

这一组单位向量称为**标准基**（normal basis）。同样，对于笛卡儿参照系、圆柱形参照系、球形参照系、极坐标系和其他常用参照系，坐标轴在空间中的任一点是互相垂直的，因而将基向量组看成**正交基**（orthogonal basis）。此外，如果基向量都是单位向量，则有满足下列条件的**正交标准基**（orthonormal basis）：

$$\begin{aligned} \mathbf{u}_k \cdot \mathbf{u}_k &= 1, \quad \text{对所有 } k \\ \mathbf{u}_j \cdot \mathbf{u}_k &= 0, \quad \text{对所有 } j \neq k \end{aligned} \tag{A.32}$$

尽管最常用的参照系是正交的，但非正交坐标参照系也用于某些应用中，包括相对性定律和某些数据集的可视化中。

对于二维笛卡儿系，正交标准基是

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{u}_x = (1, 0), \qquad \mathbf{u}_y = (0, 1) \tag{A.33}$$

而对于三维笛卡儿参照系的正交标准基是

$$\mathbf{u}_x = (1, 0, 0), \qquad \mathbf{u}_y = (0, 1, 0), \qquad \mathbf{u}_z = (0, 0, 1) \tag{A.34}$$

A.4.3 度量张量

对于“普通”的坐标空间（即可以在其中定义距离，通常称为黎曼（Riemannian）空间），该空间的**度量张量**（metric tensor）的分量（或系数）定义为基向量的标量积：

$$g_{jk} = \vec{\mathbf{u}}_j \cdot \vec{\mathbf{u}}_k \tag{A.35}$$

因此，度量张量是二级的，并且是对称的：$g_{jk} = g_{kj}$。度量张量有几个有用的特性。度量张量的分量可用来确定：(1)该空间两点之间的距离，(2)为了转换到另一空间的变换方程，(3)该空间内各种不同向量运算符（如梯度、散度和旋度）的分量。

在一个正交空间中：

$$g_{jk} = 0, \qquad j \neq k \tag{A.36}$$

例如，在基向量是常数单位向量的笛卡儿坐标系中，度量张量有如下分量：

$$g_{jk} = \begin{cases} 1, & \text{如果} j = k \\ 0, & \text{其他情况} \end{cases} \quad \text{（笛卡儿空间）} \tag{A.37}$$

在极坐标的单位基向量（参见方程（A.30））可以借助笛卡儿基向量表示为

$$\mathbf{g} = \begin{bmatrix} 1 & 0 \\ 0 & r^2 \end{bmatrix} \quad \text{（极坐标）} \tag{A.38}$$

对于圆柱坐标参照系，基向量是

$$\vec{\mathbf{u}}_\rho = \cos\theta\, \mathbf{u}_x + \sin\theta\, \mathbf{u}_y, \qquad \vec{\mathbf{u}}_\theta = -\rho \sin\theta\, \mathbf{u}_x + \rho \cos\theta\, \mathbf{u}_y, \qquad \vec{\mathbf{u}}_z = \mathbf{u}_z \tag{A.39}$$

而在圆柱坐标系中度量张量的矩阵表示为

$$\mathbf{g} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \rho & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{（圆柱坐标）} \tag{A.40}$$

可以写出球坐标的基向量为

$$\begin{aligned} \vec{\mathbf{u}}_r &= \cos\theta \sin\phi\, \mathbf{u}_x + \sin\theta \sin\phi\, \mathbf{u}_y + \cos\phi\, \mathbf{u}_z \\ \vec{\mathbf{u}}_\theta &= -r \sin\theta \sin\phi\, \mathbf{u}_x + r \cos\theta \sin\phi\, \mathbf{u}_y \\ \vec{\mathbf{u}}_\phi &= r \cos\theta \cos\phi\, \mathbf{u}_x + r \sin\theta \cos\phi\, \mathbf{u}_y - r \sin\phi\, \mathbf{u}_z \end{aligned} \tag{A.41}$$

使用方程（A.35）中的基向量，球坐标度量张量的矩阵表示则为

$$\mathbf{g} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & r^2 \sin^2\phi & 0 \\ 0 & 0 & r^2 \end{bmatrix} \quad \text{（球坐标）} \tag{A.42}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

A.5 矩阵

矩阵是若干量（数字、函数或数值表达式）的长方形阵列，这些量为称该矩阵的元素。下面是矩阵的一些例子：

$$\begin{bmatrix} 3.60 & -0.01 & 2.00 \\ -5.46 & 0.00 & 1.63 \end{bmatrix}, \quad \begin{bmatrix} e^x & x \\ e^{2x} & x^2 \end{bmatrix}, \quad \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}, \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{A.43}$$

我们按行数和列数标识矩阵。对于上例，自左到右顺序的矩阵为 2×3、2×2、1×3 和 3×1。当行数与列数相同时，如上述第 2 列，那么矩阵则称为方阵（square matrix）。

一般情况下，可将一个 $r \times c$ 矩阵写成：

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1c} \\ m_{21} & m_{22} & \cdots & m_{2c} \\ \vdots & \vdots & & \vdots \\ m_{r1} & m_{r2} & \cdots & m_{rc} \end{bmatrix} \tag{A.44}$$

其中，m_{jk} 表示矩阵 **M** 的元素。任何元素的第一个下标给定行数，而第二个下标给定列数。

一单行或单列的矩阵表示一个向量。因此，式（A.43）的最后两个矩阵分别为行向量（row vector）和列向量（column vector）。通常，矩阵可以看作为行向量的集合或列向量的集合。

当各种操作以矩阵形式表达时，标准的数学习惯是以列矩阵表示向量。沿用这种惯例，我们将笛卡儿坐标的三维向量的矩阵表示写成

$$\mathbf{V} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{A.45}$$

我们将使用这种矩阵表示点和向量，但必须注意它们之间的区别。我们可以很方便地将一个点看作为起始位置在单一坐标参照系的坐标原点的向量。但是点并不具有当从一个坐标系切换到另一个时保持不变的向量特性。而且，我们不能对点进行向量操作，如向量加法、点积和叉积。

A.5.1 标量乘法与矩阵加法

当矩阵 **M** 乘以一标量值 s 时，我们将该标量值乘以每个元素 m_{jk}。例如，若

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

则

$$3\,\mathbf{M} = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \end{bmatrix}$$

矩阵加法只对有相同行数 r 和相同列数 c 的矩阵有定义。对于任意两个 $r \times c$ 矩阵，其和由对应元素相加得到。例如：

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0.0 & 1.5 & 0.2 \\ -6.0 & 1.1 & -10.0 \end{bmatrix} = \begin{bmatrix} 1.0 & 3.5 & 3.2 \\ -2.0 & 6.1 & -4.0 \end{bmatrix}$$

A.5.2 矩阵乘法

两矩阵之积定义为向量点积的通用化。我们可以把 $m \times n$ 矩阵 **A** 和 $p \times q$ 矩阵 **B** 相乘，以形成矩阵积 **AB**，这当 **A** 的列数等于 **B** 的行数（即 $n = p$）时。我们通过计算 **A** 的行向量的元素与 **B** 的列向量对应元素之积的和来得到乘积矩阵。因此，对于下列乘积：

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mathbf{C} = \mathbf{A}\,\mathbf{B} \tag{A.46}$$

得到一个 $m \times q$ 矩阵 $\mathbf{C}$，其元素计算为

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{A.47}$$

在下例中，一个 3×2 矩阵和一个 2×2 矩阵相乘，生成一个 3×2 乘积矩阵：

$$\begin{bmatrix} 0 & -1 \\ 5 & 7 \\ -2 & 8 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 0\cdot 1 + (-1)\cdot 3 & 0\cdot 2 + (-1)\cdot 4 \\ 5\cdot 1 + 7\cdot 3 & 5\cdot 2 + 7\cdot 4 \\ -2\cdot 1 + 8\cdot 3 & -2\cdot 2 + 8\cdot 4 \end{bmatrix} = \begin{bmatrix} -3 & -4 \\ 26 & 38 \\ 22 & 28 \end{bmatrix}$$

以矩阵表示的向量乘法产生与点积同样的结果。当第一个向量表达为行向量，而第二个向量表达为列向量时：

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = [32]$$

这个向量积生成一个包含单个元素的矩阵（1×1 矩阵）。如果我们按相反顺序进行向量乘法，我们得 3×3 矩阵：

$$\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 4 & 8 & 12 \\ 5 & 10 & 15 \\ 6 & 12 & 18 \end{bmatrix}$$

正如上述两种向量乘积所示，矩阵乘法是不满足交换律的，即

$$\mathbf{A}\,\mathbf{B} \neq \mathbf{B}\,\mathbf{A} \tag{A.48}$$

但是，矩阵乘法关于矩阵加法是可以满足分配律的：

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{A}\,\mathbf{B} + \mathbf{A}\,\mathbf{C} \tag{A.49}$$

A.5.3　矩阵转置

一个矩阵的**转置**（transpose）$\mathbf{M}^T$ 由行和列的交换得来。例如：

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}, \qquad [a \quad b \quad c]^T = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{A.50}$$

对于矩阵积，转置是

$$(\mathbf{M}_1\mathbf{M}_2)^T = \mathbf{M}_2^T\mathbf{M}_1^T \tag{A.51}$$

A.5.4　矩阵的秩

对于方阵，我们可以组合矩阵单元来生成一个数，称为**秩**（determinant）。测量矩阵的秩对分析和解决许多问题很有用。对于一个 2×2 矩阵，**二阶秩**（second-order determinant）定义为

$$\det \mathbf{A} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21} \tag{A.52}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们递归地借助较低阶的秩来计算较高阶的秩。在计算3阶以上的秩时，我们可以选择$n \times n$矩阵的任意k列，并按下式计算秩：

$$\det \mathbf{A} = \sum_{j=1}^{n} (-1)^{j+k} a_{jk} \det \mathbf{A}_{jk} \tag{A.53}$$

其中，$\det\mathbf{A}_{jk}$是从$\mathbf{A}$删去第j行和第k列得到的子矩阵$(n-1)\times(n-1)$的秩。同样，我们可以选择任意j行，并计算秩：

$$\det \mathbf{A} = \sum_{k=1}^{n} (-1)^{j+k} a_{jk} \det \mathbf{A}_{jk} \tag{A.54}$$

采用数值法对于计算大矩阵（例如说$n > 4$）的秩会更方便。计算秩的一种途径是把矩阵分解为两个因子：$\mathbf{A} = \mathbf{LU}$，其中矩阵$\mathbf{L}$在对角线之上的所有元素为0，而矩阵$\mathbf{U}$在对角线之下的所有元素为0。然后我们计算$\mathbf{L}$和$\mathbf{U}$对角线的积，两积相乘，就可以得到$\det\mathbf{A}$。该方法是基于秩的如下特性：

$$\det(\mathbf{AB}) = (\det \mathbf{A})(\det \mathbf{B}) \tag{A.55}$$

计算秩的另一个办法基于高斯消去过程（参见A.14节）。

A.5.5 矩阵的逆

对于方阵，我们可以得到逆矩阵（inverse matrix），当且仅当矩阵的秩为非零。如果存在逆矩阵，则该矩阵称作**非奇异矩阵**（nonsingular matrix）。否则，矩阵为**奇异矩阵**（singular matrix）。对于大多数实际应用，矩阵表示一个物理操作，我们可以期望其逆矩阵存在。

一个$n \times n$方阵$\mathbf{M}$的逆矩阵记做$\mathbf{M}^{-1}$，并且

$$\mathbf{M}\mathbf{M}^{-1} = \mathbf{M}^{-1}\mathbf{M} = \mathbf{I} \tag{A.56}$$

其中，$\mathbf{I}$是单位矩阵。$\mathbf{I}$的所有对角线元素均为1，而所有其他不在对角线上元素均为0。

逆矩阵$\mathbf{M}^{-1}$的元素，可以从$\mathbf{M}$的元素计算出：

$$m_{jk}^{-1} = \frac{(-1)^{j+k} \det \mathbf{M}_{kj}}{\det \mathbf{M}} \tag{A.57}$$

其中，m_{jk}^{-1}是$\mathbf{M}^{-1}$的第j行和第k列的单元，而$\mathbf{M}_{kj}$是由矩阵$\mathbf{M}$删去第k行和第j列所得的$(n-1)\times(n-1)$子矩阵。数值法也可用于n值大的逆矩阵的秩和元素计算。

A.6 复数

按定义，**复数**（complex number）z是一个实数的有序对：

$$z = (x, y) \tag{A.58}$$

其中，x称为z的**实部**（real part），而y称为z的**虚部**（imaginary part）。复数的实部和虚部指定为

$$x = \mathrm{Re}(z), \qquad y = \mathrm{Im}(z) \tag{A.59}$$

在几何上，复数以复平面（complex plane）表示，如图A.20所示。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

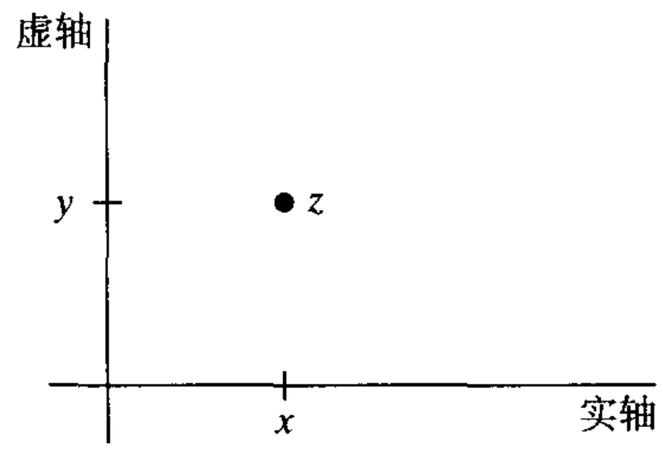


图 A.20　点 z 在复平面的位置

当 $\mathrm{Re}(z)=0$ 时，复数 z 可称为纯虚数。同样，任何实数可以表示为 $\mathrm{Im}(z)=0$ 的复数。因此，我们可以使用复数形式表示实数：

$$x=(x,0)$$

复数由方程的求解引出，例如：

$$x^2+1=0,\qquad x^2-2x+5=0$$

没有实数解。因此，建立了复数和复数算术作为实数的延伸，从而对这些方程求解。

A.6.1　基本复数算术

复数的加法、减法和标量乘法是使用与二维向量一样的规则进行的。两个复数的和为

$$z_1+z_2=(x_1,y_1)+(x_2,y_2)=(x_1+x_2,y_1+y_2)$$

按加法规则，我们可将任何复数写成和的形式：

$$z=(x,y)=(x,0)+(0,y)$$

复数的乘法定义为

$$(x_1,y_1)(x_2,y_2)=(x_1x_2-y_1y_2,x_1y_2+x_2y_1) \tag{A.60}$$

当虚部为 0 时，这个对复数的定义给出与实数乘法同样的结果。

$$(x_1,0)(x_2,0)=(x_1x_2,0)$$

A.6.2　虚单位

纯虚数(0, 1)称为**虚单位**（imaginary unit），并记为

$$i=(0,1) \tag{A.61}$$

（电气工程师常用符号 j 作为虚单位，因为符号 i 用来表示电流。）

从复数乘法规则，我们有

$$i^2=(0,1)(0,1)=(-1,0)$$

所以，i^2 是实数 -1，并且

$$i=\sqrt{-1} \tag{A.62}$$

采用复数乘法规则，可以将任何纯虚数写成

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$z = iy = (0, y)$$

所以，一个普通的复数可以表示为

$$z = x + iy \tag{A.63}$$

根据 i 的定义，我们可以确认这种表示符合复数加减法和乘法的规则。

A.6.3 共轭复数和复数的模

同复数关联的另一概念是共轭复数（complex conjugate）：

$$\bar{z} = x - iy \tag{A.64}$$

因此，共轭复数$\bar{z}$是 z 关于 x（实）轴的映射。

复数的模（modulus）或绝对值（absolute value）定义为

$$|z| = \sqrt{z\bar{z}} = \sqrt{x^2 + y^2} \tag{A.65}$$

这给出在复平面上原点到 z 点的距离，有时称为复数的“向量长度”。因而，复数的绝对值就是 Pythagorean 定理在复平面上的表示。

A.6.4 复数除法

在计算复数除法时，可以通过将分子、分母同时乘以分母的共轭复数来化简表达式。然后，使用乘法规则决定商复数的分量。两复数除法的实部和虚部可以使用下列等式进行计算：

$$\begin{aligned}\frac{z_1}{z_2} &= \frac{z_1\overline{z_2}}{z_2\overline{z_2}} \\ &= \frac{(x_1, y_1)(x_2, -y_2)}{x_2^2 + y_2^2}, \\ &= \left(\frac{x_1x_2 + y_1y_2}{x_2^2 + y_2^2}, \frac{x_2y_1 - x_1y_2}{x_2^2 + y_2^2}\right)\end{aligned} \tag{A.66}$$

A.6.5 复数的极坐标表示

借助极坐标表示复数的实部和虚部可以极大简化复数的乘法和除法（参见图 A.21）：

$$z = r(\cos\theta + i\sin\theta) \tag{A.67}$$

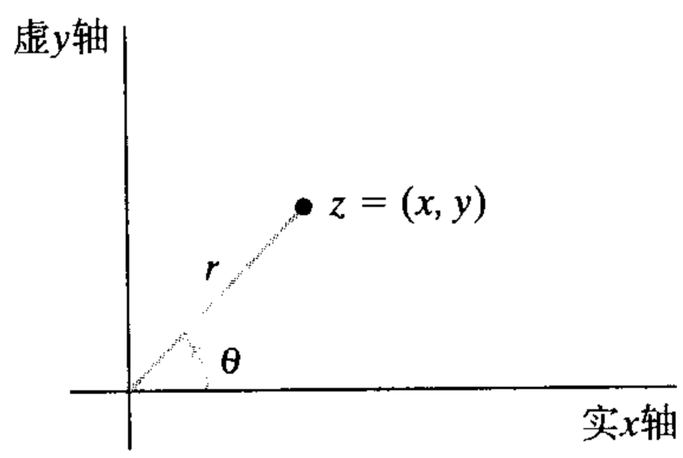


图 A.21 复数 z 的极坐标位置

我们也以可将 z 的极坐标形式写成

$$z = re^{i\theta} \tag{A.68}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

其中，e 是自然对数的底($e \approx 2.718\,281\,828$)，并且

$$e^{i\theta} = \cos\theta + i\sin\theta \tag{A.69}$$

这就是欧拉公式（Euler' s formula）。

使用极坐标形式，两个复数的乘积可以通过绝对值相乘和角度相加得到，如下所示：

$$z_1 z_2 = r_1 r_2 e^{i(\theta_1+\theta_2)} \tag{A.70}$$

类似地，两个复数的商可以通过绝对值相除和角度相减得到，如下所示：

$$\frac{z_1}{z_2} = \frac{r_1}{r_2} e^{i(\theta_1-\theta_2)} \tag{A.71}$$

我们也可以利用极坐标表示来计算复数的根。复数的 n 次方根按下式计算：

$$\sqrt[n]{z} = \sqrt[n]{r}\left[\cos\left(\frac{\theta+2k\pi}{n}\right) + i\sin\left(\frac{\theta+2k\pi}{n}\right)\right], \quad k = 0, 1, 2, \ldots, n-1 \tag{A.72}$$

这些根位于中心在复平面的原点、半径为 $\sqrt[n]{r}$ 的圆周上，并形成圆内接 n 边正多边形的顶点。

A.7 四元数

复数概念由**四元数**（quaternion）扩充到高维。四元数是一个复数，带一个实部和三个虚部，写为

$$q = s + ia + jb + kc \tag{A.73}$$

其中，虚数项系数 a、b 和 c 是实数；参数 s 也是实数，称为标量部分（scalar part）。参数 i、j、k 定义有如下特性：

$$i^2 = j^2 = k^2 = -1, \qquad ij = -ji = k \tag{A.74}$$

从这些特性，可以得到

$$jk = -kj = i, \qquad ki = -ik = j \tag{A.75}$$

标量乘法使用类似于向量和复数的相应操作来定义。即四元数四个分量的每一个乘以标量值。类似地，四元数加法定义为

$$q_1 + q_2 = (s_1 + s_2) + i(a_1 + a_2) + j(b_1 + b_2) + k(c_1 + c_2) \tag{A.76}$$

两个四元数的乘法是利用方程（A.74）和方程（A.75）的操作进行的。

四元数的排列法也与复数排列有类似的形式：

$$q = (s, \mathbf{v}) \tag{A.77}$$

其中，$\mathbf{v}$ 是向量(a，b，c)。按这种记法，四元数加法表示为

$$q_1 + q_2 = (s_1 + s_2, \mathbf{v}_1 + \mathbf{v}_2) \tag{A.78}$$

四元数乘法则可借助向量点积和叉积表示为

$$q_1 q_2 = (s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2) \tag{A.79}$$

与复数操作类似，四元数的平方值使用分量的平方和定义为

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$|q|^2 = s^2 + \mathbf{v} \cdot \mathbf{v} \tag{A.80}$$

而四元数的逆是

$$q^{-1} = \frac{1}{|q|^2}(s, -\mathbf{v}) \tag{A.81}$$

因此

$$qq^{-1} = q^{-1}q = (1, 0)$$

A.8 非参数表示

当我们直接用正在使用的参照系坐标来描述对象时，该表示称为**非参数**（nonparametric）表示。例如，我们可以使用下列笛卡儿函数之一来表示一个曲面：

$$f_1(x, y, z) = 0, \quad 或 \quad z = f_2(x, y) \tag{A.82}$$

方程（A.82）中的第一种形式给出了曲面的隐式（implicit）表达式，而第二种形式给出了显式（explicit）表示。在显式表示中，x 和 y 称为自变量，而 z 称为因变量。

类似地，我们可以利用非参数形式，通过两个曲面函数的交来表示一条三维曲线。或者，我们可以使用一对函数：

$$y = f(x) \quad 和 \quad z = g(x) \tag{A.83}$$

表示该曲线。其中选定坐标 x 为自变量。当我们通过 x 的值从一个线端点至另一端点逐步变化时，由方程（A.83）来确定因变量 y 和 z 的值。

非参数表示用于在给定参照系中描述物体，但在图形算法中，它们有某些不足。如果要平滑绘图，必须在 $f(x)$ 或 $g(x)$ 的一阶导数（斜率）大于 1 时改变自变量。这意味着必须连续检验导数值来决定是否需要改变自变量和因变量的角色。同样，方程（A.83）有时为表示多值函数提供了一个不方便的形式。例如，中心在 xy 平面原点的圆的隐式方程是

$$x^2 + y^2 - r^2 = 0$$

而 y 的显式表示是多值函数：

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

一般地，以图形算法描述对象的更方便的表示是使用参数方程。

A.9 参数表示

我们可以根据描述一个物体的坐标位置所需的参数的个数来分类物体。欧几里得曲线是一维物体，欧几里得曲面或平面是二维物体。当我们用维度参数描述物体时，该描述称为**参数表示**（parametric representation）。

可以使用下列向量点函数表示沿三维曲线的路径上任何位置的笛卡儿描述的参数形式：

$$\mathbf{P}(u) = (x(u), y(u), z(u)) \tag{A.84}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

其中每一个笛卡儿坐标都是参数 u 的函数。通常，坐标方程可以根据参数 u 定义在从 0.0 到 1.0 的单位区间而建立。例如，中心在坐标原点、半径为 r、位于 xy 平面的圆，可以用参数形式定义为

$$x(u) = r\cos(2\pi u), \qquad y(u) = r\sin(2\pi u), \qquad z(u) = 0, \qquad 0 \leqslant u \leqslant 1 \tag{A.85}$$

因为该曲线是定义在 xy 平面上的，因此可以消去恒等于 0 的 $z(u)$。

类似地，曲面上的坐标位置可以使用具有两个参数 u 和 v 的笛卡儿向量点函数表示：

$$\mathbf{P}(u, v) = (x(u, v), y(u, v), z(u, v)) \tag{A.86}$$

其中，x、y 和 z 的笛卡儿坐标值表示为参数 u 和 v 的函数。例如，中心在坐标原点、半径为 r 的球曲面可以使用下列方程描述：

$$\begin{aligned} x(u, v) &= r\cos(2\pi u)\sin(\pi v) \\ y(u, v) &= r\sin(2\pi u)\sin(\pi v) \qquad 0 \leqslant u, v \leqslant 1 \\ z(u, v) &= r\cos(\pi v) \end{aligned} \tag{A.87}$$

参数 u 描述该曲面的常数纬度线，而参数 v 描述常数经度线。与曲线一样，将参数 u 和 v 定义在从 0 到 1.0 的范围。保持这些参数之一固定不变，同时在范围 0 到 1.0 的整个子区间上改变另一个参数，我们就可绘制任何球面部分的纬度线和经度线（参见图 A.22）。

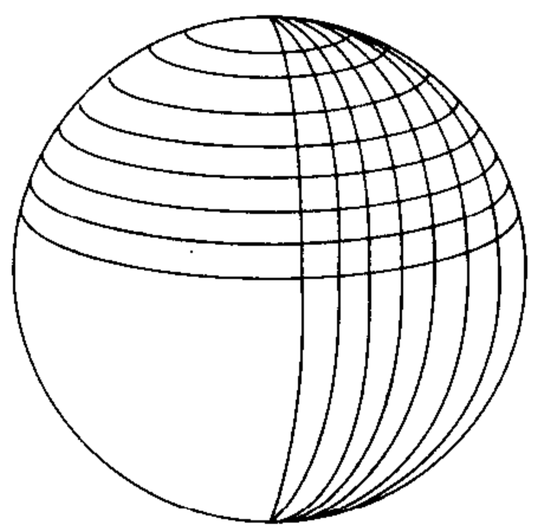


图 A.22　使用方程（A.87）中常数 u 线和常数 v 线描述的球面部分

A.10　变化率算子

对于具有一个自变量的连续函数，例如 $f(x)$，我们用 $f(x)$ 对 x 的导数（derivative）来计算该函数在任意 x 值处的变化率。导数函数定义为

$$\frac{df}{dx} \equiv \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{A.88}$$

此定义是使用数值方法解决有关变化率问题的基础。常用函数（如多项式函数和三角函数）的导数形式可以在导数表中查到。对于仅涉及简单函数的变化率问题，通常可以得到闭合形式的解。然而，在多数情况下，我们必须使用数值方法来求解变化率问题。

对于具有多个变量的函数，对某个变量的变化率运算称为偏导数（partial derivative）。例如，对于函数 $f(x, y, z, t)$，我们可以确定函数对任意一个坐标方向（x、y、z 和时间 t）的变化率。对特定自变量的偏导数也由方程（A.88）定义，其中所有其他自变量保持不变。例如，f 对时间 t 的偏导数定义如下：

$$\frac{\partial f}{\partial t} \equiv \lim_{\Delta t \to 0} \frac{f(x, y, z, t + \Delta t) - f(x, y, z, t)}{\Delta t}$$

它在某些特定的空间位置和时间处求值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

很多偏导数算子由于使用非常频繁，因而具有特殊的名称，例如梯度算子、拉普拉斯（Laplace）算子、散度算子和旋度算子。这些算子可以在多种应用中发挥作用，例如确定物体的几何形状和方向、描述物体在某些情况下的行为、计算电磁辐射效果以及分析在科学可视化研究中的数据集。

A.10.1 梯度算子

具有以下三个笛卡儿分量的向量算子称为梯度算子：

$$\text{grad} = \nabla \equiv \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \tag{A.89}$$

符号∇称为nabla、del或梯度算子。梯度算子的一个重要用处是计算曲面的法向量。当曲面用非参数表示$f(x, y, z)$ = 常量来描述时，曲面上任意位置的法向量可计算如下：

$$\mathbf{N} = \nabla f \quad \text{（非参数曲面表示的法向量）} \tag{A.90}$$

例如，一个半径为r的球面可以用非参数笛卡儿表示$f(x, y, z) = x^2 + y^2 + z^2 = r^2$来描述，并且$f$的梯度产生曲面法向量$(2x, 2y, 2z)$。但是，如果曲面用参数点函数$\mathbf{P}(u, v)$来表示，我们则使用向量叉积来确定曲面法向量：

$$\mathbf{N} = \frac{\partial \mathbf{P}}{\partial u} \times \frac{\partial \mathbf{P}}{\partial v} \quad \text{（参数曲面表示的法向量）} \tag{A.91}$$

A.10.2 方向算子

我们也可以用梯度算子和向量点积来形成一个标量积，称为函数f的方向算子（directional derivative）：

$$\frac{\partial f}{\partial u} = \mathbf{u} \cdot \nabla f \tag{A.92}$$

上式给出用单位向量$\mathbf{u}$指定的f在一个方向上的变化率。举例来说，我们确定球面函数$f = x^2 + y^2 + z^2$在z方向上的方向导数如下：

$$\frac{\partial f}{\partial z} = \mathbf{u}_z \cdot \nabla f = 2z$$

其中$\mathbf{u}_z$是沿正z轴的单位向量。并且，对于xy平面的单位向量：

$$\mathbf{u} = \frac{1}{\sqrt{2}}\mathbf{u}_x + \frac{1}{\sqrt{2}}\mathbf{u}_y$$

根据方程（A.92）计算的f的方向导数如下：

$$\frac{\partial f}{\partial u} = \frac{1}{\sqrt{2}}\frac{\partial f}{\partial x} + \frac{1}{\sqrt{2}}\frac{\partial f}{\partial y} = \sqrt{2}x + \sqrt{2}y$$

A.10.3 梯度算子的一般形式

在任何三维正交坐标系内，我们可以用下式计算梯度算子的分量：

$$\nabla = \sum_{k=1}^{3} \frac{\mathbf{u}_k}{\sqrt{g_{kk}}} \frac{\partial}{\partial x_k} \tag{A.93}$$

其中，每个$\mathbf{u}_k$表示在x_k坐标方向上的单位基向量，并且g_{kk}是该空间的度量张量的对角线元素。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

A.10.4 拉普拉斯算子

我们可以用梯度算子和向量点积形成一个标量微分算子，称为拉普拉斯算子。它的笛卡儿坐标形式如下：

$$\nabla^2 = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{A.94}$$

符号∇^2常常称为梯度平方、del平方、或nabla平方。在任何三维正交坐标系内，我们用下式计算函数$f(x, y, z)$的拉普拉斯算子：

$$\nabla^2 f = \frac{1}{\sqrt{g_{11}g_{22}g_{33}}}\left[\frac{\partial}{\partial x_1}\left(\frac{\sqrt{g_{22}g_{33}}}{\sqrt{g_{11}}}\frac{\partial f}{\partial x_1}\right)+\frac{\partial}{\partial x_2}\left(\frac{\sqrt{g_{33}g_{11}}}{\sqrt{g_{22}}}\frac{\partial f}{\partial x_2}\right)+\frac{\partial}{\partial x_3}\left(\frac{\sqrt{g_{11}g_{22}}}{\sqrt{g_{33}}}\frac{\partial f}{\partial x_3}\right)\right] \tag{A.95}$$

许多应用都会使用涉及拉普拉斯算子的方程，如对电磁辐射效果的描述。

A.10.5 散度算子

将梯度算子和一个向量函数用向量点积结合形成一个称为向量散度(divergence of a vector)的标量值，它的笛卡儿坐标形式如下：

$$\operatorname{div}\mathbf{V} = \nabla \cdot \mathbf{V} = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z} \tag{A.96}$$

其中，V_x、V_y和V_z是向量$\mathbf{V}$的笛卡儿分量。散度是对一个向量函数（如电场）在空间内一点处增加或减少率的量度。

在任何三维正交坐标系内，一个向量$\mathbf{V}$的散度计算如下：

$$\operatorname{div}\mathbf{V} = \nabla \cdot \mathbf{V} = \frac{1}{\sqrt{g_{11}g_{22}g_{33}}}\left[\frac{\partial}{\partial x_1}(\sqrt{g_{22}g_{33}}V_1)+\frac{\partial}{\partial x_2}(\sqrt{g_{33}g_{11}}V_2)+\frac{\partial}{\partial x_3}(\sqrt{g_{11}g_{22}}V_3)\right] \tag{A.97}$$

其中V_1、V_2和V_3是向量$\mathbf{V}$对应于坐标方向x_1、x_2和x_3的分量，并且g_{kk}是度量张量的对角线元素。

A.10.6 旋度算子

另一个很有用的微分算子是向量的旋度(curl of a vector)。将梯度算子和一个向量函数用向量叉积结合形成旋度算子的向量值，它的笛卡儿分量形式如下：

$$\operatorname{curl}\mathbf{V} = \nabla \times \mathbf{V} = \left(\frac{\partial V_z}{\partial y} - \frac{\partial V_y}{\partial z}, \frac{\partial V_x}{\partial z} - \frac{\partial V_z}{\partial x}, \frac{\partial V_y}{\partial x} - \frac{\partial V_x}{\partial y}\right) \tag{A.98}$$

旋度算子给出对向量的量（例如电磁辐射的散射）的旋转效果的度量。在任何三维正交坐标系内，我们用下面的行列式来表示旋度的度量张量分量。

$$\operatorname{curl}\mathbf{V} = \nabla \times \mathbf{V} = \frac{1}{\sqrt{g_{11}g_{22}g_{33}}}\begin{vmatrix} \sqrt{g_{11}}\,\mathbf{u}_1 & \sqrt{g_{22}}\,\mathbf{u}_2 & \sqrt{g_{33}}\,\mathbf{u}_3 \\ \dfrac{\partial}{\partial x_1} & \dfrac{\partial}{\partial x_2} & \dfrac{\partial}{\partial x_3} \\ \sqrt{g_{11}}V_1 & \sqrt{g_{22}}V_2 & \sqrt{g_{33}}V_3 \end{vmatrix} \tag{A.99}$$

其中向量$\mathbf{u}_k$是该空间的单位基向量，并且g_{kk}是度量张量的对角线元素。

A.11 变化率积分变换定理

在很多应用中，我们会遇到涉及对空间中的某个区域进行积分（累加）的变化率运算的问题。积分可能沿着一条线的路径、穿越一个曲面或遍及空间中的一个体。通常这类问题可以通过积分定

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

理来简化。积分定理可以将面积分转换为线积分、线积分转换为面积分、体积分转换为面积分，或将面积分转换为体积分。这些积分定理在求解广泛的实际问题时非常重要。

A.11.1 斯托克斯定理

对于一个在曲面上定义的连续向量函数 $\mathbf{F}(x, y, z)$，**斯托克斯定理**（Stokes' s theorem）表明 $\mathbf{F}$ 的旋度的垂直分量的积分等于绕该曲面的周长曲线 C 进行的线积分。即

$$\iint_{\text{surf}} (\text{curl}\,\mathbf{F}) \cdot \mathbf{n}\, dA = \oint_C \mathbf{F} \cdot \mathbf{r}\, ds \tag{A.100}$$

其中边界 C 必须是分段光滑的，也就是说，C 必须是一条连续的曲线或者由有限个连续段（如直线段或圆弧）组成。$\mathbf{n}$ 是任意点的单位曲面法向量。dA 是曲面面积的微分元。$\mathbf{r}$ 是边界曲线 C 上任意点的单位切向量。ds 是沿着 C 的一个微分线段。如图 A.23 所示，绕着 C 的积分方向是在从曲面的前方观察曲面时（参见 3.15 节）的逆时针方向。

A.11.2 格林平面定理

考虑 xy 平面上的一个被分段光滑曲线 C 包围的区域，**格林平面定理**（Green' s plane theorem）的笛卡儿形式为

$$\iint_{\text{area}} \left(\frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y} \right) dx\, dy = \oint_C (f_1\, dx + f_2\, dy) \tag{A.101}$$

这里，$f_1(x, y)$和$f_2(x, y)$是定义在整个由曲线 C 包围的平面区域上的连续函数。绕 C 进行积分的方向是逆时针方向。格林定理也可以应用在内部有洞的区域上，如图 A.24 所示，那么必须按顺时针方向绕内部边界曲线进行积分。

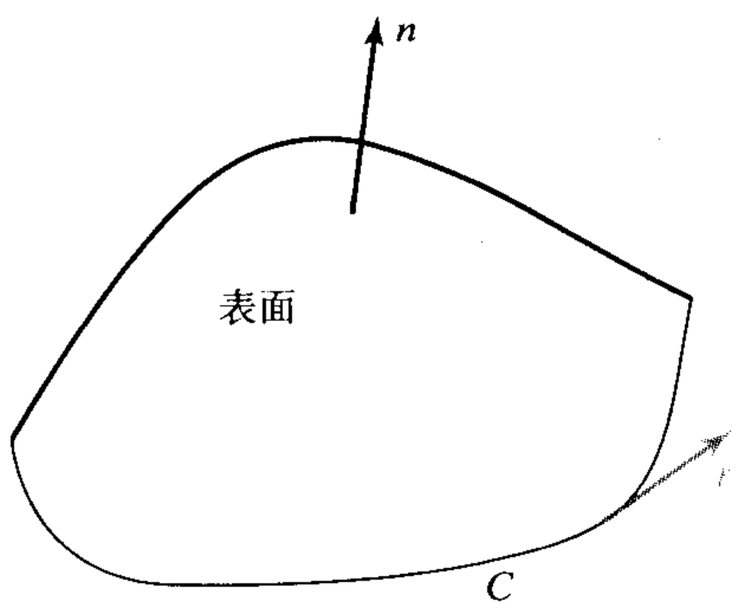


图 A.23 当从外部区域观察曲面时，斯托克斯定理中绕边界曲线 C 的积分是逆时针方向

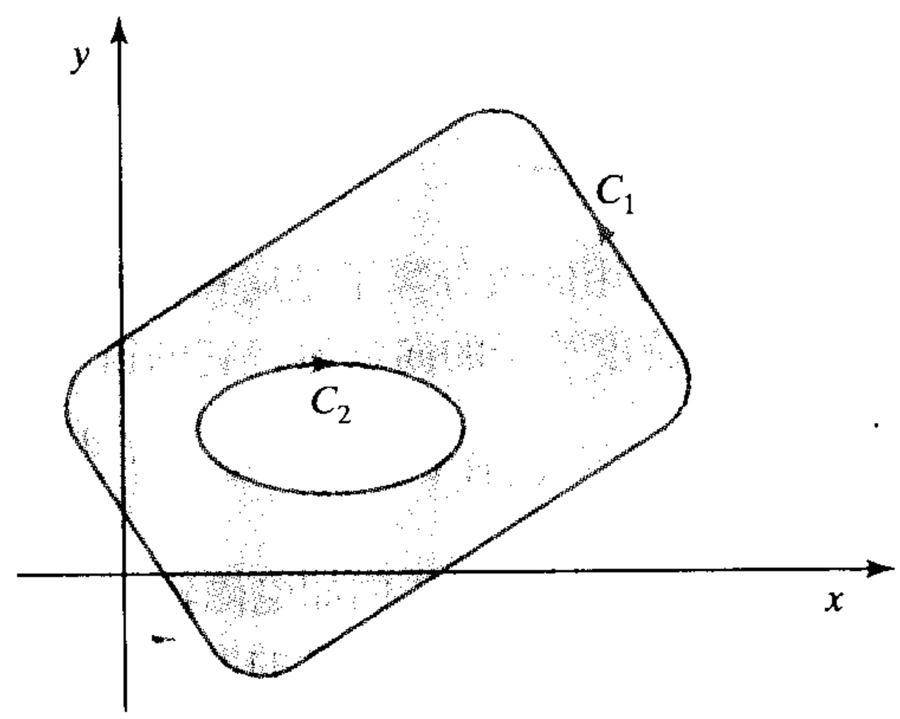


图 A.24 格林平面定理的线积分求值采用保证内部区域（阴影）总在左侧的方式来遍历边界曲线 C_1 和 C_2

格林平面定理是斯托克斯定理的一个特例。我们定义一个向量函数$\mathbf{F}$，其笛卡儿分量为$(f_1, f_2, 0)$。则格林定理可以写为如下向量形式：

$$\iint_{\text{area}} (\text{curl}\,\mathbf{F}) \cdot \mathbf{u}_z\, dA = \oint_C \mathbf{F} \cdot \mathbf{r}\, ds \tag{A.102}$$

其中 u_z 是垂直于 xy 平面（沿 z 方向）的单位向量，$dA = dxdy$，其他参数与方程（A.100）中的相同。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

设$f_1=0$并且$f_2=x$，我们可以根据格林平面定理来计算平面区域A的面积：

$$A=\iint_{\text{area}} dx\,dy=\oint_C x\,dy \tag{A.103}$$

类似地，如果设$f_1=-y$并且$f_2=0$，我们有

$$A=\iint_{\text{area}} dx\,dy=-\oint_C y\,dx \tag{A.104}$$

将上述两个面积方程相加，我们得到

$$A=\frac{1}{2}\oint_C (x\,dy-y\,dx) \tag{A.105}$$

我们也可以将笛卡儿的面积表达式转换为下列极坐标形式：

$$A=\frac{1}{2}\oint_C r^2\,d\theta \tag{A.106}$$

格林平面定理还可以表示为其他多种有用的形式。例如，设$f_1=\partial f/\partial y$并且$f_2=\partial f/\partial x$，对于某些连续函数$f$，我们有

$$\iint_{\text{area}} \nabla^2 f\,dx\,dy=\oint_C \frac{\partial f}{\partial n}\,ds \tag{A.107}$$

其中$\partial f/\partial n$是f沿边界曲线C的外法向量方向上的方向导数。

A.11.3　散度定理

前面两个定理给出了在线积分和面积分之间转换的方法，而**散度定理**（divergence theorem）给出了在面积分和体积分之间转换的公式。这个定理也称为**格林空间定理**（Green' s theorem in space）或**高斯定理**（Gauss' s theorem）。对于一个在空间体中定义的三维连续向量函数$\mathbf{F}$，散度定理的向量形式如下：

$$\iiint_{\text{vol}} \operatorname{div}\mathbf{F}\,dV=\iint_{\text{surf}} \mathbf{F}\cdot\mathbf{n}\,dA \tag{A.108}$$

其中dV是微分体元素，$\mathbf{n}$是包围曲面的法向量，dA是曲面面积的微分元。

我们可以利用散度定理得出几个有用的积分变换。例如，设$\mathbf{F}=\nabla f$，其中f是连续的三维函数，则我们得到方程（A.107）的体的形式：

$$\iiint_{\text{vol}} \nabla^2 f\,dV=\iint_{\text{surf}} \frac{\partial f}{\partial n}\,dA \tag{A.109}$$

在此方程中，$\partial f/\partial n$是f在曲面法向量方向上的方向导数。

从散度定理中可以导出利用曲面积分计算空间中一个区域的体积的公式。根据如何表示向量函数$\mathbf{F}$，我们可以得到下列任意一种曲面积分的笛卡儿形式：

$$\begin{aligned}V&=\iiint_{\text{vol}} dx\,dy\,dz\\&=\iint_{\text{surf}} x\,dy\,dz=\iint_{\text{surf}} y\,dz\,dx=\iint_{\text{surf}} z\,dx\,dy\\&=\frac{1}{3}\iint_{\text{surf}} (x\,dy\,dz+y\,dz\,dx+z\,dx\,dy)\end{aligned} \tag{A.110}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

A.11.4 格林变换方程

从散度定理中还可以推出很多其他积分变换。下面两种积分变换方程通常被称为**格林变换方程**（Green's transformation equations）、**格林第一和第二公式**（Green' s first and second formulas）或**格林恒等式**（Green's identities）：

$$\iiint_{\text{vol}} (f_1 \nabla^2 f_2 + \nabla f_1 \cdot \nabla f_2)\, dV = \iint_{\text{surf}} f_1 \frac{\partial f_2}{\partial n}\, dA \tag{A.111}$$

$$\iiint_{\text{vol}} (f_1 \nabla^2 f_2 - f_2 \nabla^2 f_1)\, dV = \iint_{\text{surf}} \left(f_1 \frac{\partial f_2}{\partial n} - f_2 \frac{\partial f_1}{\partial n} \right) dA \tag{A.112}$$

其中，f_1 和 f_2 是三维连续标量函数，$\partial f_1 / \partial n$ 和 $\partial f_2 / \partial n$ 是它们在曲面法向量方向上的方向导数。

A.12 多边形的面积和质心

我们可以使用A.11节中的积分变换来为计算机图形应用计算物体的各种性质。在涉及几何变换、仿真、系统设计和动画的程序中常常用到多边形的面积和质心。

A.12.1 多边形的面积

根据方程（A.103），我们可以将多边形的笛卡儿坐标用参数表示，然后沿着多边形的一周进行线积分来计算多边形的面积。一个在 xy 平面（参见图A.25）、具有 n 条边和 n 个顶点的多边形的参数方程表示如下：

$$\begin{aligned} x &= x_k + (x_{k+1} - x_k)u \\ y &= y_k + (y_{k+1} - y_k)u \end{aligned} \qquad 0 \leqslant u \leqslant 1, \qquad k = 1, 2, \ldots, n \tag{A.113}$$

其中，$x_{n+1} = x_1$ 并且 $y_{n+1} = y_1$。

将微分表达式 $dy = (y_{k+1} - y_k)du$ 和 x 的参数表示代入方程（A.103），我们有

$$\begin{aligned} A &= \oint_C x\, dy \\ &= \sum_{k=1}^{n} \int_0^1 [x_k + (x_{k+1} - x_k)u](y_{k+1} - y_k)\, du \\ &= \sum_{k=1}^{n} (y_{k+1} - y_k)[x_k + (x_{k+1} - x_k)/2] \\ &= \frac{1}{2} \sum_{k=1}^{n} (x_k y_{k+1} - x_k y_k + x_{k+1} y_{k+1} - x_{k+1} y_k) \end{aligned} \tag{A.114}$$

对于每条线段，和中的第二项和第三项被后继线段中具有相反符号的相应项消去。因而，多边形的面积计算为

$$A = \frac{1}{2} \sum_{k=1}^{n} (x_k y_{k+1} - x_{k+1} y_k) \tag{A.115}$$

A.12.2 多边形的质心

根据定义，质心是一个密度均匀的物体（即物体上所有点的质量相同）的质量中心的位置。因此，质心的坐标就是多边形边界内所有点的坐标的平均值。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

对于简单的多边形，可以通过计算所有顶点坐标的均值来得到质心坐标。但是，一般来说，因为忽略了多边形内其他点的位置，顶点的均值并不是准确的质心位置。如图 A.26 所示，顶点的均值靠近顶点集中的一侧，而质心则位于整个多边形区域的中心。

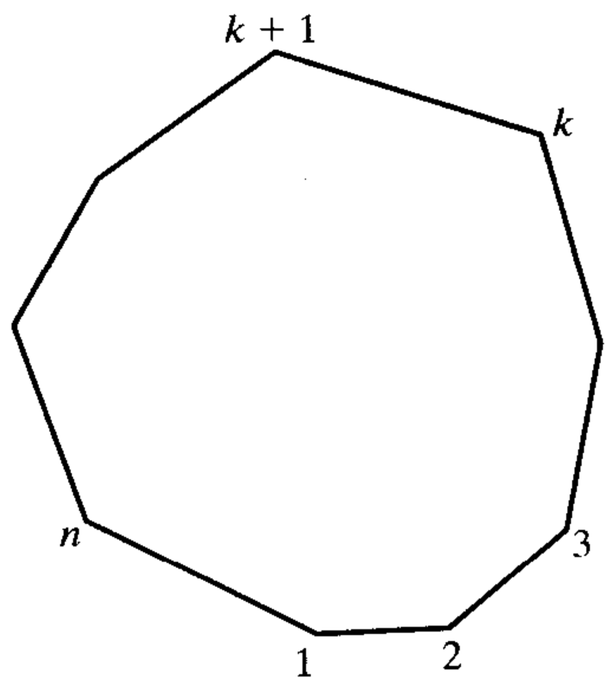


图 A.25　在 xy 平面内由 n 个顶点定义的多边形

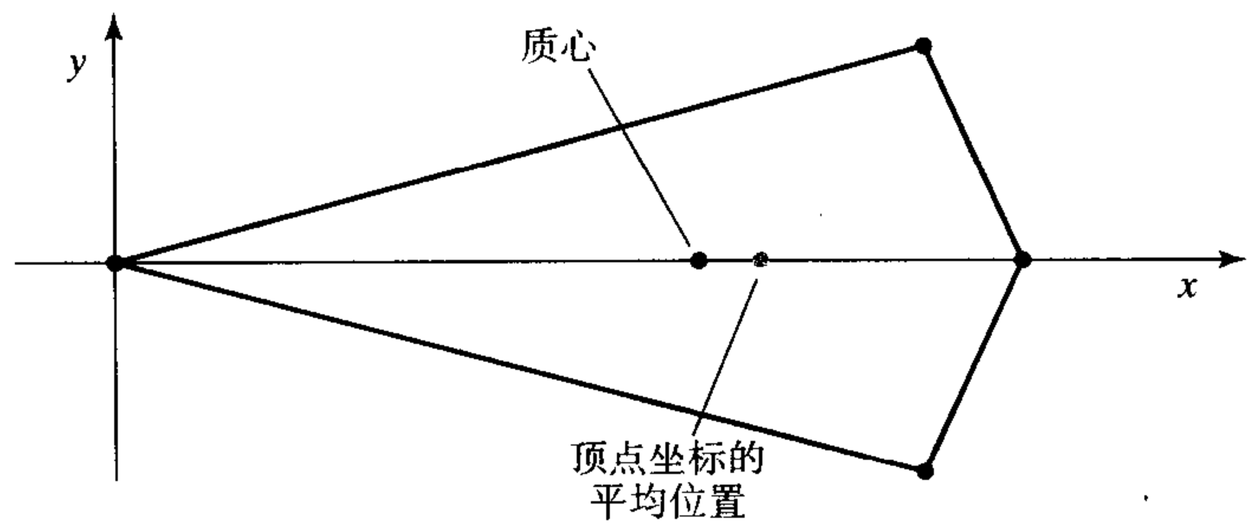


图 A.26　多边形质心的坐标位置和顶点坐标的平均位置

对于一个在 xy 平面的多边形，我们通过计算多边形边界内所有点的坐标的平均值来作为它的质心位置$(\overline{x}, \overline{y})$：

$$\overline{x} = \frac{1}{A}\iint_{\text{area}} x\,dx\,dy = \frac{\mu_x}{A}$$
$$\overline{y} = \frac{1}{A}\iint_{\text{area}} y\,dx\,dy = \frac{\mu_y}{A} \tag{A.116}$$

其中，μ_x 和 μ_y 分别称为关于 x 轴和 y 轴的区域的力矩，假设该区域在单位面积上具有单位质量。

我们可以用与计算多边形面积同样的方法来计算多边形的各个力矩。根据格林定理可以得到等于面积分的线积分，并且用多边形边界上的笛卡儿坐标的参数表示来计算线积分。格林平面定理（参见方程（A.101））指出

$$\iint_{\text{area}} \left(\frac{\partial f_2}{\partial x} - \frac{\partial f_1}{\partial y}\right) dx\,dy = \oint_C (f_1\,dx + f_2\,dy) \tag{A.117}$$

为计算 μ_x，我们在上述变换中取 $f_2 = \frac{1}{2}x^2$ 和 $f_1 = 0$，因而

$$\mu_x = \iint_{\text{area}} x\,dx\,dy = \frac{1}{2}\oint_C x^2\,dy \tag{A.118}$$

根据方程（A.113）中的多边形边界的参数表示，对于 n 条边界中的每一条，记为 $k = 1, 2, \ldots, n$，我们有

$$x^2 = x_k^2 + 2x_k(x_{k+1} - x_k)u + (x_{k+1} - x_k)^2u^2$$

和

$$dy = (y_{k+1} - y_k)\,du$$

因此

$$\begin{aligned}\mu_x &= \sum_{k=1}^{n} \frac{y_{k+1} - y_k}{2} \int_0^1 \left[x_k^2 + 2x_k(x_{k+1} - x_k)u + (x_{k+1} - x_k)^2u^2\right] du \\ &= \frac{1}{6}\sum_{k=1}^{n}(x_{k+1} + x_k)(x_k y_{k+1} - x_{k+1}y_k)\end{aligned} \tag{A.119}$$

为了计算 μ_y，我们将 $f_1 = -\frac{1}{2}y^2$ 和 $f_2 = 0$ 代入格林定理，可以得到

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\mu_y = \iint_{\text{area}} y\,dx\,dy = \frac{1}{2}\oint_C y^2\,dx \tag{A.120}$$

使用方程（A.113）中的参数表示来计算线积分，我们得到

$$\mu_y = \frac{1}{6}\sum_{k=1}^{n}(y_{k+1}+y_k)(x_k y_{k+1} - x_{k+1}y_k) \tag{A.121}$$

给定任意一组多边形顶点，我们使用方程（A.116）中的 A、μ_x 和 μ_y 的参数表示来计算多边形的质心。因为表达式 $(x_k y_{k+1} - x_{k+1} y_k)$ 在 A、μ_x 和 μ_y 的计算中都存在，对于每条线段，我们只需要计算一次。

A.13 多面体的计算性质

我们用与处理多边形类似的方法来获取多面体的性质。不同的是，现在要计算的是一个空间的体，而不是一个面。多面体的质心也是通过计算多面体内所有位置的坐标的平均值而得到的。

任意一个空间区域的体积在笛卡儿坐标系的定义为

$$V = \iiint_{\text{vol}} dx\,dy\,dz \tag{A.122}$$

可以使用方程（A.110）中的任意一种变换将这个积分转换为面积分。一个多面体的面积分可以使用各个面上位置的参数表示来计算。

计算多面体质心位置的方法与处理多边形的类似。根据定义，空间中一个区域（在单位体积上具有单位质量）的质心的笛卡儿坐标是在此区域内所有位置的平均值：

$$\begin{aligned}
\bar{x} &= \frac{1}{V}\iiint_{\text{vol}} x\,dx\,dy\,dz = \frac{\mu_x}{V}\\
\bar{y} &= \frac{1}{V}\iiint_{\text{vol}} y\,dx\,dy\,dz = \frac{\mu_y}{V}\\
\bar{z} &= \frac{1}{V}\iiint_{\text{vol}} z\,dx\,dy\,dz = \frac{\mu_z}{V}
\end{aligned} \tag{A.123}$$

同样，我们将体积分转换为面积分，代入笛卡儿坐标的参数表示，并且在多面体的每个面上计算面积分。

A.14 数值方法

在计算机图形算法中，常常需要求解线性方程组、非线性方程组、积分方程和其他函数形式。同样为使离散的数据点集可视化，可以显示一条连续曲线或曲面函数来逼近该数据点集。这一节，我们简要地概述求解各种数值问题的某些公共算法。

A.14.1 求解线性方程组

对于变量 x_k, $k = 1, 2, \ldots, n$，我们可以写出 n 个线性方程。例如

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1\\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2\\
&\vdots\\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned} \tag{A.124}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

其中，参数 a_{jk} 和 b_j 值是已知的。该方程组可以用矩阵形式表示为

$$\mathbf{AX} = \mathbf{B} \tag{A.125}$$

其中，**A** 为 $n \times n$ 方阵，其元素是系数 a_{jk}；**X** 是 x_j 值的列矩阵；而 **B** 是 b_j 值的列矩阵。该联立线性方程组的解，可以用矩阵形式表示为

$$\mathbf{X} = \mathbf{A}^{-1}\mathbf{B} \tag{A.126}$$

该方程组有解当且仅当 **A** 是一个非奇异矩阵；即其秩非 0。否则，系数矩阵 **A** 的逆矩阵不存在。

求解方程组的一种方法是 **Cramer 定律**（Cramer' s rule）：

$$x_k = \frac{\det \mathbf{A}_k}{\det \mathbf{A}} \tag{A.127}$$

其中 $\mathbf{A}_k$ 是矩阵 **A** 中第 k 列并为 **B** 元素所取代。该方法适合于变量个数较少时。当多于 3 个或 4 个变量时，此方法极为不便，因为计算每个秩需要进行大量乘法。单个 $n \times n$ 秩的求解，需要多于 $n!$ 次乘法。

利用**高斯消去法**（Gaussian elimination）的变形来求解方程组就比较方便。高斯消去法的基本原理可以用下列两个并列方程组进行解释：

$$\begin{aligned} x_1 + 2x_2 &= -4 \\ 3x_1 + 4x_2 &= 1 \end{aligned} \tag{A.128}$$

为了求解这个方程组，我们可以使用“-3”乘第一个方程，然后让这两方程相加，以消去 x_1 项，产生方程：

$$-2x_2 = 13$$

得到解 $x_2 = -13/2$。这个解则可代入原始方程组的任意一个，从而得到 x_1 的解：$x_1 = 9$。高效的算法设计通过消元和回代步骤而获得。

LU 分解法（**LU** decomposition 或 **LU** factorization）是对高斯消去法的一种改进。此方法首先将矩阵 **A** 分解为两个矩阵：下三角矩阵 **L** 和上三角矩阵 **U**，即

$$\mathbf{A} = \mathbf{LU} \tag{A.129}$$

L 中所有对角线之上的元素均为 0，并且所有对角线元素均为 1。**U** 中所有对角线之下的元素均为 0。我们可以将方程（A.125）写为

$$\mathbf{LUX} = \mathbf{B} \tag{A.130}$$

这使求解原线性方程组的问题简化为求解两个简单得多的方程组：

$$\mathbf{LY} = \mathbf{B}, \qquad \mathbf{UX} = \mathbf{Y} \tag{A.131}$$

在方程（A.131）中，一旦得到矩阵 **Y** 的元素，就可以用其来求解矩阵 **X** 的元素。下面的方程示例了对一个 2×2 系数矩阵的分解。

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 8 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & -7 \end{bmatrix}$$

下列一组方程给出一种计算分解矩阵元素的方法，其中 u_{ij} 是上三角矩阵 **U** 的元素，而 l_{ij} 是下三角矩阵 **L** 的元素。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$
\begin{aligned}
u_{1j} &= a_{1j}, \qquad j = 1, 2, \ldots, n \\
l_{i1} &= \frac{a_{i1}}{u_{11}}, \qquad i = 2, 3, \ldots, n \\
u_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \qquad j = i, i+1, \ldots, n; \quad i \geqslant 2 \\
l_{ij} &= \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) \qquad i = j+1, j+2, \ldots, n; \quad j \geqslant 2
\end{aligned}
\tag{A.132}
$$

高斯消去法有时易受高舍入误差的影响，而且有可能得不到精确解。在这种情况下，可以使用Gauss-Seidel法求解。这种方法在我们知道解的近似值时是一种有效的方法。我们从对变量x_k“猜”一个初值开始。然后，重复计算逐次逼近，直至两个相继值之间的差足够“小”。在每次迭代中，我们对变量计算逼近值：

$$
\begin{aligned}
x_1 &= \frac{b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n}{a_{11}} \\
x_2 &= \frac{b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n}{a_{22}} \\
&\vdots
\end{aligned}
\tag{A.133}
$$

如果我们可以重排矩阵 **A**，使每个对角线元素的值大于与那行交叉的其他元素的值之和，那么Gauss-Seidel法保证收敛为有解。

A.14.2 非线性方程的求根

函数$f(x)$的根是满足方程$f(x) = 0$的x值。一般来说，$f(x)$可以是代数表达式（如多项式）或者包含超越函数。代数表达式仅包含算术运算符、指数、根和幂。超越函数表示为无穷幂级数，例如三角函数、log函数和指数函数。

非线性方程的根可以是实数、复数或者二者的组合。有时候我们可以得到所有根的精确解，这依赖于方程的复杂度。例如，我们知道求解不高于4次的多项式的精确解的方法，也知道一个简单的三角方程$\sin x = 0$的根是$x = k\pi$（k为任意整数）。但是在许多实际情况中，我们必须使用数值方法来求解非线性方程的根。

非线性方程求根的最流行的方法是**Newton-Raphson算法**（Newton-Raphson Algorithm）。这个算法是个迭代过程，在迭代的每一步，以直线逼近函数$f(x)$，如图A.27所示。我们从“猜”初值x_0为所求根的值开始，然后，将x_0处的切线与x轴的交点x_1作为下一个近似值。在x_0，曲线的斜率（一阶导数）是

$$
\frac{df}{dx} = \frac{f(x_0)}{x_0 - x_1} \tag{A.134}
$$

因此，对根的下一次逼近是

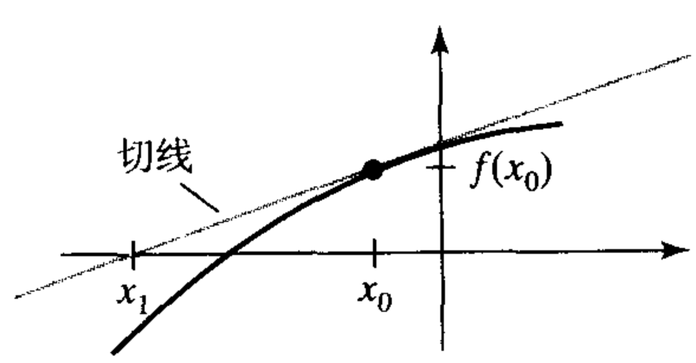


图 A.27 以直线在初值x_0处逼近曲线，该直线是曲线在该点的切线

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{A.135}$$

其中$f'(x_0)$是$f(x)$在$x = x_0$处的导数。我们在每次计算逼近时重复这个过程，直至相邻两个近似根之差“足够小”。

除了求解实变函数之外，Newton-Raphson 算法还可用于复变函数$f(z)$、多变量函数以及并列非线性的实变或复变函数组。如果 Newton-Raphson 算法收敛到一个根，那么它将比任何其他求根法收敛得快。但是，它可能并不总是收敛的。例如，若在迭代中的某些点的导数$f'(x)$是 0，该方法就失去了作用。再者，如果曲线摆动，相继的逼近可能从根的位置发散。

另一种比较慢但是保证收敛的方法是**二分法**（bisection method）。在此，我们必须首先确定包含一个根的x区间，然后在这个区间内运用二分查找法接近这个根。我们先看区间的中点，以确定此根是在该区间的下半部分还是上半部分。这个过程将对每个后继子区间重复进行，直至后继中点位置之差小于某个预置值。通过内插后继x位置而不是二等分子区间（伪定位法，false-position method）可提高速度。

A.14.3　积分估值

积分的数值解是个求和过程。对于单变量x的函数，$f(x)$的积分是该曲线“以下”的面积，如图 A.28 所示。对于简单的被积函数，我们通常可以确定积分的函数形式。但是，一般来说，积分的值需要通过数值方法来求解。

$f(x)$的积分可以按下式求和的数值逼近，即

$$\int_a^b f(x)\,dx \approx \sum_{k=1}^{n} f_k(x)\Delta x_k \tag{A.136}$$

其中，$f_k(x)$是$f(x)$在区间Δx_k上的逼近。例如，我们利用在每个子区间上的常数值逼近曲线，并把结果矩形的面积相加（参见图 A.29）。区间(a, b)的子划分越小，逼近就越好（直至一点）。实际上，如果区间划分得太小，后继矩形面积的值有可能在舍入误差中丢失。

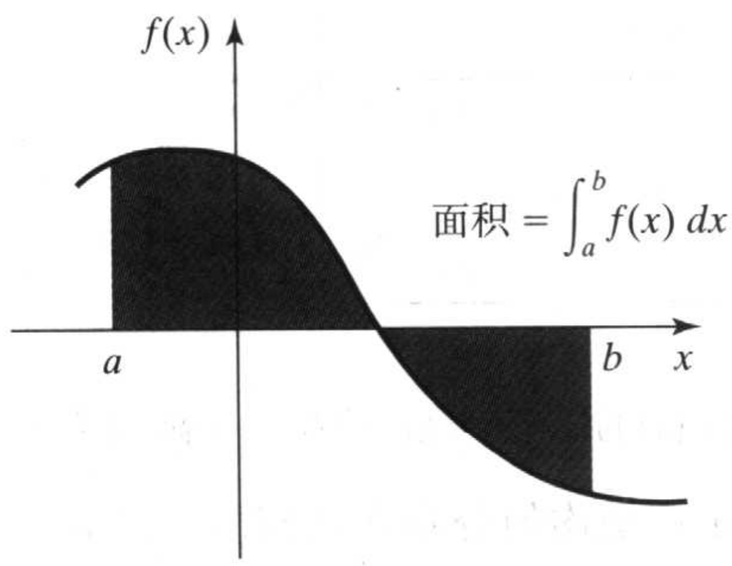


图 A.28　$f(x)$的积分等于该函数和x轴区间(a, b)之间的面积总量

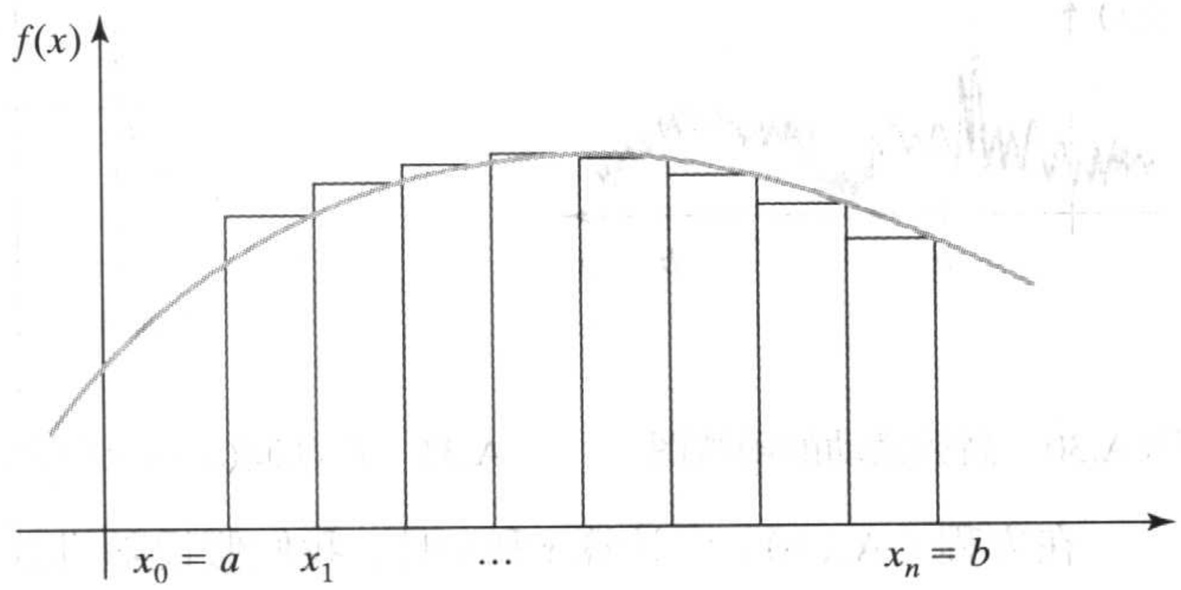


图 A.29　以小矩形面积之和来逼近积分

关于函数在每个子区间的多项式逼近，通常能给出比矩形逼近更好的结果。利用线性逼近，我们得到梯形子面积，此类逼近法则称做**梯形定律**（trapezoid rule）。如果我们在每个子区间使用二次多项式（抛物线）去逼近该函数，那么该方法称为 **Simpson 定律**（Simpson’s rule），并且该积分逼近是

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

$$\int_a^b f(x)\,dx \approx \frac{\Delta x}{3}\left[f(a)+f(b)+4\sum_{(\text{odd})k=1}^{n-1} f(x_k)+2\sum_{(\text{even})k=2}^{n-2} f(x_k)\right] \tag{A.137}$$

其中，从 a 到 b 的区间被分为几个等长区间：

$$\Delta x = \frac{b-a}{n} \tag{A.138}$$

其中，n 是 2 的倍数，而

$$x_0 = a, \qquad x_k = x_{k-1} + \Delta x, \qquad k = 1, 2, \ldots, n$$

对于高频摆动的函数（如图 A.30 所示），先前讨论的子区间逼近方法是很难给出精确结果的。再有，多变量积分（包括几个积分变量）使用 Simpson 定律或其他逼近法求解是困难的。遇上这种情况，我们可以采用 Monte Carlo 积分技术。术语“Monte Carlo”用于以随机数求解确定问题的任何方法。

Monte Carlo 法通过在区间(a, b)上包含$f(x)$的矩形区域里生成n个随机位置（参见图 A.31）来计算函数的积分。关于该积分的逼近则可计算为

$$\int_a^b f(x)\,dx \approx h(b-a)\frac{n_{\text{count}}}{n} \tag{A.139}$$

其中，参数 h 是矩形的高，参数 n_{count} 是$f(x)$和 x 轴之间随机点数的计数值。矩形区域中随机位置(x, y)的计算，通过先产生两随机数 r_1 和 r_2，然后进行如下运算：

$$h = y_{\max} - y_{\min}, \qquad x = a + r_1(b-a), \qquad y = y_{\min} + r_2 h \tag{A.140}$$

来实现。类似的方法可用于多变量积分。

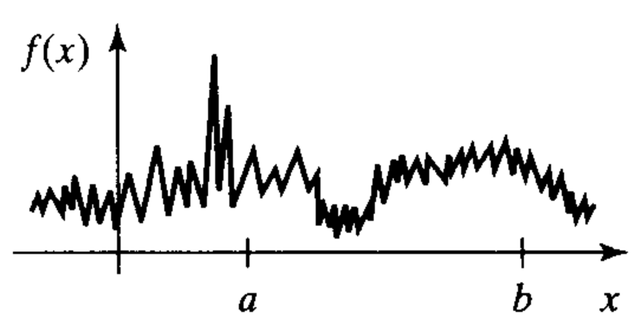
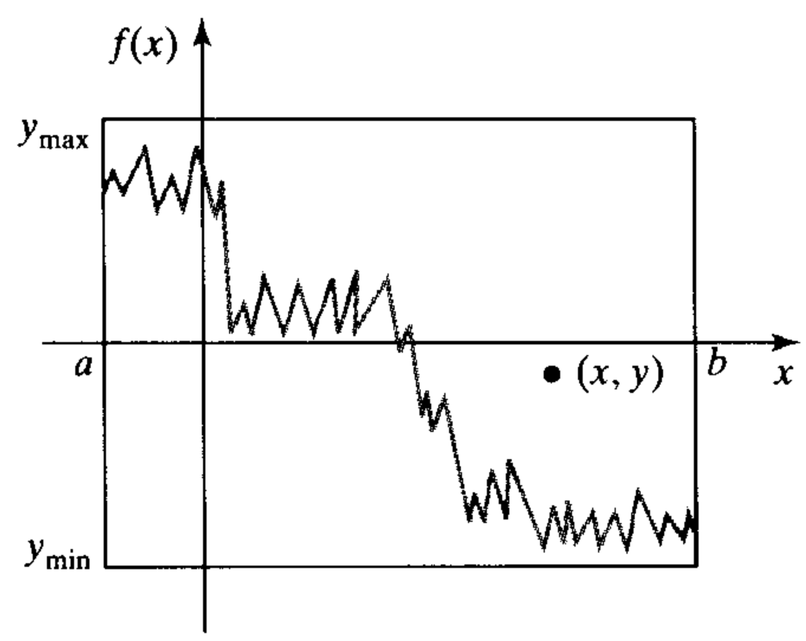


图 A.30 高频摆动的函数图



A.31 在区间(a, b)上包围函数$f(x)$的矩形区域内产生一个随机位置(x, y)

在方程（A.140）中计算 x 和 y 时，我们假设随机数 r_1 和 r_2 是均匀分布在区间(0, 1)上的。我们可以使用数学或统计库中的随机函数得到随机数，或者利用如下算法（称为线性同余发生器，linear congruential generator）：

$$\begin{aligned} i_k &= a\, i_{k-1} + c \pmod m, \qquad k = 1, 2, 3, \ldots \\ r_k &= \frac{i_k}{m} \end{aligned} \tag{A.141}$$

其中，参数 a、c、m 和 i_0 是整数，而 i_0 是起始值，称为种子（seed）。参数 m 对于特定机器选得尽可能大，而 a 和 c 的值则使随机数串在某一值被重复前尽可能长。例如，在以 32 位整数表示的机器上，我们可以设 $m = 2^{32}-1$, $a = 1\,664\,525$ 和 $c = 1\,013\,904\,223$。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

A.14.4 求解常微分方程

任何包含微分变化率算子的方程均可称为微分方程。数量的值可以从一个坐标位置到另一个位置以连续的方式改变，也可以在一个固定位置上随时间而变化，还可以关于许多其他参数变化，例如温度或旋转加速度。仅涉及单变量函数的导数的方程称为常微分方程（ordinary differential equation）。求解微分方程有两种方法：一是确定满足该方程的函数形式，二是用数值逼进方法来决定数量在选定区间上的值。

解微分方程也需要知道一个或多个起始值。仅包含一阶导数的方程，称为一阶微分方程（first-order differential equation），需要一个起始值。包含一阶和二阶导数的方程（称为二阶微分方程，second-order differential equation）需要两个起始值。对于包含更高阶导数的方程，依次类推。指定起始值有两类基本方法。**初值问题**（initial-value problem）将已知条件用于指定自变量的一个值。**边界值问题**（boundary-value problem）将已知条件用于指定因变量的边界。

一阶微分方程是初值问题的简单例子：

$$\frac{dx}{dt} = f(x, t), \qquad x(t_0) = x_0 \tag{A.142}$$

其中 x 表示随时间 t（自变量）变化的因变量，$f(x, t)$是已知的关于 x 的一阶导数的时变函数，x_0 是 x 在初始时间 t_0 的给定值。我们也可以将方程（A.142）写为

$$dx = f(x, t)\,dt$$

并且，根据导数的定义，我们可以用有限区间来逼近微分。如下所示：

$$\Delta x_k \approx f(x_k, t_k)\Delta t_k, \qquad k = 0, 1, \ldots, n \tag{A.143}$$

其中 $\Delta x_k = x_{k+1} - x_k$ 并且 $\Delta t_k = t_{k+1} - t_k$，在 n 个时间段上。给定 t_0 时刻的值 x_0，我们通常采用相等的时间间隔，并使用以下递增计算来确定在每个时间段上的 x 值。

$$x_{k+1} = x_k + f(x_k, t_k)\Delta t \tag{A.144}$$

这个数值计算过程称为**欧拉方法**，它在每个时间间隔 Δt 上用直线段来逼进 x。

虽然欧拉方法实现简便，但一般来说不够精确。对它的改进方法使用泰勒展式将高阶项合并到对微分方程的逼进中。

$$x(t + \Delta t) = x(t) + x'(t)\Delta t + \frac{1}{2}x''(t)\Delta t^2 + \cdots \tag{A.145}$$

因为 $x'(t) = f(x, t)$，我们有 $x''(t) = f'(x, t)$，对更高阶导数依次类推。

一种更精确、更广泛使用的计算一阶微分方程的方法是**Runge-Kutta算法**，也称为**四阶Runge-Kutta算法**（fourth-order Runge-Kutta algorithm）。这个方法基于四阶泰勒展式，具体算法如下：

$$\begin{aligned}
a &= f(x_k, t_k)\Delta t \\
b &= f(x_k + a/2,\ t_k + \Delta t/2)\Delta t \\
c &= f(x_k + b/2,\ t_k + \Delta t/2)\Delta t \\
d &= f(x_k + c,\ t_k + \Delta t)\Delta t \\
t_{k+1} &= t_k + \Delta t \\
x_{k+1} &= x_k + (a + 2b + 2c + d)/6, \qquad k = 0, 1, \ldots n-1
\end{aligned} \tag{A.146}$$

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

我们可以使用类似方法求解更高阶的常微分方程。通用的方法是用x的泰勒展式来包含x'',x''',以及更高阶，最高阶数取决于微分方程的阶数。例如，我们可以从泰勒级数得到二阶导数的逼近：

$$x''(t_k) \approx \frac{x_{k+1} - 2x_k + x_{k-1}}{\Delta t^2} \tag{A.147}$$

A.14.5 求解偏微分方程

正如我们预计的，偏微分方程比常微分方程更难求解。但我们可以采用类似的方法并且用有限差来替换偏导数。

首先考虑函数$f(x, t)$，它依赖于坐标x和时间t。通过用有限差替换偏导数可以将包含$\partial f/\partial x$和$\partial f/\partial t$的偏微分方程简化为常微分方程。这将具有两个变量的函数转换为具有一个变量（时间）的单下标函数：

$$f(x, t) \rightarrow f(x_k, t) \rightarrow f_k(t) \tag{A.148}$$

则偏导数可被替换为下列表达式：

$$\begin{aligned} \frac{\partial f}{\partial x} &= \frac{f_{k+1} - f_k}{\Delta x} \\ \frac{\partial f}{\partial t} &= \frac{df_k}{dt} \end{aligned} \tag{A.149}$$

利用给定的初始条件或边界条件，我们可以在有限个x位置求解方程。

对于更高阶导数，我们可以用泰勒展式获得有限差逼近。例如，f关于x的二阶偏导数可以由下式逼近：

$$\frac{\partial^2 f(x, t)}{\partial x^2} \rightarrow \frac{f_{k+1}(t) - 2f_k(t) + f_{k-1}(t)}{\Delta x^2} \tag{A.150}$$

当函数在空间内的面或者体上定义时，我们可以将该空间划分为规则的网格并对每个空间坐标使用有限差。

有限元方法（finite-element method）是另一种求解偏微分方程的方法。首先在目标区域（空间中的面或体）中设置坐标位置的网格，然后运用变分技术在节点位置上解联立方程组。这种方法使用逼近函数解，而不是有限差方程。针对不同的问题，对不同的量（如势能或剩余误差）进行积分。然后用最小二乘分析等方法最小化势能或剩余误差。最小化处理给出逼近函数解中的位置参数的值。

A.14.6 数据集的最小二乘曲线拟合方法

我们经常需要确定函数形式，用来描述计算机仿真和科学可视化研究产生的数据集。为数据点集拟合一函数（线性或非线性）的标准方法是**最小二乘法**（least-square algorithm）。我们首先选择函数形式，这可能是直线函数、一个多项式函数或某些其他曲线形式。然后，确定选定函数形式中参数的值。例如，一个二维直线方程用两个参数来描述：斜率和y轴截距。最后，通过最小化理论函数值与实际数据值之间的差的平方和来确定函数的参数。

对于二维数据点集(x_k, y_k)，$k = 1, 2, \ldots, n$，在选定用于描述数据分布的函数形式$f(x)$后，我们确定误差函数的表达式E，即在所有x_k上$f(x_k)$和y_k值之间的差的平方和：

$$E = \sum_{k=1}^{n} [y_k - f(x_k)]^2 \tag{A.151}$$

函数$f(x)$的参数通过最小化关于E的表达式来确定。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

例如，对于用线性函数 $f(x)$ 描述的数据集：

$$f(x) = a_0 + a_1 x$$

则

$$E = \sum_{k=1}^{n} \left[y_k^2 - 2y_k(a_0 + a_1 x_k) + a_0^2 + 2a_0 a_1 x_k + a_1^2 x_k^2 \right] \tag{A.152}$$

因为误差 E 是两个变量（a_0 和 a_1）的函数，我们通过下列两个联立方程最小化 E：

$$\begin{aligned} \frac{\partial E}{\partial a_0} &= \sum_{k=1}^{n} [-2y_k + 2a_0 + 2a_1 x_k] = 0 \\ \frac{\partial E}{\partial a_1} &= \sum_{k=1}^{n} \left[-2y_k x_k + 2a_0 x_k + 2a_1 x_k^2 \right] = 0 \end{aligned} \tag{A.153}$$

使用 Cramer 定律求解联立线性方程得到

$$\begin{aligned} a_0 &= \frac{(\sum_k x_k^2)(\sum_k y_k) - (\sum_k x_k)(\sum_k x_k y_k)}{D} \\ a_1 &= \frac{n \sum_k x_k y_k - (\sum_k x_k)(\sum_k y_k)}{D} \end{aligned} \tag{A.154}$$

其中这两个表达式的行列式为

$$\begin{aligned} D &= \begin{vmatrix} n & \sum_k x_k \\ \sum_k x_k & \sum_k x_k^2 \end{vmatrix} \\ &= \sum_{k=1}^{n} x_k^2 - \left(\sum_{k=1}^{n} x_k \right)^2 \end{aligned} \tag{A.155}$$

则 E 是最小的。

为其他函数可进行类似的计算，对于多项式：

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

我们需要解 n 个线性方程组，以确定参数 a_k 的值。而且也可运用最小二乘法拟合几个变量的函数 $f(x_1, x_2, \ldots, x_m)$，对于每一个变量，可以是线性的或非线性的。

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

参考文献

AKELEY, K. and T. JERMOLUK (1988). "High-Performance Polygon Rendering", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 239–246.

AKELEY, K. (1993). "RealityEngine Graphics", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 109–116.

AKENINE-MÖLLER, T. and E. HAINES (2002). *Real-Time Rendering, Second Edition*, A. K. Peters, Natick, MA.

AMANATIDES, J. (1984). "Ray Tracing with Cones", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 129–135.

ANJYO, K., Y. USAMI, and T. KURIHARA (1992). "A Simple Method for Extracting the Natural Beauty of Hair", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 111–120.

APPLE COMPUTER, INC. (1987). *Human Interface Guidelines: The Apple Desktop Interface*, Addison-Wesley, Reading, MA.

ARVO, J. and D. KIRK (1987). "Fast Ray Tracing by Ray Classification", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 55–64.

ARVO, J., ed. (1991). *Graphics Gems II*, Academic Press, San Diego, CA.

ATHERTON, P. R. (1983). "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry", in proceedings of SIGGRAPH '83, *Computer Graphics*, 17(3), pp. 73–82.

BARAFF, D. (1989). "Analytical Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 223–232.

BARAFF, D. and A. WITKIN (1992). "Dynamic Simulation of Non-Penetrating Flexible Bodies", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 303–308.

BARNSLEY, M. F., A. JACQUIN, F. MALASSENET, L. REUTER, and D. SLOAN (1988). "Harnessing Chaos for Image Synthesis", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 131–140.

BARNSLEY, M. F. (1993). *Fractals Everywhere, Second Edition*, Academic Press, San Diego, CA.

BARNSLEY, M. F. and L. P. HURD, (1993). *Fractal Image Compression*, AK Peters, Wellesly, MA.

BARR, A. H. (1981). "Superquadrics and Angle-Preserving Transformations", *IEEE Computer Graphics and Applications*, 1(1), pp. 11–23.

BARSKY, B. A. and J. C. BEATTY (1983). "Local Control of Bias and Tension in Beta-Splines", *ACM Transactions on Graphics*, 2(2), pp. 109–134.

BARSKY, B. A. (1984). "A Description and Evaluation of Various 3-D Models", *IEEE Computer Graphics and Applications*, 4(1), pp. 38–52.

BARZEL, R. and A. H. BARR (1988). "A Modeling System Based on Dynamic Constraints", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 179–188.

BARZEL, R. (1992). *Physically-Based Modeling for Computer Graphics*, Academic Press, San Diego, CA.

BAUM, D. R., S. MANN, K. P. SMITH, and J. M. WINGET (1991). "Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 51–61.

BERGMAN, L. D., J. S. RICHARDSON, D. C. RICHARDSON, and F. P. BROOKS, JR. (1993). "VIEW—an Exploratory Molecular Visualization System with User-Definable Interaction Sequences", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 117–126.

BÉZIER, P. (1972). *Numerical Control: Mathematics and Applications*, translated by A. R. Forrest and A. F. Pankhurst, John Wiley & Sons, London.

BIRN, J. (2000). *[digital] Lighting & Rendering*, New Riders Publishing, Indianapolis, IN.

BISHOP, G. and D. M. WIEMER (1986). "Fast Phong Shading", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 103–106.

BLAKE, J. W. (1993). *PHIGS and PHIGS Plus*, Academic Press, London.

BLESER, T. (1988). "TAE Plus Styleguide User Interface Description", NASA Goddard Space Flight Center, Greenbelt, MD.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

BLINN, J. F. and M. E. NEWELL (1976). "Texture and Reflection in Computer-Generated Images", *Communications of the ACM*, 19(10), pp. 542–547.

BLINN, J. F. (1977). "Models of Light Reflection for Computer-Synthesized Pictures", in proceedings of SIGGRAPH '77, *Computer Graphics*, 11(2), pp. 192–198.

BLINN, J. F. and M. E. NEWELL (1978). "Clipping Using Homogeneous Coordinates", in proceedings of SIGGRAPH '78, *Computer Graphics*, 12(3), pp. 245–251.

BLINN, J. F. (1978). "Simulation of Wrinkled Surfaces", in proceedings of SIGGRAPH '78, *Computer Graphics*, 12(3), pp. 286–292.

BLINN, J. F. (1982). "A Generalization of Algebraic Surface Drawing", *ACM Transactions on Graphics*, 1(3), pp. 235–256.

BLINN, J. F. (1982). "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces", in proceedings of SIGGRAPH '82, *Computer Graphics*, 16(3), pp. 21–29.

BLINN, J. F. (1993). "A Trip Down the Graphics Pipeline: The Homogeneous Perspective Transform", *IEEE Computer Graphics and Applications*, 13(3), pp. 75–80.

BLINN, J. (1996). *Jim Blinn's Corner: A Trip Down the Graphics Pipeline*, Morgan Kauffman, San Francisco, CA.

BLINN, J. (1998). *Jim Blinn's Corner: Dirty Pixels*, Morgan Kauffman, San Francisco, CA.

BLOOMENTHAL, J. (1985). "Modeling the Mighty Maple", in proceedings of SIGGRAPH '85, *Computer Graphics*, 19(3), pp. 305–312.

BONO, P. R., J. L. ENCARNACAO, F. R. A. HOPGOOD, et al. (1982). "GKS: The First Graphics Standard", *IEEE Computer Graphics and Applications*, 2(5), pp. 9–23.

BOUQUET, D. L. (1978). "An Interactive Graphics Application to Advanced Aircraft Design", in proceedings of SIGGRAPH '78, *Computer Graphics*, 12(3), pp. 330–335.

BOURG, D. M. (2002). *Physics for Game Developers*, O'Reilly & Associates, Sebastopol, CA.

BRESENHAM, J. E. (1965). "Algorithm for Computer Control of a Digital Plotter", *IBM Systems Journal*, 4(1), pp. 25–30.

BRESENHAM, J. E. (1977). "A Linear Algorithm for Incremental Digital Display of Circular Arcs", *Communications of the ACM*, 20(2), pp. 100–106.

BROOKS JR., F. P. (1986). "Walkthrough: A Dynamic Graphics System for Simulating Virtual Buildings", *Interactive 3D*.

BROOKS JR., F. P. (1988). "Grasping Reality Through Illusion: Interactive Graphics Serving Science", *CHI '88*, pp. 1–11.

BROOKS JR., F. P., M. OUH-YOUNG, J. J. BATTER, and P. J. KILPATRICK (1990). "Project GROPE—Haptic Display for Scientific Visualization", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 177–185.

BROWN, J. R. and S. CUNNINGHAM (1989). *Programming the User Interface*, John Wiley & Sons, New York.

BROWN, C. W. and B. J. SHEPHERD (1995). *Graphics File Formats*, Manning Publications, Greenwich, CT.

BRUDERLIN, A. and T. W. CALVERT (1989). "Goal-Directed, Dynamic Animation of Human Walking", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 233–242.

BRUNET, P. and I. NAVAZO (1990). "Solid Representation and Operation Using Extended Octrees", *ACM Transactions on Graphics*, 9(2), pp. 170–197.

BRYSON, S. and C. LEVIT (1992). "The Virtual Wind Tunnel", *IEEE Computer Graphics and Applications*, 12(4), pp. 25–34.

CALVERT, T., A. BRUDERLIN, J. DILL, T. SCHIPHORST, and C. WEILMAN (1993). "Desktop Animation of Multiple Human Figures", *IEEE Computer Graphics and Applications*, 13(3), pp. 18–26.

CAMBELL, G., T. A. DEFANTI, J. FREDERIKSEN, S. A. JOYCE, and L. A. LESKE (1986). "Two Bit/Pixel Full-Color Encoding", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 215–224.

CARPENTER, L. (1984). "The A-Buffer: An Antialiased Hidden-Surface Method", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 103–108.

CHEN, S. E., H. E. RUSHMEIER, G. MILLER, and D. TURNER (1991). "A Progressive Multi-Pass Method for Global Illumination", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 165–174.

CHIN, N. and S. FEINER (1989). "Near Real-Time Shadow Generation Using BSP Trees", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 99–106.

CHUNG, J. C., et al. (1989). "Exploring Virtual Worlds with Head-Mounted Visual Displays", *Proceedings of the SPIE (Society of Photo-Optical Instrumentation Engineers, now called the International Society for Optical Engineering) Conference on Three-Dimensional Visualization and Display Technologies*, 1083, January 1989, pp. 15–20.

COHEN, M. F. and D. P. GREENBERG (1985). "The Hemi-Cube: A Radiosity Solution for Complex Environments", in proceedings of SIGGRAPH '85, *Computer Graphics*, 19(3), pp. 31–40.

COHEN, M. F., S. E. CHEN, J. R. WALLACE, and D. P. GREENBERG (1988). "A Progressive Refinement Approach to Fast Radiosity Image Generation", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 75–84.

COHEN, M. F. and J. R. WALLACE (1993). *Radiosity and Realistic Image Synthesis*, Academic Press, Boston, MA.

COOK, R. L. and K. E. TORRANCE (1982). "A Reflectance Model for Computer Graphics", *ACM Transactions on Graphics*, 1(1), pp. 7–24.

COOK, R. L., T. PORTER, and L. CARPENTER (1984). "Distributed Ray Tracing", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 137–145.

COOK, R. L. (1984). "Shade Trees", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 223–231.

COOK, R. L. (1986). "Stochastic Sampling in Computer Graphics", *ACM Transactions on Graphics*, 6(1), pp. 51–72.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

COOK, R. L., L. CARPENTER, and E. CATMULL (1987). "The Reyes Image Rendering Architecture", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 95–102.

COQUILLART, S. and P. JANCENE (1991). "Animated Free-Form Deformation: An Interactive Animation Technique", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 23–26.

CROW, F. C. (1977a). "The Aliasing Problem in Computer-Synthesized Shaded Images", *Communications of the ACM*, 20(11), pp. 799–805.

CROW, F. C. (1977b). "Shadow Algorithms for Computer Graphics", in proceedings of SIGGRAPH '77, *Computer Graphics*, 11(2), pp. 242–248.

CROW, F. C. (1978). "The Use of Grayscale for Improved Raster Display of Vectors and Characters", in proceedings of SIGGRAPH '78, *Computer Graphics*, 12(3), pp. 1–5.

CROW, F. C. (1981). "A Comparison of Antialiasing Techniques", *IEEE Computer Graphics and Applications*, 1(1), pp. 40–49.

CROW, F. C. (1982). "A More Flexible Image Generation Environment", in proceedings of SIGGRAPH '82, *Computer Graphics*, 16(3), pp. 9–18.

CRUZ-NEIRA, C., D. J. SANDIN, and T. A. DEFANTI (1993). "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 135–142.

CUNNINGHAM, S., N. K. CRAIGHILL, M. W. FONG, J. BROWN, and J. R. BROWN, eds. (1992). *Computer Graphics Using Object-Oriented Programming*, John Wiley & Sons, New York.

CUTLER, E., D. GILLY, and T. O'REILLY, eds. (1992). *The X Window System in a Nutshell*, Second Edition, O'Reilly & Assoc., Inc., Sebastopol, CA.

CYRUS, M. and J. BECK (1978). "Generalized Two- and Three-Dimensional Clipping", *Computers and Graphics*, 3(1), pp. 23–28.

DAY, A. M. (1990). "The Implementation of an Algorithm to Find the Convex Hull of a Set of Three-Dimensional Points", *ACM Transactions on Graphics*, 9(1), pp. 105–132.

DEERING, M. (1992). "High Resolution Virtual Reality", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 195–202.

DEERING, M. F. and S. R. NELSON (1993). "Leo: A System for Cost-Effective 3D Shaded Graphics", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 101–108.

DEMERS, O. (2002). *[digital] Texturing & Painting*, New Riders Publishing, Indianapolis, IN.

DEPP, S. W. and W. E. HOWARD (1993). "Flat-Panel Displays", *Scientific American*, 266(3), pp. 90–97.

DE REFFYE, P., C. EDELIN, J. FRANÇON, M. JAEGER, and C. PUECH (1988). "Plant Models Faithful to Botanical Structure and Development", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 151–158.

DEROSE, T. D. (1988). "Geometric Continuity, Shape Parameters, and Geometric Constructions for Catmull-Rom Splines", *ACM Transactions on Graphics*, 7(1), pp. 1–41.

DIGITAL EQUIPMENT CORP. (1989). "Digital Equipment Corporation XUI Style Guide", Maynard, MA.

DOCTOR, L. J. and J. G. TORBERG (1981). "Display Techniques for Octree-Encoded Objects", *IEEE Computer Graphics and Applications*, 1(3), pp. 29–38.

DORSEY, J. O., F. X. SILLION, and D. P. GREENBERG (1991). "Design and Simulation of Opera Lighting and Projection Effects", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 41–50.

DREBIN, R. A., L. CARPENTER, and P. HANRAHAN (1988). "Volume Rendering", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 65–74.

DURRETT, H. J., ed. (1987). *Color and the Computer*, Academic Press, Boston.

DUVANENKO, V. (1990). "Improved Line-Segment Clipping", *Dr. Dobb's Journal*, July 1990.

DYER, S. (1990). "A Dataflow Toolkit for Visualization", *IEEE Computer Graphics and Applications*, 10(4), pp. 60–69.

EARNSHAW, R. A., ed. (1985). *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, Berlin.

EDELSBRUNNER, H. (1987). *Algorithms in Computational Geometry*, Springer-Verlag, Berlin.

EDELSBRUNNER, H. and E. P. MUCKE (1990). "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms", *ACM Transactions on Graphics*, 9(1), pp. 66–104.

ELBER, G. and E. COHEN (1990). "Hidden-Curve Removal for Free-Form Surfaces", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 95–104.

ENDERLE, G., K. KANSY, and G. PFAFF (1984). *Computer Graphics Programming: GKS—The Graphics Standard*, Springer-Verlag, Berlin.

FARIN, G. (1988). *Curves and Surfaces for Computer-Aided Geometric Design*, Academic Press, Boston, MA.

FARIN, G. and D. HANSFORD (1998). *The Geometry Toolbox for Graphics and Modeling*, A. K. Peters, Natick, MA.

FEDER, J. (1988). *Fractals*, Plenum Press, New York.

FEYNMAN, R. P., R. B. LEIGHTON, and M. L. SANDS (1989). *The Feynman Lectures on Physics*, Addison-Wesley, Reading, MA.

FISHKIN, K. P. and B. A. BARSKY (1984). "A Family of New Algorithms for Soft Filling", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 235–244.

FIUME, E. L. (1989). *The Mathematical Structure of Raster Graphics*, Academic Press, Boston.

FOLEY, J. D., A. VAN DAM, S. K. FEINER, and J. F. HUGHES (1990). *Computer Graphics: Principles and Practice, Second Edition*, Addison-Wesley, Reading, MA.

FOURNIER, A., D. FUSSEL, and L. CARPENTER (1982). "Computer Rendering of Stochastic Models", *Communications of the ACM*, 25(6), pp. 371–384.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

FOURNIER, A. and W. T. REEVES (1986). "A Simple Model of Ocean Waves", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 75–84.

FOWLER, D. R., H. MEINHARDT, and P. PRUSINKIEWICZ (1992). "Modeling Seashells", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 379–387.

FRANKLIN, W. R. and M. S. KANKANHALLI (1990). "Parallel Object-Space Hidden Surface Removal", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 87–94.

FREEMAN, H. ed. (1980). *Tutorial and Selected Readings in Interactive Computer Graphics*, IEEE Computer Society Press, Silver Springs, MD.

FRENKEL, K. A. (1989). "Volume Rendering", *Communications of the ACM*, 32(4), pp. 426–435.

FRIEDER, G., D. GORDON, and R. A. REYNOLD (1985). "Back-to-Front Display of Voxel-Based Objects", *IEEE Computer Graphics and Applications*, 5(1), pp. 52–60.

FRIEDHOFF, R. M. and W. BENZON (1989). *The Second Computer Revolution: Visualization*, Harry N. Abrams, New York.

FU, K. S., and A. ROSENFELD (1984). "Pattern Recognition and Computer Vision", *Computer*, 17(10), pp. 274–282.

FUJIMOTO, A., and K. IWATA (1983). "Jag-Free Images on Raster Displays", *IEEE Computer Graphics and Applications*, 3(9), pp. 26–34.

FUNKHOUSER, T. A. and C. H. SEQUIN (1993). "Adaptive Display Algorithms for Interactive Frame Rates during Visualization of Complex Virtual Environments", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 247–254.

GARDNER, T. N., and H. R. NELSON (1983). "Interactive Graphics Developments in Energy Exploration", *IEEE Computer Graphics and Applications*, 3(2), pp. 33–34.

GARDNER, G. Y. (1985). "Visual Simulation of Clouds", in proceedings of SIGGRAPH '85, *Computer Graphics*, 19(3), pp. 297–304.

GASCUEL, M.-P. (1993). "An Implicit Formulation for Precise Contact Modeling between Flexible Solids", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 313–320.

GASKINS, T. (1992). *PHIGS Programming Manual*, O'Reilly & Associates, Sebastopol, CA.

GHARACHORLOO, N., S. GUPTA, R. F. SPROULL, and I. E. SUTHERLAND (1989). "A Characterization of Ten Rasterization Techniques", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 355–368.

GIRARD, M. (1987). "Interactive Design of 3D Computer-Animated Legged Animal Motion", *IEEE Computer Graphics and Applications*, 7(6), pp. 39–51.

GLASSNER, A. S. (1984). "Space Subdivision for Fast Ray Tracing", *IEEE Computer Graphics and Applications*, 4(10), pp. 15–22.

GLASSNER, A. S. (1986). "Adaptive Precision in Texture Mapping", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 297–306.

GLASSNER, A. S. (1988). "Spacetime Ray Tracing for Animation", *IEEE Computer Graphics and Applications*, 8(2), pp. 60–70.

GLASSNER, A. S., ed. (1989a). *An Introduction to Ray Tracing*, Academic Press, San Diego, CA.

GLASSNER, A. S. (1989b). *3D Computer Graphics: A User's Guide for Artists and Designers, Second Edition*, Design Books, Lyons & Bufford Publishers, New York.

GLASSNER, A. S., ed. (1990). *Graphics Gems*, Academic Press, San Diego, CA.

GLASSNER, A. S. (1992). "Geometric Substitution: A Tutorial", *IEEE Computer Graphics and Applications*, 12(1), pp. 22–36.

GLASSNER, A. S. (1995). *Principles of Digital Image Synthesis, Vols. 1–2*, Morgan Kaufmann, San Francisco, CA.

GLASSNER, A. S. (1999). *Andrew Glassner's Notebook: Recreational Computer Graphics*, Morgan Kaufmann, San Francisco, CA.

GLASSNER, A. S. (2002). *Andrew Glassner's Other Notebook: Further Recreations in Computer Graphics*, A. K. Peters, Natick, MA.

GLEICHER, M. and A. WITKIN (1992). "Through-the-Lens Camera Control", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 331–340.

GOLDSMITH, J. and J. SALMON (1987). "Automatic Creation of Object Hierarchies for Ray Tracing", *IEEE Computer Graphics and Applications*, 7(5), pp. 14–20.

GOMES, J., L. DARSA, B. COSTA, and L. VELHO (1999). *Warping and Morphing of Graphical Objects*, Morgan Kaufmann, San Francisco, CA.

GONZALEZ, R. C. and P. WINTZ (1987). *Digital Image Processing*, Addison-Wesley, Reading, MA.

GOOCH, B. and A. GOOCH (2001). *Non-Photorealistic Rendering*, A. K. Peters, Natick, MA.

GORAL, C. M., K. E. TORRANCE, D. P. GREENBERG, and B. BATTAILE (1984). "Modeling the Interaction of Light Between Diffuse Surfaces", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 213–222.

GORDON, D. and S. CHEN (1991). "Front-to-Back Display of BSP Trees", *IEEE Computer Graphics and Applications*, 11(5), pp. 79–85.

GORTLER, S. J., P. SCHRÖDER, M. F. COHEN, and P. HANRAHAN (1993). "Wavelet Radiosity", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 221–230.

GOURAUD, H. (1971). "Continuous Shading of Curved Surfaces", *IEEE Transactions on Computers*, C-20(6), pp. 623–628.

GREENE, N., M. KASS, and G. MILLER (1993). "Hierarchical Z-Buffer Visibility", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 231–238.

GROTCH, S. L. (1983). "Three-Dimensional and Stereoscopic Graphics for Scientific Data Display and

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Analysis", *IEEE Computer Graphics and Applications*, 3(8), pp. 31–43.

HAEBERLI, P. and K. AKELEY (1990). "The Accumulation Buffer: Hardware Support for High-Quality Rendering", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 309–318.

HALL, R. A. and D. P. GREENBERG (1983). "A Testbed for Realistic Image Synthesis", *IEEE Computer Graphics and Applications*, 3(8), pp. 10–20.

HALL, R. (1989). *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, New York.

HALLIDAY, D., R. RESNICK, and J. WALKER (2000). *Fundamentals of Physics, Sixth Edition*, John Wiley & Sons, New York.

HANRAHAN, P. and J. LAWSON (1990). "A Language for Shading and Lighting Calculations", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 289–298.

HARDY, V. J. (2000). *Java 2D API Graphics*, Sun Microsystems Press, Palo Alto, CA.

HART, J. C., D. J. SANDIN, and L. H. KAUFFMAN (1989). "Ray Tracing Deterministic 3D Fractals", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 289–296.

HART, J. C. and T. A. DEFANTI (1991). "Efficient Antialiased Rendering of 3-D Linear Fractals", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 91–100.

HAWRYLYSHYN, P. A., R. R. TASKER, and L. W. ORGAN (1977). "CASS: Computer-Assisted Stereotaxic Surgery", in proceedings of SIGGRAPH '77, *Computer Graphics*, 11(2), pp. 13–17.

HE, X. D., P. O. HEYNEN, R. L. PHILLIPS, K. E. TORRANCE, D. H. SALESIN, and D. P. GREENBERG (1992). "A Fast and Accurate Light Reflection Model", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 253–254.

HEARN, D. and P. BAKER (1991). "Scientific Visualization: An Introduction", *Eurographics '91 Technical Report Series*, Tutorial Lecture 6, Vienna, Austria.

HECKBERT, P. S. (1982). "Color Image Quantization for Frame Buffer Display", in proceedings of SIGGRAPH '82, *Computer Graphics*, 16(3), pp. 297–307.

HECKBERT, P. S. and P. HANRAHAN (1984). "Beam Tracing Polygonal Objects", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 119–127.

HECKBERT, P. S., ed. (1994). *Graphics Gems IV*, Academic Press Professional, Cambridge, MA.

HENDERSON, L. R. and A. M. MUMFORD (1993). *The CGM Handbook*, Academic Press, San Diego, CA.

HOPGOOD, F. R. A., D. A. DUCE, J. R. GALLOP, and D. C. SUTCLIFFE (1983). *Introduction to the Graphical Kernel System (GKS)*, Academic Press, London.

HOPGOOD, F. R. A. and D. A. DUCE (1991). *A Primer for PHIGS*, John Wiley & Sons, Chichester, England.

HORSTMANN, C. S. and G. CORNELL (2001). *Core Java 2, Vols. I–II*, Sun Microsystems Press, Palo Alto, CA.

HOWARD, T. L. J., W. T. HEWITT, R. J. HUBBOLD, and K. M. WYRWAS (1991). *A Practical Introduction to PHIGS and PHIGS Plus*, Addison-Wesley, Wokingham, England.

HUFFMAN, D. A. (1952). "A Method for the Construction of Minimum-Redundancy Codes", *Communications of the ACM*, 40(9), pp. 1098–1101.

HUGHES, J. F. (1992). "Scheduled Fourier Volume Morphing", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 43–46.

HUITRIC, H. and M. NAHAS (1985). "B-Spline Surfaces: A Tool for Computer Painting", *IEEE Computer Graphics and Applications*, 5(3), pp. 39–47.

IMMEL, D. S., M. F. COHEN, and D. P. GREENBERG (1986). "A Radiosity Method for Non-Diffuse Environments", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 133–142.

ISAACS, P. M. and M. F. COHEN (1987). "Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions, and Inverse Dynamics", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 215–224.

JARVIS, J. F., C. N. JUDICE, and W. H. NINKE (1976). "A Survey of Techniques for the Image Display of Continuous Tone Pictures on Bilevel Displays", *Computer Graphics and Image Processing*, 5(1), pp. 13–40.

JENSEN, H. W. (2001). *Realistic Image Synthesis Using Photon Mapping*, A. K. Peters, Natick, MA.

JOHNSON, S. A. (1982). "Clinical Varifocal Mirror Display System at the University of Utah", *Proceedings of SPIE*, 367, August 1982, pp. 145–148.

KAJIYA, J. T. (1983). "New Techniques for Ray Tracing Procedurally Defined Objects", *ACM Transactions on Graphics*, 2(3), pp. 161–181.

KAJIYA, J. T. (1986). "The Rendering Equation", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 143–150.

KAJIYA, J. T. and T. L. KAY (1989). "Rendering Fur with Three-Dimensional Textures", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 271–280.

KAPPEL, M. R. (1985). "An Ellipse-Drawing Algorithm for Faster Displays", in *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, Berlin, pp. 257–280.

KAY, T. L. and J. T. KAJIYA (1986). "Ray Tracing Complex Scenes", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 269–278.

KAY, D. C. and J. R. LEVINE (1992). *Graphics File Formats*, Windcrest/McGraw-Hill, New York.

KELLEY, A. D., M. C. MALIN, and G. M. NIELSON (1988). "Terrain Simulation Using a Model of Stream Erosion", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 263–268.

KENT, J. R., W. E. CARLSON, and R. E. PARENT (1992). "Shape Transformation for Polyhedral Objects", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 47–54.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

KERLOW, I. V. (2000). *The Art of 3-D Computer Animation and Imaging*, John Wiley & Sons, New York.

KILGARD, M. J. (1996). *OpenGL Programming for the X Window System*, Addison-Wesley, Reading, MA.

KIRK, D. and J. ARVO (1991). "Unbiased Sampling Techniques for Image Synthesis", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 153–156.

KIRK, D., ed. (1992). *Graphics Gems III*, Academic Press, San Diego, CA.

KNUDSEN, J. (1999). *Java 2D Programming*, O'Reilly & Associates, Sebastopol, CA.

KNUTH, D. E. (1987). "Digital Halftones by Dot Diffusion", *ACM Transactions on Graphics*, 6(4), pp. 245–273.

KOCHANEK, D. H. U. and R. H. BARTELS (1984). "Interpolating Splines with Local Tension, Continuity, and Bias Control", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 33–41.

KOH, E.-K. and D. HEARN (1992). "Fast Generation and Surface Structuring Methods for Terrain and Other Natural Phenomena", in proceedings of Eurographics '92, *Computer Graphics Forum*, 11(3), pp. C169–180.

KORIEN, J. U. and N. I. BADLER (1982). "Techniques for Generating the Goal-Directed Motion of Articulated Structures", *IEEE Computer Graphics and Applications*, 2(9), pp. 71–81.

KORIEN, J. U. and N. I. BADLER (1983). "Temporal antialiasing in Computer-Generated Animation", in proceedings of SIGGRAPH '83, *Computer Graphics*, 17(3), pp. 377–388.

KREYSZIG, E. (1998). *Advanced Engineering Mathematics, Eighth Edition*, John Wiley & Sons, New York.

LASSETER, J. (1987). "Principles of Traditional Animation Applied to 3D Computer Animation", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 35–44.

LATHROP, O. (1997). *The Way Computer Graphics Works*, John Wiley & Sons, New York.

LAUREL, B. (1990). *The Art of Human-Computer Interface Design*, Addision-Wesley, Reading, MA.

LENGYEL, E. (2002). *Mathematics for 3D Game Programming & Computer Graphics*, Charles River Media, Hingham, MA.

LEVOY, M. (1990). "A Hybrid Ray Tracer for Rendering Polygon and Volume Data", *IEEE Computer Graphics and Applications*, 10(2), pp. 33–40.

LEWIS, J.-P. (1989). "Algorithms for Solid Noise Synthesis", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 263–270.

LIANG, Y.-D. and B. A. BARSKY (1983). "An Analysis and Algorithm for Polygon Clipping." *Communications of the ACM*, 26(11), pp. 868–877.

LIANG, Y.-D. and B. A. BARSKY (1984). "A New Concept and Method for Line Clipping", *ACM Transactions on Graphics*, 3(1), pp. 1–22.

LINDLEY, C. A. (1992). *Practical Ray Tracing in C*, John Wiley & Sons, New York.

LISCHINSKI, D., F. TAMPIERI, and D. P. GREENBERG (1993). "Combining Hierarchical Radiosity and Discontinuity Meshing", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 199–208.

LITWINOWICZ, P. C. (1991). "Inkwell: A $2^1/_2$-D Animation System", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 113–122.

LODDING, K. N. (1983). "Iconic Interfacing", *IEEE Computer Graphics and Applications*, 3(2), pp. 11–20.

LOKE, T.-S., D. TAN, H.-S. SEAH, and M.-H. ER (1992). "Rendering Fireworks Displays", *IEEE Computer Graphics and Applications*, 12(3), pp. 33–43.

LOOMIS, J., H. POIZNER, U. BELLUGI, A. BLAKEMORE, and J. HOLLERBACH (1983). "Computer-Graphics Modeling of American Sign Language", in proceedings of SIGGRAPH '83, *Computer Graphics*, 17(3), pp. 105–114.

LOPES, A. and K. BRODLIE (2003). "Improving the Robustness and Accuracy of the Marching-Cubes Algorithm for Isosurfaces", IEEE Transactions on Visualization and Computer Graphics, 9(1), pp. 16–29.

LORENSON, W. E. and H. CLINE (1987). "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 163–169.

MACKINLAY, J. D., S. K. CARD, and G. G. ROBERTSON (1990). "Rapid Controlled Movement Through a Virtual 3D Workspace", in proceedings of SIGGRAPH '90, *Computer Graphics*, pp. 171–176.

MACKINLAY, J. D., G. G. ROBERTSON, and S. K. CARD (1991). "The Perspective Wall: Detail and Context Smoothly Integrated", *CHI '91*, pp. 173–179.

MAESTRI, G. (1999). *[digital] Character Animation 2, Volume 1—Essential Techniques*, New Riders Publishing, Indianapolis, IN.

MAESTRI, G. (2002). *[digital] Character Animation 2, Volume 2—Advanced Techniques*, New Riders Publishing, Indianapolis, IN.

MAGNENAT-THALMANN, N. and D. THALMANN (1985). *Computer Animation: Theory and Practice*, Springer-Verlag, Tokyo.

MAGNENAT-THALMANN, N. and D. THALMANN (1987). *Image Synthesis*, Springer-Verlag, Tokyo.

MAGNENAT-THALMANN, N. and D. THALMANN (1991). "Complex Models for Animating Synthetic Actors", *IEEE Computer Graphics and Applications*, 11(5), pp. 32–45.

MANDELBROT, B. B. (1977). *Fractals: Form, Chance, and Dimension*, Freeman Press, San Francisco.

MANDELBROT, B. B. (1982). *The Fractal Geometry of Nature*, Freeman Press, New York.

MANTYLA, M. (1988). *An Introduction to Solid Modeling*, Computer Science Press, Rockville, MD.

MAX, N. L. and D. M. LERNER (1985). "A Two-and-a-Half-D Motion Blur Algorithm", in proceedings of SIGGRAPH '85, *Computer Graphics*, 19(3), pp. 85–94.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

MAX, N. L. (1986). "Atmospheric Illumination and Shadows", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 117–124.

MAX, N. L. (1990). "Cone-Spheres", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 59–62.

McCARTHY, M. and A. DESCARTES (1998). *Reality Architecture: Building 3D Worlds with Java and VRML*, Prentice-Hall Europe, United Kingdom.

MEYER, G. W. and D. P. GREENBERG (1988). "Color-Defective Vision and Computer Graphics Displays", *IEEE Computer Graphics and Applications*, 8(5), pp. 28–40.

MEYERS, D., S. SKINNER, and K. SLOAN (1992). "Surfaces from Contours", *ACM Transactions on Graphics*, 11(3), pp. 228–258.

MIANO, J. (1999). *Compressed Image File Formats*, Addison-Wesley/ACM Press, New York.

MILLER, G. S. P. (1988). "The Motion Dynamics of Snakes and Worms", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 169–178.

MILLER, J. V., D. E. BREEN, W. E. LORENSON, R. M. O'BARA, and M. J. WOZNY (1991). "Geometrically Deformed Models: A Method for Extracting Closed Geometric Models from Volume Data", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 217–226.

MITCHELL, D. P. (1991). "Spectrally Optimal Sampling for Distribution Ray Tracing", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 157–165.

MITCHELL, D. P. and P. HANRAHAN (1992). "Illumination from Curved Reflectors", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 283–291.

MITROO, J. B., N. HERMAN, and N. I. BADLER (1979). "Movies from Music: Visualizing Music Compositions", in proceedings of SIGGRAPH '79, *Computer Graphics*, 13(2), pp. 218–225.

MIYATA, K. (1990). "A Method of Generating Stone Wall Patterns", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 387–394.

MOLNAR, S., J. EYLES, and J. POULTON (1992). "PixelFlow: High-Speed Rendering Using Image Composition", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 231–240.

MOON, F. C. (1992). *Chaotic and Fractal Dynamics*, John Wiley & Sons, New York.

MOORE, M. and J. WILHELMS (1988). "Collision Detection and Response for Computer Animation", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 289–298.

MORTENSON, M. E. (1985). *Geometric Modeling*, John Wiley & Sons, New York.

MURAKI, S. (1991). "Volumetric Shape Description of Range Data Using the 'Blobby Model' ", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 227–235.

MUSGRAVE, F. K., C. E. KOLB, and R. S. MACE (1989). "The Synthesis and Rendering of Eroded Fractal Terrains", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 41–50.

MYERS, B. A. and W. BUXTON (1986). "Creating High-Interactive and Graphical User Interfaces by Demonstration", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 249–258.

NAYLOR, B., J. AMANATIDES, and W. THIBAULT (1990). "Merging BSP Trees Yields Polyhedral Set Operations", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 115–124.

NICHOLL, T. M., D. T. LEE, and R. A. NICHOLL (1987). "An Efficient New Algorithm for 2D Line Clipping: Its Development and Analysis", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 253–262.

NIELSON, G. M., B. SHRIVER, and L. ROSENBLUM, ed. (1990). *Visualization in Scientific Computing*, IEEE Computer Society Press, Los Alamitos, CA.

NIELSON, G. M. (1993). "Scattered Data Modeling", *IEEE Computer Graphics and Applications*, 13(1), pp. 60–70.

NISHIMURA, H. (1985). "Object Modeling by Distribution Function and a Method of Image Generation", *Journal Electronics Comm. Conf. '85*, J68(4), pp. 718–725.

NISHITA, T. and E. NAKAMAE (1986). "Continuous-Tone Representation of Three-Dimensional Objects Illuminated by Sky Light", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 125–132.

NISHITA, T., T. SIRAI, K. TADAMURA, and E. NAKAMAE (1993). "Display of the Earth Taking into Account Atmospheric Scattering", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 175–182.

NORTON, A. (1982). "Generation and Display of Geometric Fractals in 3-D", in proceedings of SIGGRAPH '82, *Computer Graphics*, 16(3), pp. 61–67.

NSF INVITATIONAL WORKSHOP (1992). "Research Directions in Virtual Environments", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(3), pp. 153–177.

OKABE, H., H. IMAOKA, T. TOMIHA, and H. NIWAYA (1992). "Three-Dimensional Apparel CAD System", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 105–110.

OLANO, M., J. C. HART, W. HEIDRICH, and M. McCOOL (2002). *Real-Time Shading*, A. K. Natick, MA.

OPPENHEIMER, P. E. (1986). "Real-Time Design and Animation of Fractal Plants and Trees", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 55–64.

O'ROURKE, M. (1998). *Principles of Three-Dimensional Computer Animation, Revised Edition*, W. W. Norton, New York.

OSF/MOTIF (1989). *OSF/Motif Style Guide*, Open Software Foundation, Prentice-Hall, Englewood Cliffs, NJ.

PAETH, A. W., ed. (1995). *Graphics Gems V*, Morgan Kaufmann, San Diego, CA.

PAINTER, J. and K. SLOAN (1989). "Antialiased Ray Tracing by Adaptive Progressive Refinement", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 281–288.

PALMER, I. (2001). *Essential Java 3D Fast*, Springer-Verlag, London.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

PANG, A. T. (1990). "Line-Drawing Algorithms for Parallel Machines", *IEEE Computer Graphics and Applications*, 10(5), pp. 54–59.

PAO, Y. C. (1984). *Elements of Computer-Aided Design*. John Wiley & Sons, New York.

PARENT, R. (2002). *Computer Animation: Algorithms and Techniques*, Morgan Kaufmann, San Francisco, CA.

PAVLIDIS, T. (1982). *Algorithms For Graphics and Image Processing*, Computer Science Press, Rockville, MD.

PAVLIDIS, T. (1983). "Curve Fitting with Conic Splines", *ACM Transactions on Graphics*, 2(1), pp. 1–31.

PEACHEY, D. R. (1986). "Modeling Waves and Surf", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 65–74.

PEITGEN, H.-O. and P. H. RICHTER (1986). *The Beauty of Fractals*, Springer-Verlag, Berlin.

PEITGEN, H.-O. and D. SAUPE, eds. (1988). *The Science of Fractal Images*, Springer-Verlag, Berlin.

PENTLAND, A. and J. WILLIAMS (1989). "Good Vibrations: Modal Dynamics for Graphics and Animation", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 215–222.

PERLIN, K. and E. M. HOFFERT (1989). "Hypertexture", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 253–262.

PHONG, B. T. (1975). "Illumination for Computer-Generated Images", *Communications of the ACM*, 18(6), pp. 311–317.

PIEGL, L. and W. TILLER (1997). *The NURBS Book*, Springer-Verlag, New York.

PINEDA, J. (1988). "A Parallel Algorithm for Polygon Rasterization", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 17–20.

PITTEWAY, M. L. V. and D. J. WATKINSON (1980). "Bresenham's Algorithm with Gray Scale", *Communications of the ACM*, 23(11), pp. 625–626.

PLATT, J. C. and A. H. BARR (1988). "Constraint Methods for Flexible Models", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 279–288.

POCOCK, L. and J. ROSEBUSH (2002). *The Computer Animator's Technical Handbook*, Morgan Kaufmann, San Francisco, CA.

POTMESIL, M. and I. CHAKRAVARTY (1982). "Synthetic Image Generation with a Lens and Aperture Camera Model", *ACM Transactions on Graphics*, 1(2), pp. 85–108.

POTMESIL, M. and I. CHAKRAVARTY (1983). "Modeling Motion Blur in Computer-Generated Images", in proceedings of SIGGRAPH '83, *Computer Graphics*, 17(3), pp. 389–399.

POTMESIL, M. and E. M. HOFFERT (1987). "FRAMES: Software Tools for Modeling, Rendering and Animation of 3D Scenes", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 85–93.

POTMESIL, M. and E. M. HOFFERT (1989). "The Pixel Machine: A Parallel Image Computer", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 69–78.

PRATT, W. K. (1978). *Digital Image Processing*, John Wiley & Sons, New York.

PREPARATA, F. P. and M. I. SHAMOS (1985). *Computational Geometry*, Springer-Verlag, New York.

PRESS, W. H., S. A. TEUKOLSKY, W. T. VETTERLING, and B. P. FLANNERY (1993). *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*, Cambridge University Press, Cambridge, England.

PRESS, W. H., S. A. TEUKOLSKY, W. T. VETTERLING, and B. P. FLANNERY (2002). *Numerical Recipes in C++: The Art of Scientific Computing, Second Edition*, Cambridge University Press, Cambridge, England.

PRESTON, K., FAGAN, HUANG, and PRYOR (1984). "Computing in Medicine", *Computer*, 17(10), pp. 294–313.

PRUSINKIEWICZ, P., M. S. HAMMEL, and E. MJOLSNESS (1993). "Animation of Plant Development", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 351–360.

PRUYN, P. W. and D. P. GREENBERG (1993). "Exploring 3D Computer Graphics in Cockpit Avionics", *IEEE Computer Graphics and Applications*, 13(3), pp. 28–35.

QUEK, L.-H. and D. HEARN (1988). "Efficient Space-Subdivision Methods in Ray-Tracing Algorithms", University of Illinois, Department of Computer Science Report UIUCDCS-R-88-1468.

RAIBERT, M. H. and J. K. HODGINS (1991). "Animation of Dynamic Legged Locomotion", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 349–358.

RAO, K. R. and P. YIP (1990). *Discrete Cosine Transform*, Academic Press, New York.

REEVES, W. T. (1983a). "Particle Systems: A Technique for Modeling a Class of Fuzzy Objects", *ACM Transactions on Graphics*, 2(2), pp. 91–108.

REEVES, W. T. (1983b). "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects", in proceedings of SIGGRAPH '83, *Computer Graphics*, 17(3), pp. 359–376.

REEVES, W. T. and R. BLAU (1985). "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", in proceedings of SIGGRAPH '85, *Computer Graphics*, 19(3), pp. 313–321.

REEVES, W. T., D. H. SALESIN, and R. L. COOK (1987). "Rendering Antialiased Shadows with Depth Maps", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 283–291.

REQUICHA, A. A. G. and J. R. ROSSIGNAC (1992). "Solid Modeling and Beyond", *IEEE Computer Graphics and Applications*, 12(5), pp. 31–44.

REYNOLDS, C. W. (1982). "Computer Animation with Scripts and Actors", in proceedings of SIGGRAPH '82, *Computer Graphics*, 16(3), pp. 289–296.

REYNOLDS, C. W. (1987). "Flocks, Herds, and Schools: A Distributed Behavioral Model", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 25–34.

RHODES, M. L., et al. (1983). "Computer Graphics and An Interactive Stereotactic System for CT-Aided

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Neurosurgery", *IEEE Computer Graphics and Applications*, 3(5), pp. 31–37.

RIESENFELD, R. F. (1981). "Homogeneous Coordinates and Projective Planes in Computer Graphics", *IEEE Computer Graphics and Applications*, 1(1), pp. 50–55.

ROBERTSON, P. K. (1988). "Visualizing Color Gamuts: A User Interface for the Effective Use of Perceptual Color Spaces in Data Displays", *IEEE Computer Graphics and Applications*, 8(5), pp. 50–64.

ROBERTSON, G. G., J. D. MACKINLAY and S. K. CARD (1991). "Cone Trees: Animated 3D Visualizations of Hierarchical Information", *CHI '91*, pp. 189–194.

ROGERS, D. F. and R. A. EARNSHAW, eds. (1987). *Techniques for Computer Graphics*, Springer-Verlag, New York.

ROGERS, D. F. and J. A. ADAMS (1990). *Mathematical Elements for Computer Graphics*, McGraw-Hill, New York.

ROGERS, D. F. (1998). *Procedural Elements for Computer Graphics*, McGraw-Hill, New York.

ROSENTHAL, D. S. H., J. C. MICHENER, G. PFAFF, R. KESSEMER, and M. SABIN (1982). "The Detailed Semantics of Graphics Input Devices", in proceedings of SIGGRAPH '82, *Computer Graphics*, 16(3), pp. 33–38.

RUBINE, D. (1991). "Specifying Gestures by Example", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 329–337.

RUSHMEIER, H. and K. TORRANCE (1987). "The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 293–302.

RUSHMEIER, H. E. and K. E. TORRANCE (1990). "Extending the Radiosity Method to Include Specularly Reflecting and Translucent Materials", *ACM Transactions on Graphics*, 9(1), pp. 1–27.

SABELLA, P. (1988). "A Rendering Algorithm for Visualizing 3D Scalar Fields", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 51–58.

SABIN, M. A. (1985). "Contouring: The State of the Art", in *Fundamental Algorithms for Computer Graphics*, R. A. Earnshaw, ed., Springer-Verlag, Berlin, pp. 411–482.

SAKAGUCHI, H., S. L. KENT, and T. COX (2001). *The Making of Final Fantasy, The Spirits Within*, Brady Games, Indianapolis, IN.

SALESIN, D. and R. BARZEL (1993). "Adjustable Tools: An Object-Oriented Interaction Metaphor", *ACM Transactions on Graphics*, 12(1), pp. 103–107.

SAMET, H. and M. TAMMINEN (1985). "Bintrees, CSG Trees, and Time", in proceedings of SIGGRAPH '85, *Computer Graphics*, 19(3), pp. 121–130.

SAMET, H. and R. E. WEBBER (1985). "Sorting a Collection of Polygons using Quadtrees", *ACM Transactions on Graphics*, 4(3), pp. 182–222.

SAMET, H. and R. E. WEBBER (1988a). "Hierarchical Data Structures and Algorithms for Computer Graphics: Part 1", *IEEE Computer Graphics and Applications*, 8(4), pp. 59–75.

SAMET, H. and R. E. WEBBER (1988b). "Hierarchical Data Structures and Algorithms for Computer Graphics: Part 2", *IEEE Computer Graphics and Applications*, 8(3), pp. 48–68.

SCHACHTER, B. J., ed. (1983). *Computer Image Generation*, John Wiley & Sons, New York.

SCHEIFLER, R. W. and J. GETTYS (1986). "The X Window System", *ACM Transactions on Graphics*, 5(2), pp. 79–109.

SCHOENEMAN, C., J. DORSEY, B. SMITS, J. ARVO, and D. GREENBERG (1993). "Painting with Light", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 143–146.

SCHRODER, P. and P. HANRAHAN (1993). "On the Form Factor Between Two Polygons", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 163–164.

SCHWARTZ, M. W., W. B. COWAN, and J. C. BEATTY (1987). "An Experimental Comparison of RGB, YIQ, LAB, HSV, and Opponent Color Models", *ACM Transactions on Graphics*, 6(2), pp. 123–158.

SEDERBERG, T. W. and E. GREENWOOD (1992). "A Physically Based Approached to 2-D Shape Bending", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 25–34.

SEDERBERG, T. W., P. GAO, G. WANG, and H. MU (1993). "2D Shape Blending: An Intrinsic Solution to the Vertex Path Problem", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 15–18.

SEGAL, M. (1990). "Using Tolerances to Guarantee Valid Polyhedral Modeling Results", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 105–114.

SEGAL, M., C. KOROBKIN, R. VAN WIDENFELT, J. FORAN, and P. HAEBERLI (1992). "Fast Shadows and Lighting Effects Using Texture Mapping", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 249–252.

SELMAN, D. (2002). *Java 3D Programming*, Manning Publications, Greenwich, CT.

SEQUIN, C. H. and E. K. SMYRL (1989). "Parameterized Ray-Tracing", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 307–314.

SHERR, S. (1993). *Electronic Displays*, John Wiley & Sons, New York.

SHILLING, A. and W. STRASSER (1993). "EXACT: Algorithm and Hardware Architecture for an Improved A-Buffer", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 85–92.

SHIRLEY, P. (1990). "A Ray Tracing Method for Illumination Calculation in Diffuse-Specular Scenes", *Graphics Interface '90*, pp. 205–212.

SHIRLEY, P. (2000). *Realistic Ray Tracing*, A. K. Peters, Natick, MA.

SHNEIDERMAN, B. (1986). *Designing the User Interface*, Addison-Wesley, Reading, MA.

SHOEMAKE, K. (1985). "Animating Rotation with Quaternion Curves", in proceedings of SIGGRAPH '85, *Computer Graphics*, 19(3), pp. 245–254.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

SHREINER, D., ed. (2000). *OpenGL Reference Manual, Third Edition*, Addison-Wesley, Reading, MA.

SIBERT, J. L., W. D. HURLEY, and T. W. BLESER (1986). "An Object-Oriented User Interface Management System", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 259–268.

SILLION, F. X. and C. PUECH (1989). "A General Two-Pass Method Integrating Specular and Diffuse Reflection", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 335–344.

SILLION, F. X., J. R. ARVO, S. H. WESTIN, and D. P. GREENBERG (1991). "A Global Illumination Solution for General Reflectance Distributions", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 187–196.

SILLION, F. X. and C. PUECH (1994). *Radiosity and Global Illumination*, Morgan Kaufmann, San Francisco, CA.

SIMS, K. (1990). "Particle Animation and Rendering Using Data Parallel Computation", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 405–413.

SIMS, K. (1991). "Artificial Evolution for Computer Graphics", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 319–328.

SMITH, A. R. (1978). "Color Gamut Transform Pairs", in proceedings of SIGGRAPH '78, *Computer Graphics*, 12(3), pp. 12–19.

SMITH, A. R. (1979). "Tint Fill", in proceedings of SIGGRAPH '79, *Computer Graphics*, 13(2), pp. 276–283.

SMITH, A. R. (1984). "Plants, Fractals, and Formal Languages", in proceedings of SIGGRAPH '84, *Computer Graphics*, 18(3), pp. 1–10.

SMITH, A. R. (1987). "Planar 2-Pass Texture Mapping and Warping", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 263–272.

SMITS, B. E., J. R. ARVO, and D. H. SALESIN (1992). "An Importance-Driven Radiosity Algorithm", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 273–282.

SNYDER, J. M. and J. T. KAJIYA (1992). "Generative Modeling: A Symbolic System for Geometric Modeling", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 369–378.

SNYDER, J. M., A. R. WOODBURY, K. FLEISCHER, B. CURRIN, and A. H. BARR (1993). "Interval Methods for Multi-Point Collisions between Time-Dependent Curved Surfaces", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 321–334.

SOWIZRAL, H., K. RUSHFORTH, and M. DEERING (2000). *The Java 3D API Specification, Second Edition*, Addison-Wesley, Reading, MA.

SPROULL, R. F. and I. E. SUTHERLAND (1968). "A Clipping Divider", AFIPS Fall Joint Computer Conference.

STAM, J. and E. FIUME (1993). "Turbulent Wind Fields for Gaseous Phenomena", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 369–376.

STETTNER, A. and D. P. GREENBERG (1989). "Computer Graphics Visualization for Acoustic Simulation", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 195–206.

STRASSMANN, S. (1986). "Hairy Brushes", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 225–232.

STRAUSS, P. S. and R. CAREY (1992). "An Object-Oriented 3D Graphics Toolkit", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 341–349.

STROTHOTTE, T. and S. SCHLECHTWEG (2002). *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*, Morgan Kaufmann, San Francisco, CA.

SUNG, H. C. K., G. ROGERS, and W. J. KUBITZ (1990). "A Critical Evaluation of PEX", *IEEE Computer Graphics and Applications*, 10(6), pp. 65–75.

SUTHERLAND, I. E. (1963). "Sketchpad: A Man-Machine Graphical Communication System", *AFIPS Spring Joint Computer Conference*, 23, pp. 329–346.

SUTHERLAND, I. E. and G. W. Hodgman (1974). "Reentrant Polygon Clipping", *Communications of the ACM*, 17(1), pp. 32–42.

SUTHERLAND, I. E., R. F. SPROULL, and R. SCHUMACKER (1974). "A Characterization of Ten Hidden Surface Algorithms", *ACM Computing Surveys*, 6(1), pp. 1–55.

SWEZEY, R. W. and E. G. DAVIS (1983). "A Case Study of Human Factors Guidelines in Computer Graphics", *IEEE Computer Graphics and Applications*, 3(8), pp. 21–30.

TAKALA, T. and J. HAHN (1992). "Sound Rendering", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 211–220.

TANNAS, JR., L. E., ed. (1985). *Flat-Panel Displays and CRTs*, Van Nostrand Reinhold, New York.

TAUBMAN, D. and M. MARCELLIN (2001). *JPEG 2000: Image-Compression Fundamentals, Standards, and Practice*, Kluwer Academic Publishers, Norwell, MA.

TELLER, S. and P. HANRAHAN (1993). "Global Visibility Algorithms for Illumination Computations", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 239–246.

TERZOPOULOS, D., J. PLATT, A. H. BARR, et al. (1987). "Elastically Deformable Models", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 205–214.

THALMANN, N. M. and D. THALMANN (1985). *Computer Animation: Theory and Practice*, Springer-Verlag, Tokyo.

THALMANN, D., ed. (1990). *Scientific Visualization and Graphics Simulation*, John Wiley & Sons, Chichester, England.

THIBAULT, W. C. and B. F. NAYLOR (1987). "Set Operations on Polyhedra using Binary Space Partitioning Trees", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 153–162.

THOMAS, B. and W. LEFKON (1997). *Disney's Art of Animation from Mickey Mouse to Hercules*, Hyperion Press, New York.

THOMAS, F., O. JOHNSON, and C. JOHNSTON (1995). *The Illusion of Life: Disney Animation*, Hyperion Press, New York.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

TORBERG, J. G. (1987). "A Parallel Processor Architecture for Graphics Arithmetic Operations", in proceedings of SIGGRAPH '87, *Computer Graphics*, 21(4), pp. 197–204.

TORRANCE, K. E. and E. M. SPARROW (1967). "Theory for Off-Specular Reflection from Roughened Surfaces", *Journal of the Optical Society of America*, 57(9), pp. 1105–1114.

TRAVIS, D. (1991). *Effective Color Displays*, Academic Press, London.

TUFTE, E. R. (1990). *Envisioning Information*, Graphics Press, Cheshire, CN.

TUFTE, E. R. (1997). *Visual Explanations: Images and Quantities, Evidence and Narrative*, Graphics Press, Cheshire, CN.

TUFTE, E. R. (2001). *The Visual Display of Quantitative Information, Second Edition*, Graphics Press, Cheshire, CN.

TURKOWSKI, K. (1982). "Antialiasing Through the Use of Coordinate Transformations", *ACM Transactions on Graphics*, 1(3), pp. 215–234.

UPSON, C. and M. KEELER (1988). "VBUFFER: Visible Volume Rendering", in proceedings of SIGGRAPH '88, *Computer Graphics*, 22(4), pp. 59–64.

UPSON, C., et al. (1989). "The Application Visualization System: A Computational Environment for Scientific Visualization", *IEEE Computer Graphics and Applications*, 9(4), pp. 30–42.

UPSTILL, S. (1989). *The RenderMan Companion*, Addison-Wesley, Reading, MA.

VAN WIJK, J. J. (1991). "Spot Noise-Texture Synthesis for Data Visualization", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 309–318.

VEENSTRA, J. and N. AHUJA (1988). "Line Drawings of Octree-Represented Objects", *ACM Transactions on Graphics*, 7(1), pp. 61–75.

VELHO, L. and J. D. M. GOMES (1991). "Digital Halftoning with Space-Filling Curves", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 81–90.

VON HERZEN, B., A. H. BARR, and H. R. ZATZ (1990). "Geometric Collisions for Time-Dependent Parametric Surfaces", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 39–48.

WALLACE, V. L. (1976). "The Semantics of Graphic Input Devices", in proceedings of SIGGRAPH '76, *Computer Graphics*, 10(1), pp. 61–65.

WALLACE, J. R., K. A. ELMQUIST, and E. A. HAINES (1989). "A Ray-Tracing Algorithm for Progressive Radiosity", in proceedings of SIGGRAPH '89, *Computer Graphics*, 23(3), pp. 315–324.

WALSH, A. E. and D. GEHRINGER (2002). *Java 3D*, Prentice-Hall, Upper Saddle River, NJ.

WANGER, L. R., J. A. FERWERDA, and D. P. GREENBERG (1992). "Perceiving Spatial Relationships in Computer-Generated Images", *IEEE Computer Graphics and Applications*, 12(3), pp. 44–58.

WARN, D. R. (1983). "Lighting Controls for Synthetic Images", in proceedings of SIGGRAPH '83, *Computer Graphics*, 17(3), pp. 13–21.

WATT, A. (1989). *Fundamentals of Three-Dimensional Computer Graphics*, Addison-Wesley, Wokingham, England.

WATT, M. (1990). "Light-Water Interaction Using Backward Beam Tracing", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 377–386.

WATT, A. and M. WATT (1992). *Advanced Animation and Rendering Techniques*, Addison-Wesley, Wokingham, England.

WEGHORST, H., G. HOOPER, and D. P. GREENBERG (1984). "Improved Computational Methods for Ray Tracing", *ACM Transactions on Graphics*, 3(1), pp. 52–69.

WEIL, J. (1986). "The Synthesis of Cloth Objects", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 49–54.

WEILER, K. and P. ATHERTON (1977). "Hidden-Surface Removal Using Polygon Area Sorting", in proceedings of SIGGRAPH '77, *Computer Graphics*, 11(2), pp. 214–222.

WEILER, K. (1980). "Polygon Comparison Using a Graph Representation", in proceedings of SIGGRAPH '80, *Computer Graphics*, 14(3), pp. 10–18.

WEINBERG, R. (1978) "Computer Graphics in Support of Space-Shuttle Simulation", in proceedings of SIGGRAPH '78, *Computer Graphics*, 12(3), pp. 82–86.

WELCH, T. (1984). "A Technique for High-Performance Data Compression", *IEEE Computer*, 17(6), pp. 8–19.

WERNECKE, J. (1994). *The Inventor Mentor*, Addison-Wesley, Reading, MA.

WESTIN, S. H., J. R. ARVO, and K. E. TORRANCE (1992). "Predicting Reflectance Functions from Complex Surfaces", in proceedings of SIGGRAPH '92, *Computer Graphics*, 26(2), pp. 255–264.

WESTOVER, L. (1990). "Footprint Evaluation for Volume Rendering", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 367–376.

WHITTED, T. (1980). "An Improved Illumination Model for Shaded Display", *Communications of the ACM*, 23(6), pp. 343–349.

WHITTED, T. and D. M. WEIMER (1982). "A Software Testbed for the Development of 3D Raster Graphics Systems", *ACM Transactions on Graphics*, 1(1), pp. 43–58.

WHITTED, T. (1983). "Antialiased Line Drawing Using Brush Extrusion", in proceedings of SIGGRAPH '83, *Computer Graphics*, 17(3), pp. 151–156.

WILHELMS, J. (1987). "Toward Automatic Motion Control", *IEEE Computer Graphics and Applications*, 7(4), pp. 11–22.

WILHELMS, J. and A. VAN GELDER (1991). "A Coherent Projection Approach for Direct Volume Rendering", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 275–284.

WILHELMS, J. and A. VAN GELDER (1992). "Octrees for Faster Isosurface Generation", *ACM Transactions on Graphics*, 11(3), pp. 201–227.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

WILLIAMS, L. (1983). "Pyramidal Parametrics", in proceedings of SIGGRAPH '83, *Computer Graphics*, 17(3), pp. 1–11.

WILLIAMS, L. (1990). "Performance-Driven Facial Animation", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 235–242.

WITKIN, A. and W. WELCH (1990). "Fast Animation and Control of Nonrigid Structures", in proceedings of SIGGRAPH '90, *Computer Graphics*, 24(4), pp. 243–252.

WITKIN, A. and M. KASS (1991). "Reaction-Diffusion Textures", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 299–308.

WOLFRAM, S. (1984) "Computer Software in Science and Mathematics", *Scientific American*, 251(3), 188–203.

WOLFRAM, S. (1991). *Mathematica*, Addison-Wesley, Reading, MA.

WOO, A., P. POULIN, and A. FOURNIER (1990). "A Survey of Shadow Algorithms", *IEEE Computer Graphics and Applications*, 10(6), pp. 13–32.

WOO M., J. NEIDER, T. DAVIS, and D. SHREINER (1999). *OpenGL Programming Guide, Third Edition*, Addison-Wesley, Reading, MA.

WRIGHT, W. E. (1990). "Parallelization of Bresenham's Line and Circle Algorithms", *IEEE Computer Graphics and Applications*, 10(5), pp. 60–67.

WU, X. (1991). "An Efficient Antialiasing Technique", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 143–152.

WYSZECKI, G. and W. S. STILES (1982). *Color Science*, John Wiley & Sons, New York.

WYVILL, G., B. WYVILL, and C. McPHEETERS (1987). "Solid Texturing of Soft Objects", *IEEE Computer Graphics and Applications*, 7(12), pp. 20–26.

YAEGER, L., C. UPSON, and R. MYERS (1986). "Combining Physical and Visual Simulation: Creation of the Planet Jupiter for the Film '2010' ", in proceedings of SIGGRAPH '86, *Computer Graphics*, 20(4), pp. 85–94.

YAGEL, R., D. COHEN, and A. KAUFMAN (1992). "Discrete Ray Tracing", *IEEE Computer Graphics and Applications*, 12(5), pp. 19–28.

YAMAGUCHI, K., T. L. KUNII, and F. FUJIMURA (1984). "Octree-Related Data Structures and Algorithms", *IEEE Computer Graphics and Applications*, 4(1), pp. 53–59.

YESSIOS, C. I. (1979). "Computer Drafting of Stones, Wood, Plant, and Ground Materials", in proceedings of SIGGRAPH '79, *Computer Graphics*, 13(2), pp. 190–198.

YOUNG, D. A. (1990). *The X Window System—Programming and Applications with Xt, OSF/Motif Edition*, Prentice-Hall, Englewood Cliffs, NJ.

ZELEZNICK, R. C., et al. (1991). "An Object-Oriented Framework for the Integration of Interactive Animation Techniques", in proceedings of SIGGRAPH '91, *Computer Graphics*, 25(4), pp. 105–112.

ZELTZER, D. (1982). "Motor Control Techniques for Figure Animation", *IEEE Computer Graphics and Applications*, 2(9), pp. 53–60.

ZHANG, Y. and R. E. WEBBER (1993). "Space Diffusion: An Improved Parallel Halftoning Technique Using Space-Filling Curves", in proceedings of SIGGRAPH '93, *Computer Graphics*, pp. 305–312.

ZIV, J. and A. LEMPEL (1977). "A Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory*, 23(3), pp. 337–343.

ZIV, J. and A. LEMPEL (1978). "Compression of Individual Sequences via Variable-Rate Coding", *IEEE Transactions on Information Theory*, 24(5), pp. 530–536.

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

索　引

A

Absolute: 绝对:
coordinates 坐标 3.1.2
value (imaginary number) 值(虚数) A.6.3
A-buffer algorithm A缓存算法 9.4
Acoustical touch panel 声学触摸板 2.4.8
Acoustic digitizer 声学数据板 2.4.6
Accelerating: 加速:
anode (CRT) 阳极 2.1.1
voltage (CRT) 电压 2.1.1
Active edge list 活化边表 4.1
Active-matrix LCD 激活矩阵 2.1.5
Adaptive sampling 适应性采样 10.11.8
Adaptive spatial subdivision: 适应性空间分割:
BSP tree BSP树 8.22
ray tracing 光线跟踪 10.11
Additive color model 加色模型 12.3 ~ 12.4
Adobe Photoshop format Adobe Photoshop格式 15.4.6
Affine transformation 仿射变换 5.16
AGL (Apple GL) AGL 2.9.2
Aliasing 走样 4.17
Alignment (text) 对齐(文字) 4.15
Ambient light (see also Illumination models) 环境光(又见光照模型) 10.2
Ambient-reflection coefficient 环境光反射系数 10.3.2
American National Standards Institute (ANSI) 美国国家标准化组织(ANSI) 2.8.3, 15.4
Angle: 角度:
direction 方向角 A.2.2
field-of-view 视场角 7.8.6
of incidence (light) 入射(光)角 10.3.2
phase 相位角 13.7.1
refraction 折射角 10.4.2
rotation 旋转角 5.1.2
specular-reflection 镜面反射角 10.3.3
Angstrom 埃 12.1.1
Angular intensity attenuation (light source) 角强度衰减 10.1.5
Animation 动画: 5.4.8, 13
accelerations 模拟加速度 13.6.2
action specifications 动作描述 13.5
anticipation 运动预期 13.3
applications 应用 1.2, 1.6, 1.7
articulated figures 关节链形体 13.8
cels 13.6
color-table transformations 颜色表变换 13.1.2
designing 动画序列设计 13.2
direct motion specification 直接运动描述 13.7.1
double buffering 双缓存 13.1.1
dynamics 动力学 13.7.3
follow through 运动完结 13.3
frame-by-frame 逐帧动画 13
frame rates 动画帧率 13.1.1
functions 动画功能 13.4
goal-directed 目标导向 13.7.2
in-betweens 插值帧 13.2
inverse dynamics 反向动力学 13.7.3
inverse kinematics 反向运动学 13.7.3, 13.8
irregular frame rates 不规则动画帧率 13.1.1
key frame 关键帧 13.2
key-frame system 关键帧系统 13.6
kinematics 运动学 13.6, 13.8
Kochanek-Bartels splines Kochanek Bartels样条 8.9.4
languages 语言 13.5
morphing 变形技术 1.6, 13.6.1
motion specification 运动的描述 13.7
Newton's second law of motion 牛顿第二定律 13.7.3
object definitions 对象定义 13.2
panning effects 移镜效果 7.3.4
parameterized system 参数系统 13.5
periodic motions 周期性运动 13.9
physically based 基于物理的 13.6, 13.7.3
raster methods 光栅方法 13.1
real-time 实时动画 2.2.1, 13
rigid-body degrees of freedom 刚体自由度 13.5
rotational 旋转运动 5.1.2, 13.8
round-off and approximation errors 近似误差 5.4.8, 13.9
sampling rates 采样率 13.9
scene description 场景描述 13.5
scripting system 剧本系统 13.5
squash and stretch 挤压和拉伸 13.3
staging 分级 13.3

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

storyboard　故事情节　13.2
timing　定时　13.3
traditional cartoon techniques　传统的卡通技术　13.3
undersampled periodic-motion displays　低取样周期运动显示　13.9
ANSI（American National Standards Institute）　ANSI（美国国家标准化组织）　2.8.3
Antialiasing（see also Sampling）　反走样（又见取样）　5.6
area boundaries　区域边界的反走样　4.17.7
area sampling　区域取样　4.17.3
filtering　过滤技术　4.17.4
Gaussian filter function　高斯过滤函数　4.17.4
line segments　线段　4.17.1
Nyquist sampling interval　Nyquist取样间隔　4.17
Nyquist sampling frequency　Nyquist取样频率　4.17
Pitteway-Watkinson method　Pitteway-Watkinson算法　4.17.7
pixel phasing　像素移相　4.17
postfiltering　后过滤　4.17
prefiltering　前滤波　4.17
ray tracing　光线跟踪反走样　10.11.8
stochastic sampling　随机采样　10.11.9
supersampling　过取样　4.17.1
surface boundaries　面边界反走样　4.17.7
texture maps　纹理映射　10.17
weighting masks　子像素加权掩模　4.17.2
weighting surface　加权表面　4.17.4
API（Application programming interface）　API（应用编程接口）　2.8
Apple GL（AGL）　2.9.2
Application icon　应用图符　11.6.2
Applications（see also Graphics applications）　应用（又见图形学应用）　1
Approximation spline　逼近样条　8.8.1
Area clipping　区域裁剪　6.8
Area filling（see also Fill-area algorithms）　区域填充（又见区域填充算法）　4.1
Area sampling　区域取样　4.17
Area-subdivision visibility algorithm　区域细分　9.8
Arithmetic encoding　算术编码　15.3.5
Art applications（see also Computer art）　艺术应用（又见计算机艺术）　1.6
Articulated figures（animation）　关节链形体（动画）　13.8
Artificial reality（see Virtual reality）　人造现实（又见虚拟现实）
Aspect ratio　长宽比　2.1.2
Atmospheric effects（lighting model）　雾气效果（光照模型）　10.5
Attenuation functions（light）　衰减函数（光）　10.1.3
Attribute parameters　属性参数　2.8.2
brush and pen styles　画刷和画笔类型　4.5.3
characters and text　字符和文字　4.15
color　颜色　4.2
curved lines　曲线　4.6
fill areas　填充区　4.9
gray scale　灰度　4.2.3
marker symbols　标记符号　4.15
points　点　4.4
polygons　多边形　4.9
straight-line segment　直线段　4.5
system list　系统属性表　4
Attribute tables（polygon）　属性表（多边形）　3.15.6
Auxiliary buffer　辅助缓存　3.19.2
Axis:　坐标轴：
principal　主轴　7.6.1
reflection　反射　5.5.1
rotation　旋转　5.1.2
shear　错切　5.5.2
Axis vector（rotation）　轴向量（旋转）　5.11.2
Axis vectors（basis）　轴向量（基）　A.4
Axonometric projection　轴测投影　7.6.1

B

Back buffer　后缓存　13.1
Back face（polygon）　后向面（多边形）　3.15.8
Back-face elimination　后向面消除　9.2
Back plane（clipping）　后平面（裁剪）　7.6.3
Background（ambient）light　背景（环境）光　10.2
Bar chart　直方图　1.1
Barn doors（light control）　挡光板（光线控制）　10.1.6
Baseline（character）　基线（字符）　4.15
Base vector（see also Basis）　基向量（又见基）　A.4
Basic illumination model　基本光照模型　10.3
Basis:　基:
coordinate vectors　坐标向量　A.3
normal　标准基　A.4.2
orthogonal　正交基　A.4.2
orthonormal　正交标准基　A.4.2
Basis functions（see also Blending functions）　基函数（又见混合函数）　8.8.4
Basis matrix（spline）　基本矩阵　8.8.4
Beam-penetration CRT（see also Cathode-ray tube）　电子束穿透法CRT（又见阴极射线管）　2.1.4
Bernstein polynomials　Bernstein多项式　8.38
Beta parameter　beta参数　8.14
Beta-spline　beta样条　8.14
continuity conditions　连续性条件　8.14.1
cubic periodic　三次周期性beta样条　8.14.2

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Bevel polyline join 斜切折线连接 4.5.1
Bézier:
blending functions 混合函数 8.10.1
B-spline conversions B样条转换 8.16
closed curve 封闭曲线 8.10.4
cubic curve 三次曲线 8.10.4
curves 曲线 8.1
design techniques 设计技术 8.10.4
matrix 矩阵 8.10.5
properties 特性 8.10.3
surfaces 曲面 8.11
Bias parameter（spline） 偏离参数（样条） 8.9.4
Big endian 高位在前 15.1
Binary space-partitioning tree（see also BSP tree） 二叉空间分割树（又见BSP树） 8.22, 9.7
Binding（language） 绑定（语言） 2.8.3
Binomial coefficients 二项式系数 8.10.1
Bisection root finding 二分求根算法 A.14.2
Bitblt（bit-block transfer） Bitblt（块移动） 5.6
Bitmap（see also Frame buffer） 位图（又见帧缓存） 2.1.2, 3.19.1
Bitmap font 位图字体 2.1.3, 3.20
Bit planes（raster-scan system） 位平面（光栅扫描系统） 2.1.2
Blending functions（spline） 混合函数（样条） 8.10.1
Bézier Bézier样条 8.10.1
B-spline B样条 8.12.1
cardinal cardinal样条 8.9.3
Hermite Hermite样条 8.9.2
Blobby objects 柔性对象 8.7
Block transfer 块移动 5.6
BMP 位图格式 15.4.9
Body: 体:
character 字符体 4.15
nonrigid 非刚性 8.26
rigid 刚体 5.1.1
Bottomline（character） 底线（字符） 4.15
Boundary conditions（spline） 边界条件（样条） 8.8
Boundary-fill algorithms: 边界填充算法:
8-connected region 8-连通区域 4.13.1
4-connected region 4-连通区域 4.13.1
Boundary representation 边界表示 8
Boundary-value problem 边界值问题 A.14.4
Bounding: 包围:
box 包围盒 3.1
rectangle 包围矩形 3.1
volume 包围体 10.11.4
Box: 盒:
covering（fractal） 覆盖（分形） 8.23.3
dimension 维数 8.23.3
filter 过滤 4.17.4
B-rep（boundary representation） B-rep（边界表示） 8
Bresenham's algorithm: Bresenham算法:
circle 画圆弧算法 3.9
line 画线算法 3.5.3
Brightness（light） 亮度（光） 12.1.2
Brownian motion 布朗运动 8.23.6
Brush and pen attributes 画刷和画笔属性 4.5.3
BSP:
tree 树 8.22
visibility algorithm 可见性算法 9.7
B-spline: B样条:
Bézier conversions Bézier样条转换
blending functions 混合函数 8.12.1
Cox-deBoor recursion formulas Cox-deBoor递归公式 8.12.1
cubic 三次B样条 8.12.3
curves B样条曲线 8.12.3
degree parameter 阶参数 8.12.1
knot vector 节点向量 8.12.1
local control 局部控制 8.12
matrix B样条矩阵 8.12.3
nonuniform 非均匀B样条 8.12.5
nonuniform rational（NURB） 非均匀有理B样条（NURB） 8.15
open 开放B样条 8.12.4
periodic 周期性B样条 8.12.2
properties B样条性质 8.12.1
quadratic 二次B样条 8.12.2
rational 有理B样条 8.15
surfaces B样条曲面 8.13
tension parameter B样条张量参数 8.12.3
uniform 均匀B样条 8.12.2
Buffer: 缓存:
auxiliary 辅助缓存 3.19.2
back 后缓存 3.19.2
color 颜色缓存 2.1.2
depth 深度值 3.19.2
frame 帧缓存 2.1.2, 3.1
front 前缓存 3.19.2
left 左缓存 3.19.2
refresh 刷新缓存 2.1.2
right 右缓存 3.19.2
stencil 模板缓存 3.19.2
Bump function 凹凸函数 10.18
Bump mapping 凹凸映射 10.18
Business visualization（see also Data visualization） 商务可视化（又见数据可视化） 1.4
Butt line cap 方线帽 4.5.1
Button box 按钮盒 2.4.1

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

C

Cabinet projection　斜二测投影　7.7.2
CAD（computer-aided design）　CAD（计算机辅助设计）　1.2
CADD（computer-aided drafting and design）　CADD（计算机辅助绘图和设计）　1.2
Callback functions　回调函数　2.9, 11.3.3
Calligraphic（vector）display　笔迹（向量）显示器　2.1.3
CAM（computer-aided manufacturing）　CAM（计算机辅助制造）　1.2
Camera:　照相机:
　circle of confusion　模糊圆　10.11.7
　depth of field　景深　10.11.7
　f-number　*f*数　10.11.7
　focusing effects　聚焦效果　10.11.7
　lens effects　镜头效果　10.11.9
　parameters（illumination model）　照相机参数（光照模型）　10.7
　thin-lens equation　薄透镜公式　10.11.7
　viewing model　观察模型　7.1.1
Candle（standard light source）　烛光　12.3.1
Capline（character）　帽线　4.15
Cardinal spline　cardinal 样条　8.9.3
Cardioid　螺旋线　3.25
Cartesian coordinates　笛卡儿坐标系　A.1
Cartesian-product representation（spline surface）　笛卡儿积　8.8.5, 8.11
CAT（computed axial tomography）　CAT（计算机轴向造影术）　1.8
Cathode rays　阴极射线　2.1.1
Cathode-ray tube（see also Video monitors; Raster-scan display）　阴极射线管（又见视频监视器；光栅扫描显示器）　2.1
　accelerating anode　加速阳极　2.1.1
　accelerating voltage　加速电压　2.1.1
　aspect ratio　纵横比　2.1.2
　beam deflection　电子束偏转　2.1.1
　beam intensity　电子束强度　2.1.2
　beam penetration　电子束穿透法　2.1.4
　cathode　阴极　2.1.1
　color　颜色　2.1.4
　components　元件　2.1.1
　control grid　控制栅极　2.1.1
　electron gun　电子枪　2.1.1
　delta-delta shadow mask　delta-delta荫罩法　2.1.4
　filament　灯丝　2.1.1
　focusing　聚焦　2.1.1
　high-definition　高清晰度　2.1.1
　in-line shadow-mask　按线排列荫罩法　2.1.4
　persistence　余辉　2.1.1
　phosphor　荧光　2.1.1
　refresh rate　刷新速率　2.1.1
　resolution　分辨率　2.1.1
　RGB　RGB 颜色模型　2.1.4
　shadow-mask　荫罩法　2.1.4
　spot size　亮点尺寸　2.1.1
Catmull-Rom spline　Catmull-Rom 样条　8.9.3
Cavalier projection　斜等测投影　7.7.2
CAVE　CAVE 系统　1.4, 11
Cell:　单元:
　encoding　编码　2.2.2
　traversal（ray tracing）　遍历　10.11.6
Cels　13.6
Center of projection　投影中心　7.8
Centroid（polygon）　质心（多边形）　A.12
CG API（computer-graphics application programming interface）　CG API（计算机图形应用编程接口）　2.8
CGM（computer-graphics metafile）　CGM（计算机图形元文件）　15.4.2
Character:　字符:
　attributes　属性　4.15
　baseline　字符基线　4.15
　bitmap font　位图字体　3.2
　body　字符体　4.15
　bottomline　字符底线　4.15
　capline　字符帽线　4.15
　clipping　文字裁剪　6.1
　color　颜色　4.16
　descender　下行字符　4.15
　font　字体　3.2
　font cache　字形缓存　3.2
　grid　字符网格　3.2
　height　字符高度　4.15
　kern　字符核　4.15
　legible　字迹清楚　3.2
　marker symbol　标记符号　3.2
　monospace　单一宽度　3.2
　outline fonts　轮廓字体　3.2
　point size　磅大小　4.15
　polymarker　多点标记　3.2
　primitives　图元　3.2
　proportionally spaced　比例间隔　3.21
　raster font　光栅字体　3.2
　readable　可读　3.2
　sans-serif　无衬线　3.2
　serif　有衬线　3.2
　stroke font　笔划字体　3.2
　text precision　文字精度　4.15
　topline　顶线　4.15

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

typeface 字体 3.2
up vector 向上向量 4.15
Characteristic polygon 特征多边形 8.8.1
Chart: 图表:
bar 直方图 1.1
line 折线图 1.1
pie 饼图 1.1
three dimensional 三维统计图 1.1
time 时间表 1.1
Choice input device 选择输入设备 11.2
Chromaticity diagram 色度图: 12.3.3
color gamuts 颜色范围 12.3.4
complimentary colors 互补色 12.3.5
dominant wavelength 主波长 12.3.6
illuminant C 亮白光C 12.3.3
purity 纯度 12.3.7
purple line 紫色线 12.3.3
Chromaticity values 色度值 12.3.2
CIE（International Commission on Illumination） CIE（国际照明委员会） 12.3
Circle: 圆:
equations 方程 3.9.1
properties 特性 3.9.1
symmetry 对称性 3.9.1
Circle of confusion 模糊圆 10.11.7
Circle-generating algorithms 圆生成算法 3.9
Bresenham Bresenham画圆算法 3.9.1
midpoint 中点圆算法 3.9.2
Client computer 客户计算机 2.6
Clipping algorithms 裁剪算法 6.5
Cohen-Sutherland line clipping Cohen-Sutherland线段裁剪算法 6.7.1
concave polygons 凹多边形 6.8.1
curves 曲线 6.8.4
Cyrus-Beck line clipping Cyrus-Beck线段裁剪算法 6.7.2
homogeneous coordinates 齐次坐标系 7.11.1
Liang-Barsky line clipping 梁友栋-Barsky线段裁剪算法 6.7.2
Liang-Barsky polygon clipping 梁友栋-Barsky多边形裁剪算法 6.7.4
Nichol-Lee-Nichol line clipping Nicholl-Lee-Nicholl线段裁剪算法 6.7.3
out code 外部码 6.7.1
parallel methods 并行算法 6.8.1
parametric methods 参数线裁剪算法 6.7.2
points 点裁剪算法 6.6
polygons 多边形裁剪算法 6.8
polyhedra 多面体裁剪算法 7.11.4
region codes 区域码 6.7.1
straight-line segments 线裁剪算法 6.7.1
Sutherland-Hodgman polygon clipping Sutherland-Hodgman多边形裁剪算法 6.8.1
text 文字裁剪算法 6.1
three-dimensional 三维裁剪 7.11
two-dimensional 二维裁剪 6.6
Weiler-Atherton polygon clipping Weiler-Atherton裁剪算法 6.8.2
Clipping planes 裁剪平面 7.2
arbitrary orientations 任意裁剪平面 7.11.6
near and far 近和远裁剪平面 7.6.3
Clipping window 裁剪窗口 6.1, 7.6.3
nonlinear 非线性 6.2
nonrectangular 非矩形裁剪窗口 6.7.4
panning effects 摇镜头效果 6.1
in viewing coordinates 观察坐标系统 6.2.1
viewport mapping 观察映射 6.3.1
in world coordinates 世界坐标系 6.2.2
zooming effects 拉镜头效果 6.1
Cloth modeling 布料建模 8.26
CMY color model CMY颜色模型 12.6
CMYK color model CMYK颜色模型 12.6
Codes（ray tracing） 光线码（光线跟踪） 10.11.9
Coefficient: 系数:
ambient-reflection 环境光反射 10.3.2
diffuse-reflection 漫反射 10.3.2
specular-reflection 镜面反射 10.3.3
transparency 透明系数 10.4
Cohen-Sutherland line clipping Cohen-Sutherland线裁剪算法 6.7.1
Coherence properties 相关特征 4.1
Color（see also Light; Light sources）: 颜色（又见光、光源）:
blending 调和 2.9.5, 4.3.3
brightness 亮度 12.1.2
coding（data visualization） 颜色编码 8.27.1
complementary 基色 12.2.1
chromaticity 色度 12.1.2
gamut 颜色范围 12.2.1
hue 色彩 12.1.1
illuminant C 亮白光C 12.3.3
illumination models 光照模型 10.3
intuitive concepts 直观颜色概念 12.2.2
luminance 亮度 12.3.1
perception 感觉 12.1.2
primary 基色 12.2.1
pure 纯色 12.2.2
purity 纯度 12.1.2
psychological characteristics 颜色的心理学特征 12.1.2

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

saturation　饱和度　12.1.2
selection considerations　选择颜色的考虑　12.9
shades　明暗　12.2.2
spectral　光谱颜色　12.1.1
spectrum（electromagnetic）　光波段（电磁波）12.1.1
tints　色泽　12.2.2
tones　色调　12.2.2
Color buffer（see also Refresh buffer）　颜色缓存（又见刷新缓存）2.1.2
Color CRT　彩色 CRT　2.1.4
Color gamut　颜色范围　12.2.1
Color lookup table　颜色查找表　4.2.2
Color matching functions　颜色匹配函数　12.2.1
Color model　颜色模型　12.2
additive　加色模型　12.3, 12.4
chromaticity diagram　色度图　12.3.3
chromaticity values　色度值　12.3.2
CIE primaries　CIE 基色　12.3
CMY　CMY 颜色模型　12.6
CMYK　CMYK 颜色模型　12.6
color-matching functions　颜色匹配函数　12.2.1
HLS　HLS 颜色模型　12.8
HSB（same as HSV）　HSB颜色模型（与HSV同）
HSV　HSV 颜色模型　12.7
HSV-RGB conversion　HSV-RGB 颜色模型转换　12.7.3
normalized XYZ values　规范化的XYZ值　12.3.2
primaries　基色　12.2.1
RGB　RGB 颜色模型　12.4
RGB-CMY conversion　RGB-CMY 颜色模型转换　12.6.2
RGB-YIQ conversion　RGB-YIQ 颜色模型转换　12.5.2
standard CIE primaries　标准 CIE 基色　12.3
subtractive　减色模型　12.6
tristimulus vision theory　三刺激理论　12.4
XYZ　XYZ 颜色模型　12.3.1
YC_rC_b　YC_rC_b 系统　12.5.3
YIQ　YIQ 颜色模型　12.5
YUV　YUV 颜色模型　12.5.3
Color monitor（see also Video monitor）:　彩色监视器（又见视频监视器）:　2.1
beam-penetration　电子束穿透法　2.1.4
composite　合成式监视器　2.1.4
delta-delta　delta-delta 荫罩法　2.1.4
full-color system　全彩色系统　2.1.4
in-line　按线排列　2.1.4
RGB　RGB 颜色模型　2.1.4
shadow-mask　荫罩监视器　2.1.4
true-color system　真彩色系统　2.1.4
Color-table animation　光栅颜色表动画　13.1.2
Column:　列:
number　列号　3.1.1
vector（matrix）　列向量　A.5
Command icon　命令图符　11.8.2
Commission Internationale de l'Éclairage（CIE）　国际照明委员会（CIE）　12.3
Complementary colors　互补色　12.2.1
Complex number:　复数:
absolute value　绝对值　A.6.3
arithmetic operations　算术运算　A.6.1
conjugate　共轭复数　A.6.3
Euler's formula　欧拉公式　A.6.5
imaginary part　虚部　A.6
modulus　模　A.6.3
ordered-pair representation　有序对表示　A.6
polar representation　极坐标表示　A.6.5
pure imaginary　纯虚数　A.6
real part　实部　A.6
roots　根　A.6.5
vector length　向量长度　A.6.3
Complex plane　复平面　A.6
Composite monitor　合成式监视器　2.1.4
Composite transformations　复合变换　5.4
Compression methods（image files）　压缩算法（图像文件）15.3
Computed axial tomography（CAT）　计算机轴向造影术（CAT）　1.8
Computed tomography（CT）　计算机断层造影术（CT）　1.8
Computer-aided:　计算机辅助:
design（CAD）　设计　1.2
drafting and design（CADD）　绘图和设计　1.2
education and training　教学与培训　1.5
manufacturing（CAM）　制造　1.2
surgery　手术　1.8
Computer animation（see also Animation）　计算机动画（又见动画）　13
Computer art　计算机艺术　1.6, 4.6
Concatenation（matrix）　合并　5.4
Concave polygon　凹多边形　3.15.1
clipping　裁剪　6.8.1
identifying　识别　3.15.2
splitting　分割　3.15.1
Cone:　圆锥:
filter　过滤　4.17.4
receptors　锥状细胞接收器　12.4
tracing（see also Ray tracing）　锥形光线跟踪　10.11.8
of vision（perspective viewing）　视觉棱锥体（透视观察）　7.8.4

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Conic sections 圆锥曲线 3.52
Conjugate（complex number） 共轭（复数） A.6.3
Constant-intensity surface rendering 恒定强度的明暗处理 10.10.1
Constraints（interactive drawing） 约束（交互式构图） 11.4
Constructive solid geometry（CSG） 结构实体几何法（CSG） 8.2
  firing plane 射线平面 8.2
  mass calculations 质量计算 8.2
  octree methods 八叉树 8.21
  ray-casting methods 光线投射 8.2
  volume calculations 体积计算 8.2
Continuity conditions（spline）: 连续性条件（样条）:
  geometric 几何连续性条件 8.8.3
  parametric 参数 8.8.2
Continuity parameter 连续性参数 8.9.4
Continuous-tone images（see also Halftone） 连续色调图像 10.8.3
Contour（intensity border） 轮廓（强度边界） 10.8.3
Contour plots 等值线图 8.27.1
Contraction（tensor） 缩减（张量） 8.27.3
Control: 控制:
  graph 控制图 8.8.1
  grid（CRT） 控制栅极 2.1.1
  icon 图符 11.8.2
  operations 控制操作 2.8.2
  point（spline） 控制点 8.8.1
  polygon 控制多边形 8.8.1
  surface（fractal terrain） 控制面 8.23.8
Convex: 凸多边形:
  angle 角 3.15.4
  hull 凸壳 8.8.1, 8.12.1
  polygon 多边形 3.15.1
  polygon splitting 多边形分割 3.15.4
Coordinate-axis rotations 坐标轴旋转 5.11.1
Coordinate-axis vectors（basis） 坐标轴向量 A.4
Coordinate extents 坐标范围 3.1
Coordinates: 坐标:
  absolute 绝对坐标 3.1.2
  axes 坐标轴 A.1.4
  Cartesian 笛卡儿坐标系 A.1.4
  current position 当前位置 3.1.2
  curvilinear 曲线坐标系 A.1.6
  cylindrical 圆柱坐标系 A.1.6
  device 设备坐标系 2.8.1
  homogeneous 齐次坐标 3.3
  left-handed 左手坐标系 A.15
  local（frame mapping） 本地坐标系 10.19
  local（modeling） 局部坐标系 2.8.1
  master 主坐标系 2.8.1
  modeling 建模坐标系 2.8.1
  normalized 规范化设备坐标系 2.8.1
  normalized projection 规范化投影 7.2, 7.6.4
  orthogonal 正交坐标系 A.1.4
  pick 拾取 11.2.6
  pixel（screen grid） 像素坐标（屏幕网格） 3.1.1
  polar 极坐标 A.1.3
  relative 相对坐标 3.1.2
  right-handed 右手坐标系 2.8.1, A.1.4
  screen 屏幕坐标系 2.8.1, A.1.5
  spherical 球面坐标系 A.1.6
  surface 坐标曲面 A.1.6
  texture 纹理坐标 10.17
  three-dimensional 三维坐标系 A.1.4
  transformation 坐标变换
  two-dimensional 二维坐标系 A.1.1
  uvn uvn观察坐标系 7.3
  viewing 观察坐标系 2.8.1
  world 世界坐标系 2.8.1
Cox-deBoor recursion formulas Cox-deBoor递归公式 8.12.1
Cramer's rule Cramer定律 A.14.1
Graphs and charts（see also Chart） 图和表 1.1
Cross-hatch fill 交叉影线填充 4.9.1
Cross product（vectors） 叉积（向量） A.2.5
CRT（see also Cathode-ray tube） CRT（又见阴极射线管） 2.1.1
CSG methods（see also Constructive solid geometry） CSG方法（又见结构实体几何法） 8.2
CT（Computed Tomography）scan CT（计算机控制X射线断层造影术）扫描 1.8
Cubic spline 三次样条 8.9
  beta beta样条 8.14
  Bézier Bézier样条 8.1
  B-spline B样条 8.13
  interpolation 插值样条 8.9
Curl operator 旋度算子 A.10.6
Current position 当前位置 3.1.2
Current raster position（OpenGL） 当前光栅位置 3.19.1
Curve: 曲线:
  attributes 曲线属性 4.6
  beta spline beta样条 8.14
  Bézier spline Bézier样条 8.1
  B-spline B样条 8.13
  cardinal spline cardinal样条 8.9.3
  cardioid 心形线 3.25
  Catmull-Rom spline Catmull-Rom样条 8.9.3
  circle 圆
  clipping 弧的裁剪 6.9
  conic sections 圆锥曲线 3.11.1

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

ellipse　椭圆　3.1
four-leaf　四叶曲线　3.25
fractal（see also Fractal curves）　分形（又见分形曲线）　8.23
Hermite spline　Hermite 插值　8.9.2
hyperbola　双曲线　3.11.1
Koch（fractal）　Koch 曲线　8.23.4
Kochanek-Bartels spline　Kochanek-Bartels 样条　8.9.2
limaçon　蜗形线　3.25
natural spline　自然三次样条　8.9.1
Overhauser spline　Overhauser 样条　8.9.3
parabola　抛物线　3.11.1
parallel algorithms　并行曲线算法　3.12
parametric representations　参数方程　3.11
piecewise construction　分段构造　3.11.2
polynomials　多项式　3.11.2
rational spline　有理样条　8.15
spiral　三叶曲线　3.25
splines（see also Spline curve）　样条（又见样条曲线）　8.8
straight-line approximations　直线逼近法　3.11
superquadric　超二次曲线　8.5
symmetry considerations　对称性考虑　3.10.1
three-leaf　三叶曲线　3.25
Curved surface:　曲面:
Bézier spline　Bézier 样条曲面　8.1
B-spline　B 样条曲面　8.13
contour plots　等值线图　8.27.1
ellipsoid　椭球面　8.4
explicit representations　显式表示　A.8
implicit representations　隐式表示　A.8
isosurface　等值面　8.27
nonparametric representation　非参数表示　A.8
parametric representations　参数表示　A.9
quadric　二次曲面　8.3
rendering（see Surface rendering）　绘制（又见表面绘制）
sphere　球面　8.4
spline（see also Spline surface）　样条（又见样条曲面）
superquadric　超二次曲面
torus　环面　8.4.3
trimming curve　修剪曲线　8.8.6
visibility(see also Visible-surface detection）　可见性（又见可见面检测）　9.1
weighting　加权　4.17.2
Curvilinear coordinates　曲线坐标　A.1.6
Cutaway views　剖切面视图　7.1.6
Cylindrical coordinates　圆柱坐标　A.16
Cyrus-Beck line clipping　Cyrus-Beck 线裁剪算法　6.7.2

D

Damping constant　衰减常数　13.7.1
Dashed-line generation　线型　4.5.2
Data glove（see also Virtual reality）　数据手套（又见虚拟现实）　2.4.5
Data tablet（see also Digitizer）　数据板（又见数字化仪）　2.4.6
Data visualization:　数据可视化:
applications　应用　1.4
color-coding methods　颜色编码技术　8.27.1
contour plots　等值线图　8.27.1
field lines　场线　8.27.2
glyphs　点符　8.27.4
isolines　等值线　8.27.1
isosurfaces　等值面　8.27.1
multivariate fields　多变量数据场　8.27.4
scalar fields　标量场　8.27.1
streamlines　流线　8.27.2
tensor fields　张量场　8.27.3
vector fields　向量场　8.27.2
volume rendering　体绘制　8.27.1
DDA line algorithm　DDA 画线算法　3.5.2
Deflection（see also Cathode-ray tube）:　偏转（又见阴极射线管）:
coils　线圈　2.1.1
current　电流　2.1.1
plates　极板　2.1.1
Degenerate polygon　退化多边形　3.15.1
Degree parameter（B-spline）　阶参数（B 样条）　8.12.1
Degrees of freedom　自由度　13.5
Del　梯度　A.10.1
Del squared　梯度平方　A.10.2
Delta-delta shadow-mask CRT　delta-delta 荫罩法　2.1.4
Density function（blobby object）　密度函数（又见凹凸对象）　8.7
Dependent variable　自变量　A.8
Depth（frame buffer）　深度（帧缓存）　2.1.2
Depth buffer　深度缓存　3.19.2
Depth-buffer algorithm（visibility detection）　深度缓存算法　9.3
Depth cueing　深度提示　7.1.3
Depth of field　景深　10.11.7
Depth-sorting algorithm　深度排序算法　9.11
Derivative operators:　微分算子:
curl　旋度算子　A.10.6

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

directional 方向算子 A.10.2
divergence 散度算子 A.10.5
gradient 梯度算子 A.10.1
Laplace 拉普拉斯算子 A.10.4
ordinary 常微分算子 A.10
partial 偏导数算子 A.10
Descender（character） 下行字符 4.15
Determinant 矩阵的秩 A.5.4
Device coordinates 设备坐标系 2.8.1
Dial input box 旋钮 2.4.1
Dictionary-based algorithm 基于字典的算法 15.3.2
Differential equations 微分方程 A.14.4
Differential scaling 差值缩放 5.1.3
Diffuse: 漫射:
reflection 漫反射 10.2
reflection coefficient 反射系数 10.3.2
transmission 透射 10.4
Digital differential analyzer（DDA）line algorithm 数字微分分析仪（DDA）方法 3.5.2
Digitizer 数字化仪: 2.4.6
accuracy 精度 2.4.6
acoustic 声学数据板 2.4.6
electromagnetic 电磁脉冲 2.4.6
hand cursor 手持光标 2.4.6
resolution 分辨率 2.4.6
sonic 声音数据板 2.4.6
stylus 触笔 2.4.6
three-dimensional 三维数字化仪 2.4.6
Dimension 维数 A.3
box 立方体 8.23.3
Euclidean 欧氏 8.23
fractal 分形 8.23
fractional 分形维数 8.23
topological 拓扑 8.23.3
Directed line segment（vector） 有向线段 A.2.2
Directional: 方向:
derivative 导数 A.8
light source 方向光源 10.1.4
Direction angles 方向角 A.2.2
Direction cosines 方向余弦 A.2.2
Discrete cosine encoding 离散余弦编码 15.3.6
Display: 显示:
callback functions 显示回调函数 2.9.5
coprocessor 显示协处理器 2.2.2
controller 显示控制器 2.2
devices（see also Video monitors; Display processors） 显示设备（又见视频显示器；显示处理器） 2.1
file 显示文件 2.1.3
list 显示列表 2.1.3
processor 显示处理器 2.2.2
program 显示程序 2.1.3
window 显示窗口 2.9.2
Display-window management 显示窗口管理 2.9.4, 6.1
Distance: 距离:
point to line 点到线距离
ray-tracing path 光线跟踪路径 10.11.1
Distributed ray tracing 分布式光线跟踪 10.11.9
Distribution ray tracing 分布式光线跟踪 10.11.9
Dithering 抖动技术: 10.9.2
dot diffusion method 点分散 10.9.2
error diffusion method 误差分散方法 10.9.2
matrix 抖动矩阵 10.9.2
noise 抖动噪音 10.9.2
ordered-dither method 按序抖动算法 10.9.2
random 随机抖动 10.9.2
Divergence: 散度:
operator 散度算子 A.10.5
theorem 散度定理 A.11.3
Dominant frequency 主频率 12.1.1
Dominant wavelength 主波长 12.1.1
Dot product（vectors） 点积（向量） A.2.4
Dot-matrix printer 点阵打印机 2.5
Dot-diffusion algorithm 点分散方法 10.9.2
Double buffering 双缓存 2.9.4
Double refraction 双重折射 10.4.2
Dragging 拖曳 11.4.2
Drawing methods 绘画方法 11.4.7
Dynamics（see also Animation） 动力学 13.7.7

E

Eccentric angle 离心角 3.10.1
Echo feedback 回显反馈 11.3.2
Edge（polygon） 边（多边形） 3.15
flag（OpenGL） 标志（OpenGL） 4.14.3
stitching effects 缝线 4.14.3
vector 向量 3.15.2
Edge list: 边表:
active 活化边表 4.1
in polygon tables 多边形表 3.15.6
sorted 有序边表 4.1
Education and training applications 教学与培训 1.5
8-connected fill method 8-连通区域 4.13.1
Elastic material（nonrigid object） 弹性材料（非刚体对象） 8.26
Electroluminescent display 光电显示器 2.1.5
Electromagnetic: 电磁:
radiation 电磁辐射 4.2.4, 12.1.1
spectrum 电磁频谱 12.1.1
Electron beam（see also Cathode-ray tube）: 电子束（又见阴极射线管）:

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

convergence　会聚　2.1.1
electrostatic deflection　偏转　2.1.1
focusing　聚焦　2.1.1
intensity　强度　2.1.1
magnetic deflection　磁性偏转　2.1.1
spot size　亮点尺寸　2.1.1
Electron gun（see also Cathode-ray tube）　电子枪（又见阴极射线管）　2.1.1
Electrostatic:　静电:
deflection（CRT）　静电偏转（CRT）　2.1.1
focusing（CRT）　聚焦（CRT）　2.1.1
printer　打印机　2.5
Electrothermal printer　热转印　2.5
Elevation view　立面图　7.6
Ellipse:　椭圆:
eccentric angle　离心角　3.10.1
equations　椭圆方程　3.10.1
focus point　焦点　3.10.1
midpoint algorithm　中点椭圆算法　3.10.2
properties　特征　3.10.1
symmetry　对称性　3.10.1
Ellipsoid　椭球面　8.4
Emissive display（emitter）　发射显示器（发射器）　2.1.5
Energy:　能量:
distribution（light source）　分布（又见光源）　12.1.2
function（cloth-modeling）　函数（又见布料建模）　8.26
photon　光子　10.14
quantum levels　量子能级　2.1.1
radiant　辐射能　10.12.1
Entertainment applications　娱乐应用　1.7
Environment array　环境数组　10.13
Environment mapping　环境映射　10.13
Error-diffusion algorithm　误差分散算法　10.9.2
Euclidean:　欧氏:
dimension　维数　8.23
geometry methods　几何方法　8.23
Euler:　欧拉:
formula（see also Complex numbers）　公式（又见复数）　A.6.5
method（differential-equation solving）　方法（微分方程）　A.14.4
Event:　事件:
input mode　输入模式　11.3
queue　队列　11.3
Explicit functional form　显式函数表示　A.8
Exploded view　拆散显示　7.1.6
Extended light source　扩展光源　10.1.6, 10.11.9

F

Fakespace Pinch gloves　Fakespace Pinch 手套　2.1.7
False-position root finding　伪定位求根法　A.14.2
Far plane（clipping）　远平面（裁剪）　7.6.3
Fast Phong surface rendering　快速 Phong 明暗处理　10.10.4
Field lines　场线　8.27.2
Field-of-view angle　视场角　7.8.6
Filament（CRT）　灯丝（CRT）　2.1.1
File-compression techniques　文件压缩技术　15.3
File-transfer protocol（ftp）　文件传送协议　2.7
Fill area　填充区　3.14
attributes　属性　4.9
curved boundary　曲线边界　4.12
logical operations　逻辑操作　4.9.2
nonzero winding-number rule　非零环绕数规则　3.15.5
odd-even rule　奇偶规则　3.15.5
interior　内部　3.15.5
polygon　多边形　3.15.1
Fill-area algorithms:　填充区算法:
antialiasing　反走样　4.17
color blending　颜色调和　4.9.1
convex polygons　凸多边形　4.11
curved-boundary areas　曲线边界区域　4.12
boundary-fill method　边界填充算法　4.13.1
8-connected method　8- 连通方法　4.13.1
fill patterns　填充模式　4.9.1
flood-fill method　泛滥填充算法　4.13.2
4-connected method　4- 连通方法　4.13.1
hatch fill　影线填充　4.9.1
irregular-boundary areas　不规则边界区域　4.13
parallel methods　并行方法　4.1
polygons　多边形　4.1
scan-line method　扫描线填充算法　4.1
soft fill　软填充　4.9.2
tiling　平铺　4.9.1
tint fill　色彩填充　4.9.2
Fill styles　填充模式　4.9.1
Filter（see also Antialiasing）:　过滤（又见反走样）:
box　矩形过滤器　4.17.4
cone　圆锥过滤器　4.17.4
function　过滤函数　4.17.4
Gaussian　高斯过滤器　4.17.4
Finite differences　有限差　A.14.5
Firing plane（ray casting）　射线平面　8.2
First-order continuity:　一阶连续性:　8.10.4
geometric　几何连续性　8.8.3
parametric　参数连续性　8.8.2
Fixed point（scaling）　固定点（缩放）　5.1.3

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Flaps（light control） 快门（光线控制） 10.1.6
Flat-panel displays 平板显示器: 2.1.5
active-matrix 激活矩阵 2.1.5
emissive 发射显示器 2.1.5
gas-discharge 气体放电显示器 2.1.5
light-emitting diode（LED） 发光二极管（LED） 2.1.5
liquid-crystal（LCD） 液晶显示器（LCD） 2.1.5
nonemissive 非发射显示器 2.1.5
passive-matrix 无源矩阵 2.1.5
plasma 等离子体显示板 2.1.5
thin-film electroluminescent 薄膜光电显示器 2.1.5
Flat surface rendering 平面绘制 10.10.1
Flight simulators 飞行模拟器 1.5
Flood-fill algorithm 泛滥填充算法 4.13.2
f-number *f*数 10.11.7
Focusing system（CRT） 聚焦系统 2.1.1
Focus point（ellipse） 焦点 3.10.1
Follow-through（animation） 运动完结（动画） 13.3
Font（see also Typeface） 字体: 3.2
bitmap 位图字体 3.2
cache 字形缓存 3.2
legible 字迹清楚 3.2
monospace 单一宽度 3.2
outline 轮廓字体 3.2
proportional 比例宽度 3.2
raster 光栅字体 3.2
readable 可读 3.2
sans-serif 无衬线 3.2
serif 有衬线 3.2
stroke 笔画字体 3.2
Force constant 力常数 8.26
Form factors（radiosity） 形状因子 10.12.2
Forward differences 向前差分 8.17.2
4-connected fill method 4-连通填充算法 4.13.2
Fractal: 分形:
affine constructions 仿射分形构造方法 8.23.6
box-covering methods 方框覆盖方法 8.23.3
box dimension 方框维数 8.23.3
Brownian motion 布朗运动 8.23.6
characteristics 分形特性 8.23
classification 分形分类 8.23.2
control surfaces 控制面 8.23.8
dimension 维数 8.23.3
fractional Brownian motion 分形布朗运动 8.23.6
fractional dimension 分形维数 8.23.3
generation procedures 生成过程 8.23.1
generator 生成元 8.23.4
geometric constructions 几何构造 8.23.7
Hausdorff-Besicovitch dimension Hausdorff-Besicovitch 维数 8.23.3
initiator 初始元 8.23.4
invariant set 不变分形集 8.23.2
inversion methods 逆方法 8.23.10
Julia set Julia 集 8.23.9
Koch curve Koch 曲线 8.23.4
Mandelbrot set Mandelbrot 集合 8.23.9
midpoint displacement 随机中点位移方法 8.23.7
Peano curve Peano 曲线 8.23.3
Peano space Peano 空间 8.23.4
quaternion representations 四元数 8.23.9
random midpoint-displacement methods 随机中点位移方法 8.23.7
self-affine 自仿射 8.23.2
self-inverse 自逆 8.23.2
self-similar 自相似 8.23.2
self-similarity characteristic 自相似性特性 8.23
self-squaring 自平方 8.23.2
similarity dimension 相似维数 8.23.3
snowflake 雪花 8.23.4
statistically self-similar 统计自相似 8.23.2
subdivision methods 细分方法 8.23.3
terrain features 地面特征 8.23.7
topological covering methods 拓扑覆盖方法 8.23.3
Fractional Brownian motion 分形布朗运动 8.23.6
Fractional dimension 分形维数 8.23.3
Frame: 帧:
animation 动画 13
rate 刷新速率 13.1.1
screen area 屏幕范围 2.1.2
Frame buffer 帧缓存: 2.1.2
address calculations 地址计算 3.7
bit-block transfers 块移动 5.6
bit planes 位平面 2.1.2
depth 缓存深度 2.1.2
getPixel command getPixel 命令 3.1.1
lookup table 查找表 4.2.2
raster operations 光栅操作 2.1.2
resolution 分辨率 2.1.2
setPixel command setPixel 命令 3.1
Frame mapping 帧映射 10.19
Fresnel reflection laws Fresnel 反射定律 10.3.3
Frequency spectrum（electromagnetic） 频段（电磁频谱） 12.1.1
Front buffer 前缓存 13.1
Front face（polygon） 前向面 3.15.8
Front plane（clipping） 前平面（裁剪） 7.6.3
f-stop *f*刻度 10.11.7
ftp（file-transfer protocol） ftp（文件传送协议） 2.7

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Full-color system　全彩色系统　2.1.4
Frustum　棱台　7.8.4
Function representation:　函数表示:
　explicit　显式　A.8
　implicit　隐式　A.8

G

Gamma correction　gamma 校正　10.8.2
Gamut (color)　颜色范围　12.2
Gas-discharge displays　气体放电显示器　2.1.5
Gauss:　高斯:
　density (bump) function　密度 (凸起) 函数　8.7
　distribution　高斯分布　2.1.1
　elimination　高斯消去法　A.14.1
　filter function　过滤函数　4.17.4
　theorem　高斯定理　A.11.3
Gauss-Seidel method　Gauss-Seidel 法　A.14.1
Generator (fractal)　生成元　8.23.4
Geometric:　几何:
　continuity (spline)　连续性　8.9.4
　model　几何模型　14.1
　primitives (see also Graphics output primitives)　几何图元 (又见图形输出原素)　2.8.2
　production rules　产生式规则　8.24
　representations (image files)　几何表示　15.1
　tables (polygon)　多边形表　3.15.6
　transformations (see also Transformations)　几何变换 (又见变换)　2.8.2
getPixel procedure　getPixel 函数　3.1.1
GIFF　GIFF 文件　15.4.12
GKS (Graphical Kernel System)　GKS (图形核心系统)　2.8.3
GL (Graphics Library)　GL (图形库)　2.8.3
Global lighting effects　全局光照效果　10.3.1, 10.11.3
GLU (OpenGL Utility)　GLU (OpenGL实用库)　2.9.2
　B-spline curve functions　B样条曲线函数　8.18.3
　B-spline surface functions　B样条曲面函数　8.18.4
　display window　显示窗口　2.9.2
　look-at function　观察函数　7.10.1
　pick-window function　拾取窗口　11.2.6
　quadric-surface functions　二次曲面函数　8.6.3
　trimming functions　曲面修剪函数　8.18.5
　two-dimensional clipping-window function　二维裁剪窗口　2.9.5
GLUT (OpenGL Utility Toolkit)　GLUT (OpenGL实用函数工具包)　2.9.2
　character attribute functions　字符属性函数　4.16
　character functions　字符函数　3.21
　cubic-surface function (teapot)　三次曲面 (茶壶) 函数　8.6.2
　current menu　当前菜单　11.7.2
　button-box function　按键盒函数　11.6.5
　dials function　拨号盘函数　11.6.6
　display-window management　显示窗口管理　2.9.4
　display-window reshape function　显示窗口重定形函数　3.24
　idle function　idle 函数　6.4.15
　keyboard functions　键盘函数　11.6.2
　menu-creation function　创建菜单函数　11.7.1
　menu modification functions　修改菜单函数　11.7.4
　mouse functions　鼠标函数　11.6.1
　multiple menu management　管理多个菜单　11.7.2
　quadric-surface functions　二次曲面函数　8.6.1
　query functions　查询函数　6.4.8
　regular-polyhedron functions　规则多面体　8.2
　screen-cursor function　屏幕光标设定函数　6.4.12
　spaceball functions　空间球函数　11.6.4
　submenu creation　创建子菜单　11.7.3
　tablet functions　数据板函数　11.6.3
　teapot function　茶壶函数　8.6.2
Glyph　点符　8.27.4
GLX (OpenGL extension to the X window system)　GLX (OpenGL的X窗口系统扩充)　2.9.2
Goal-directed motion　目标导向系统　13.7.2
Gouraud surface rendering　Gouraud表面绘制　10.10.2
Gradient (grad) operator　梯度算子　A.10
Grad squared　梯度平方　A.10.4
Graftal　嫁接　8.24
Graph:　图:
　bar　直方图　1.1
　line　折线图　1.1
　pie　饼图　1.1
　three dimensional　三维图　1.1
　time　时间表　1.1
Graphical input data　图形数据的输入　11.1
Graphical Kernel System (GKS)　图形核心系统 (GKS)　2.8.3
Graphical user interface:　图形用户界面:
　accommodating multiple skill levels　适应多种熟练程度　11.8.3
　backup and error handling　回退和出错处理　11.8.6
　components　成分　11.8.1
　consistency　一致性　11.8.4
　designing　设计　11.8
　feedback　反馈　11.8.7
　help facilities　帮助功能　11.8.3
　icons　图符　11.8.2
　menus　菜单　11.8.2
　minimizing memorization　减少记忆量　11.85
　user dialogue　用户对话　11.8.1
　user's model　用户模型　11.8.1

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

windows 窗口 11.8.2
Graph plotting（see also Charts） 绘图（又见图表） 1.1
Graphics applications: 图形学应用:
advertising 广告 1.2
agriculture 农业 1.3
animations 动画 1.2
architecture 建筑 1.2
art 艺术 1.6
astronomy 天文 1.4
business 商务 1.1
CAD 计算机辅助设计 1.2
CAM 计算机辅助制造 1.2
drafting and design 绘图和设计 1.2
education 教学 1.5
engineering 工程 1.2
entertainment 娱乐 1.7
facility planning 设备布局 1.2
flight simulators 飞行模拟器 1.5
geology 地质学 1.8
graphical user interfaces 图形用户界面 1.9
graphs and charts 图和表 1.1
image processing 图像处理 1.8
manufacturing 制造 1.2
mathematics 数学 1.6
medicine 医学 1.8
modeling 建模 1.2
physical sciences 物理学 1.5
publishing 出版 1.6
scientific visualization 科学计算可视化 1.4
simulations 模拟 1.4
simulators 模拟器 1.5
training 培训 1.5
virtual reality 虚拟现实 1.3
Graphics controller 图形控制器 2.2
Graphics file formats: 图形文件格式:
Adobe Photoshop Adobe Photoshop 格式 15.4.6
big endian 高位在前 15.1
BMP BMP 格式 15.4.9
arithmetic encoding 算术编码 15.3.5
CGM 计算机图形元文件 15.4.2
configurations 文件结构 15.1
dictionary algorithm 基于字典的算法 15.3.2
discrete cosine encoding 离散余弦编码 15.3.6
file-compression techniques 文件压缩技术 15.3
fractal methods 分形方法 15.3.3
geometric representation 几何表示 15.1
GIF GIF 文件 15.4.12
header 文件头 15.1
Huffman encoding Huffman 编码 15.3.4
hybrid format 混合格式 15.1
inverse cosine transformation 逆余弦变换 15.3.6
JFIF JPEG 文件交换格式 15.4.1
JPEG JPEG 格式 15.4.1
little endian 低位在前 15.1
lossless compression 无损压缩 15.3
lossy compression 有损压缩 15.3
LZW encoding LZW 编码 15.3.2
MacPaint MacPaint 绘图格式 15.4.7
median-cut color reduction 中值切割降色法 15.2.3
metafile 元文件格式 15.1
PCX PCX 格式 15.4.10
PICT PICT 格式 15.4.8
PNG PNG 格式 15.4.4
popularity color reduction 基于出现频率的降色法 15.2.2
raster file 光栅文件 15
raw data 原始数据 15.1
raw raster file 原始光栅文件 15.1
run-length encoding 行程编码 15.3.1
SPIFF 静止图像交换文件格式 15.4.1
substitutional algorithm3 替代算法 15.3.2
Targa TGA 格式 15.4.11
TGA Targa 格式 15.4.11
TIFF TIFF 格式 15.4.3
uniform color reduction 均匀降色法 15.2.1
XBM XBM 格式 15.4.5
XPM XPM 格式 15.4.5
Graphics monitors（see also Video monitors） 图形监视器（又见视频显示器） 2.1
Graphics networks 图形网络 2.6
Graphics object 图形对象 3.14
Graphics output primitives 图形输出元素 2.8.2
bitmap 位图 2.1.2
circle 圆 3.9.1
character 字符 3.2
conic section 圆锥曲线 3.52
ellipse 椭圆 3.10.1
fill area 填充区 3.14
marker 标记 3.2
pixel arrays 像素阵列 3.18
pixmap 像素图 3.18
point 点 3.3
polyline 折线 3.4
polymarker 多点标记 3.2
polynomial 多项式 3.11.2
spline 样条曲线 3.11.2
straight-line segment 直线段 3.5
text 文字 3.2
Graphics server 图形服务器 2.6

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Graphics software (see also OpenGL): 图形软件（又见 OpenGL）:
basic functions 基本函数 2.8
control operations 控制操作 2.8.2
coordinate representations 坐标表示 2.8.1
CG API 计算机图形应用编程接口 2.8
GKS 图形核心系统 2.8.3
GL GL（图形库） 2.8.3
language binding 语言绑定 2.8.3
Open Inventor Open Inventor 软件包 2.8.4
PHIGS 程序员级的分层结构交互图形系统 2.8.3
PHIGS+ PHIGS+ 2.8.3
Renderman Renderman 接口 2.8.4
standards 标准
three-dimensional 三维图形软件 2.8.3
two-dimensional 二维图形软件 2.8.3
viewing pipeline 观察流水线 2.8.1
VRML 虚拟现实建模语言 2.8.4
Graphics tablet (see also Digitizer) 图形数据板（又见数字化仪） 2.4.6
Graphics workstation 图形工作站 2.3
Gravitational acceleration 引力加速 3.11.1
Gravity field 引力场 11.4.6
Gray scale 灰度 4.2.3
Green's: 格林:
identities 格林恒等式 A.11.4
plane theorem 格林平面定理 A.11.2
theorem in space 格林空间定理 A.11.3
transformation equations 格林变换方程 A.11.4
Grid: 网格:
character 字符网格 3.2
in interactive picture constructions 交互式构图 11.4

H

Halftone: 半色调:
approximations 半色调近似 10.9.1
color methods 颜色方法 10.9.1
dithering 抖动技术 10.9.2
patterns 半色调模式 10.9
Halfway vector 半角向量 10.3.3
Hand cursor(Digitizer) 手持光标（数字化仪） 2.4.6
Hard-copy devices 硬拷贝输出 2.5
Hatch fill 影线 4.9.1
Hausdorff-Besicovitch dimension Hausdorff-Besicovitch 维数 8.23.3
Header files for OpenGL OpenGL 头文件 2.9.3
Headers (image files) 文件头（图像文件） 15.1
Head-mounted display (see also Virtual reality) 头盔显示器（又见虚拟现实） 2.1.7
Hemicube (radiosity) 半立方体（辐射度） 10.12.2
Hemisphere (radiosity) 半球体（辐射度） 10.12.2
Hermite spline Hermite 样条 8.9.2
Hertz 赫兹 2.1.2
Hexcone (HSV) 六棱锥（HSV） 12.7.1
Hierarchical modeling 层次建模 14.3
High-definition video monitor 高清晰度系统 2.1.1
Highlighting (pick method) 醒目显示 11.2.6
HLS color model HLS 颜色模型 12.8
Homogeneous coordinates 齐次坐标 3.3
Hooke's law 胡克定律 8.26
Horizontal retrace 水平回扫 2.1.2
Horner's polynomial factoring method Horner 多项式因子规则 8.17.1
h parameter 齐次参数 h 3.3
HSB color model (see HSV) HSB 颜色模型（又见 HSV） 12.7
HSV color model HSV 颜色模型 12.7
Hue 色彩 12.1.1
Huffman encoding Huffman 编码 15.3.4
Hybrid formats 混合格式 15.1
Hyperbola 双曲线 3.11.1
Hypertext Markup Language(HTML) 超文本标记语言（HTML） 2.7

I

IBM OS/2 interface to OpenGL (PGL) IBM OS/2 与 OpenGL 的接口（PGL） 2.9.2
Icon 图符 1.9
Ideal diffuse reflector 理想漫反射体 10.3.2
Identity matrix 单位矩阵 A.5.5
Illuminant C 亮白光 C 12.3.3
Illumination models 光照明模型 10
ambient (background) light 环境光（背景光） 10.3.1
ambient-reflection coefficient 环境光反射系数 10.3.2
angle of incidence 入射角 10.3.2
angle of refraction 折射率 10.4.2
angular intensity attenuation 角强度衰减 11.1.5
atmospheric effects 雾气效果 10.5
background light 背景光 10.4.3
basic model 基本模型 10.4.3
camera parameters 照相机参数 10.7
color considerations 颜色考虑 10.3.8
combined diffuse-specular reflections 漫反射和镜面反射的合并 10.3.4
diffuse reflection 漫反射 10.2
diffuse-reflection coefficient 漫反射系数 10.3.2
diffuse transmission 半透明性 10.4.1
Fresnel's reflection laws Fresnel 反射定律 10.3.3

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

halfway vector 半角向量 10.3.3
ideal diffuse reflector 理想漫反射体 10.3.2
index of refraction 折射率 10.4.2
Lambertian reflector 朗伯反射体 10.3.2
Lambert's cosine law 朗伯余弦定律 10.3.2
light sources 光源 10.1
light-source vector 光源向量 10.1.4
luminance 亮度 10.3.10
multiple light sources 多光源 10.3.5
opacity factor 不透明因子 10.4.3
opaque material 不透明材质 10.3.3
Phong model Phong 模型 10.3.3
radial intensity attenuation OpenGL光源的光线强度衰减系数 10.20.4
radiosity 辐射度 10.12
reflection coefficients 反射系数 10.3.2
refraction effects 折射 10.4.2
RGB lighting components 折射效果 10.4.2
shadows 阴影 10.6
Snell's law Snell 定律 10.4.2
specular reflection 镜面反射 10.2
specular-reflection angle 镜面反射角 10.3.3
specular-reflection coefficient 镜面反射系数 10.3.3
specular-reflection exponent 镜面反射指数 10.3.3
specular-reflection vector 镜面反射向量 10.3.3
spotlights 投射光源 10.3
surface emissions 表面光发射 10.3.6
surface lighting effects 表面光照效果 10.2
translucent material 半透明材料 10.4.1
transmission vector 透射向量 10.4.2
transparency 透明性 10.4
transparency coefficient 透明系数 10.4.3
viewer vector 视点向量 10.3.3
Warn light source Warn 光源 10.1.6
Image file（see also Graphics file formats） 图像文件（又见图像文件格式） 15.1
color-reduction methods 降色方法 15.2
compression methods 压缩方法 15.3
configurations 结构 15.1
major formats 主流文件格式 15.4
Image-order scanning（texture mapping） 图像次序扫描（纹理映射） 10.17.2
Image processing 图像处理 1.8
Image scanners 图像扫描仪 2.4.7
Image-space methods（visibility detection） 像空间算法（隐藏线面检测） 9.1
Imaginary number 虚部 A.6
Imaginary unit 虚单位 A.6.2
Impact printer 击打式打印机 2.5
Implicit functional form 隐式函数表达式 A.8
In-betweens（animation frames） 插值帧（动画帧） 13.2
Independent variable 自变量 A.8
Index of refraction 折射率 10.4.2
Initial-value problem 初值问题 A.14.4
Initiator（fractal） 初始元 8.23.4
Ink-jet printer 喷墨打印机 2.5
In-line shadow-mask CRT 按线荫罩 CRT 2.1.4
Inner product（vector） 内积 A.2.4
Input devices: 输入设备:
button box 按钮盒 2.4.1
choice device 选择设备 11.2.5
data glove 数据手套 2.4.5
dials 旋钮 2.4.1
digitizer 数字化仪 2.4.6
graphics tablet 图形数据板 2.4.6
joystick 操纵杆 2.4.4
keyboard 键盘 2.4.1
light pen 光笔 2.4.9
locator device 定值设备 11.2.4
logical classification 逻辑分类 11.2
mouse 鼠标 2.4.2
pick device 拾取设备 11.2.6
scanner 扫描仪 2.4.7
spaceball 空间球 2.4.3
string device 字符串设备 11.2.3
stroke device 笔划设备 11.2.2
switches 开关 2.4.1
three-dimensional sonic digitizers 三维声学数字化仪 2.4.6
touch panel 触摸板 2.4.8
trackball 跟踪球 2.4.3
valuator device 定值设备 11.2.4
voice systems 语音系统 2.4
Z mouse Z 鼠标 2.4.2
Input functions 输入函数 2.8.2
callback 回调 2.9
echo feedback 回显反馈 11.3.2
Input modes 输入模式 11.3
Inside-outside test: 内－外测试:
polygon odd-even rule 多边形奇偶规则 3.15.4
polygon nonzero winding-number rule 多边形非零环绕数规则 3.15.5
spatial planes 空间平面 3.15.7
Inside surface 内含表面 9.8
Instance（see also Modeling） 实例（又见建模） 14.1.1
Integral equation solving: 积分方程求解:
Monte Carlo methods Monte Carlo 积分方法 A.14.3
rectangle approximations 矩形逼近 A.14.3

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Simpson's rule　Simpson 定律　A.14.3
trapezoid rule　梯形定律　A.14.3
Intensity（see also Illumination models; Radiosity）　强度（又见光照模型，辐射度）　10.12.1
angular attenuation　角强度衰减　10.1.5
array（environment mapping）　强度数组（环境映射）　10.13
electron beam（CRT）　电子束　2.1.1
gamma correction　gamma 校正　10.8.2
interpolation（Gouraud rendering）　强度插值（Gouraud明暗处理）　10.10.2
levels（system）　强度等级　10.8
radial attenuation　辐射衰减　10.1.3
Interactive input methods　交互式输入方法　11
Interactive picture-construction techniques　交互式构图技术　11.4
constraints　约束　11.4.3
dragging　拖曳　11.4.2
gravity field　引力场　11.4.6
grids　网格　11.4.4
painting and drawing　绘画　11.4.7
positioning　定位　11.4.1
rubber-band methods　橡皮条方法　11.4.5
Interior angle（polygon）　内角　3.15.1
Interlacing scan lines　隔行扫描线　2.1.2
International Commission on Illumination（CIE）　国际照明委员会（CIE）　12.3
International Standards Organization（ISO）　国际标准化组织（ISO）　2.8.3
Internet graphics:　因特网上的图形学:
file-transfer protocol（ftp）　文件传送协议（ftp）　2.7
ftp site　ftp 场所　2.7
HTML（hypertext markup language）　HTML（超文本标记语言）　2.7
http（hypertext transfer protocol）　http（超文本传送协议）　2.7
Mosaic browser　Mosaic 浏览器　2.7
Netscape Navigator　Netscape 浏览器　2.7
URL（uniform resource locator）　URL（统一资源定位器）　2.7
TCP/IP　TCP/IP　2.7
World Wide Web　万维网　2.7
Interpolation spline　插值样条曲线　8.8.1
Invariant fractal set　不变分形集　8.23.2
Inverse:　逆:
cosine transformation　余弦变换　15.3.6
geometric transformations（see also individual transformations）　几何变换（又见单一变换）　5
dynamics　动力学　13.7.3
kinematics　运动学　13.7.3
matrix　矩阵　A.5
quaternion　四元数　A.7
scanning（texture mapping）　纹理扫描　10.17.2
ISO（International Standards Organization）　ISO（国际标准化组织）　2.8.3, 15.4.1
Isolines　等值线　8.27.1
Isometric joystick　感应式操纵杆　2.4.4
isometric projection　轴测正投影　7.6.1
Isosurfaces　等值面　8.27.1

J

Jaggies（see also Antialiasing）　阶梯效果（又见反走样）　3.5
JFIF　JPEG 文件交换格式　15.4.1
Jittering　抖动　10.11.9
Joystick:　操纵杆:
isometric　等轴操纵杆　2.4.4
movable　可移动操纵杆　2.4.4
pressure sensitive（isometric）　压力感应式操纵杆（等容积）　2.4.4
JPEG　JPEG　15.4.1
Julia set　Julia 集合　8.23.9

K

Kerned characters　有核字符　4.15
Keyboard　键盘　2.4.1
Key frame（animation）　关键帧（动画）　13.2
Key-frame system　关键帧系统　13.6
Kinematics（see also Animation）　运动学（又见动画）　13.7.3
Knot vector　节点向量　8.12.1
Kochanek-Bartels spline　Kochanek-Bartels样条　8.9.4
Koch curve（fractal snowflake）　Koch 曲线（分形雪花形状）　8.23.4

L

Lambertian reflector　朗伯反射体　10.3.2
Lambert's cosine law　朗伯余弦定律　10.3.2
Language binding　语言绑定　2.8.3
Laplacian operator　拉普拉斯算子　A.10.4
Large-screen systems　大屏幕系统　2.3
LCD（liquid-crystal display）　LCD（液晶显示器）　2.1.5
Least-squares data fitting　最小二乘法　A.14.6
LED（light-emitting diode）　LED（发光二极管）　2.1.5
LED touch panel　LED 触摸板　2.4.8
Left-handed Cartesian coordinates　左手笛卡儿坐标系　A.1.5
Legible typeface　字迹清楚　3.2

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

L-grammar L语法 8.24
Liang-Barsky clipping: 梁友栋-Barskey算法:
polygons 多边形裁剪 6.8.4
two-dimensional lines 二维线裁剪 6.7
Light（see also Color; Illumination models; Surface rendering） 光（又见光照模型；面绘制）
ambient（background） 环境光（背景光） 10.12.3
angle of incidence 入射角 10.3.2
brightness 亮度 12.1.2
chromaticity 色度 12.1.2
chromaticity diagram 色度图 12.3.3
electromagnetic spectrum 电磁频谱 12.1.1
frequency 频率 12.1.1
frequency band 频率范围 12.1.1
hue 色彩 12.1.1
illuminant C 亮白光C 12.3.3
luminance 亮度 10.3.10
period 周期 12.1.1
properties 特性 12.1
purity 纯度 12.1.2
spectral color 光谱颜色 12.1.1
spectrum 频谱 12.1.1
speed 光速 12.1.1
wavelength 波长 12.1.1
white 白色光 12.1.1
Light buffer（ray tracing） 光线缓存 10.11.6
Light-emitting diode（LED） 发光二极管（LED） 2.1.5
Lighting model（see also Illumination model） 光照明模型（又见光照模型） 10
Light pen 光笔 2.4.9
Light polarizer（LCD） 光偏振器（LCD） 2.1.5
Light source 光源: 10.1
angular intensity attenuation 角强度衰减 10.1.5
barn-door parameters 挡光板参数 10.1.6
brightness 亮度 12.1.2
candle 烛光 12.3.1
directional 方向光源 10.1.4
direction vector 方向向量 10.1.4
dominant frequency 主频率 12.1.1
dominant wavelength 主波长 12.1.1
energy distribution 能量分布 12.1.2
extended 扩展光源 10.1.6, 10.11.9
flap parameters 快门参数 10.1.6
infinitely distant 无穷远光源 10.1.2
intensity 强度 10.12.1
luminance 亮度 10.3.10
point 点光源 10.1.1
radial intensity attenuation 辐射强度衰减 10.1.3
radiant energy 辐射能 10.12.1
radiant flux 光通量 10.12.1
radiant power 光能 10.12.1
radiosity 辐射度 10.12.1
spectral radiance 光谱辐射能 10.12.1
spotlight 投射光源 10.14
Warn model Warn模型 10.1.6
Limaçon 蜗形线 3.25
Line: 直线:
antialiasing 反走样 4.17
attributes 属性 4.5
caps 线帽 4.5
equations 直线方程 3.5.1
intensity variations 直线亮度差 4.17.6
parametric representation 参数表示 6.7
pen and brush options 画笔或画刷的选择 4.5.3
slope-intercept equation 斜率截距方程 3.5.1
Linear congruential generator 线性同余发生器 A.14.3
Linear equation solving: 求解常微分方程:
Cramer's rule Cramer定律 A.14.1
Gaussian elimination 高斯消去法 A.14.1
Gauss-Seidel Gauss-Seidel法 A.14.1
LU decomposition LU分解法 A.14.1
Linear soft-fill algorithm 线性软填充算法 4.9.2
Line chart 折线图 1.1
Line clipping: 线段裁剪算法:
Cohen-Sutherland Cohen-Sutherland线段裁剪算法 6.7.1
Cyrus-Beck Cyrus-Beck线段裁剪算法 6.8.3
Liang-Barsky 梁友栋-Barsky线段裁剪算法 6.7.2
Nichol-Lee-Nichol Nichol-Lee-Nichol线段裁剪算法 6.7.3
nonlinear clip window 非线性裁剪窗口 6.2
nonrectangular clip window 非矩形裁剪窗口 6.7.4
three-dimensional 三维裁剪 7.11
two-dimensional 二维裁剪 6.6～6.9
Line contours 等值线 8.27.1
Line-drawing algorithms 画线算法: 3.5
Bresenham Bresenham画线算法 3.5.3
DDA DDA画线算法 3.5.2
parallel methods 并行画线算法 3.6
polylines 折线 3.5.4
setting frame-buffer values 帧缓存值的装载 3.7
Line sampling 线段取样 3.5.1
Little endian 低位在前 15.1
Liquid-crystal display（LCD） 液晶显示器（LCD） 2.1.5
Local coordinates 局部坐标系 2.8.1
Local control（spline） 局部控制（样条） 8.12
Locator input device 定位设备 11.2.1
Logical input devices 逻辑输入设备 11.2
Logical operations 逻辑操作 4.9.2

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Look-at point　注视点　7.3.1
Lookup table　查找表　10.8.2
Lossless compression　无损压缩　15.3
Lossy compression　有损压缩　15.3
LU decomposition　LU 分解法　A.14.1
Luminance　亮度　10.3.10
LZW encoding　LZW 编码　15.3.2

M

Mach bands　马赫带　10.10.2
MacPaint format　MacPaint 绘图格式　15.4.7
Magnetic deflection（CRT）　磁性偏转（CRT）　2.1.1
Mandelbrot set　Mandelbrot 集合　8.23.9
Marching cubes algorithm（see Isosurfaces）　行进立方体算法（又见等值面）　8.27.1
Marker（character）　标记（字符）　3.2
Marker attributes　标记属性　4.15
Mask　掩模　3.18
Mass calculations（CSG）　质量计算（CSG）　8.2
Master coordinates　主坐标系　2.8.1
Matrix　矩阵　A.5
　addition　矩阵加法　A.5.1
　basis（spline）　基本矩阵（样条）　8.8.4
　Bézier　Bézier 矩阵　8.10.5
　B-spline　B 样条矩阵　8.12.3
　cardinal　8.9.3
　column　列向量　A.5
　concatenation　合并　5.4.7
　determinant　矩阵的秩　A.5.4
　dither　抖动　10.9.2
　Hermite　Hermite 矩阵　8.9.2
　identity　单位矩阵　A.5.5
　inverse　矩阵的逆　A.5.5
　multiplication　乘法　A.5.1
　nonsingular　非奇异矩阵　A.5.5
　reflection　反射矩阵　5.5.1
　row　行向量　A.5
　rotation　旋转矩阵　5.2.3
　scalar multiplication　标量乘法　A.5.1
　scaling　缩放矩阵　5.2.4
　shear　错切矩阵　5.5.2
　singular　奇异矩阵　A.5.5
　spline characterization（basis）　特征（基）　8.8.4
　square　方阵　A.5
　translation　平移矩阵　5.2.2
　transpose　矩阵转置　A.5.3
Median-cut color reduction　中值切割降色法　15.2.3
Medical applications　医学应用　1.8
Menus　菜单　11.7
Mesh（polygon）　面片（多边形）　8.1
Meta-ball model　元球　8.7
Metafile　元文件　15.1
Metric tensor　度量张量　A.4
Midpoint algorithms:　中点算法:
　circle generation　圆生成算法　3.9
　ellipse generation　椭圆生成算法　3.1
Midpoint-displacement（fractal）　中点位移方法（分形）　8.23.7
Mip map（texture reduction pattern）　MIP 图（纹理缩减图）　10.17.4
Miter polyline join　斜角连接　4.5.1
Modeling:　建模:
　basic concepts　基本建模概念　14.1
　cloth　布料　8.26
　coordinates　建模坐标　2.8.1
　geometric　几何模型　14.1
　hierarchical　层次式　5
　instance　实例　14.1.1
　local coordinates　局部坐标　14.3.1
　master coordinates　主坐标　14.3.1
　modules　模块　14.1.2
　packages　建模软件包　14.2
　physically based　基于物理的建模　8.26
　representations　表示　14.1.1
　symbols　符号　14.1.1
　symbol hierarchies　符号层次　14.1.2
　transformations　建模变换　2.8
Modes（input）　输入模式　11.3
Module　符号层次　14.2
Modulus（complex）　模（复数）　A.6.3
Monte Carlo methods　Monte Carlo 法　A.14.3
Monitor（see also Video monitor）　监视器　2.1
Monitor response curve　监视器响应曲线　10.8.2
Monospace font　单一宽度　3.2
Morphing　变形技术　1.6
Motion blur　动感模糊　10.11.9
Motion specification　运动描述　13.7
Mouse　鼠标　2.4.2
Multi-panel displays　多板显示系统　2.3
Multivariate-data visualization　多变量数据场的可视表示　8.27.4
Multum in parvo（mip map）　MIP 图　10.17.4

N

Nabla　梯度算子　A.10.1
Nabla squared　梯度平方　A.10.4
National Television System Committee（NTSC）　美国国家电视系统委员会（NTSC）　10.8.2
Natural cubic spline　自然三次样条　8.9.1

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

NCSA(National Center for Supercomputing Applications) NCSA（美国超级计算应用国家中心） 2.7
NCSA CAVE NCSA CAVE 11
Near plane（clipping） 近平面 7.7.4
Nematic liquid-crystal compounds 线状液晶 2.1.5
Newton-Raphson root-finding Newton-Raphson 求根算法 A.14.2
Newton's second law of motion 牛顿第二定律 13.7.3
Nicholl-Lee-Nicholl line clipping Nicholl-Lee-Nicholl 线段裁剪算法 6.7.3
Noise（dither） 噪音（抖动） 10.9.2
Nonemissive display（nonemitter） 非发射显示器（非发射器） 2.1.5
Nonimpact printers 非击打式打印机 2.5
Nonlinear clip windows 非线性裁剪窗口 6.7.5
Nonlinear-equation solving 非线性方程求根算法 A.14.2
Nonparametric representations 非参数表示 A.8
Nonrectangular clip windows 非矩形裁剪窗口 6.7.4
Nonrigid object 非刚性对象 8.26
Nonsingular matrix 非奇异矩阵 A.5.5
Nonuniform B-splines 非均匀 B 样条曲线 8.12.5
Nonuniform rational B-spline（NURB） 非均匀有理B样条（NURB） 8.15
Nonuniform scaling 非均匀缩放 5.1.3
Nonzero winding number rule 非零环绕数规则 3.15.5
Normal basis 正交基 A.4.2
Normalization transformation: 规范化变换:
orthogonal projection 正投影 7.6
perspective projection 透视投影 7.8
two-dimensional 二维窗口 6.2.2
Normalized: 规范化:
coordinates 规范坐标 6.3.1
projection coordinates 投影坐标 7.2
square 正方形 6.3.2
viewport 视口 6.3.1
view volumes（see also Clipping） 观察体（又见裁剪） 7.2
Normal vector: 法向量:
average（polygon mesh） 平均法向量 10.10.2
curved surface 曲面 10.1
interpolation（Phong surface rendering） 插值（Phong表面绘制） 10.10.3
plane surface 平面 3.15.8
view plane 观察平面 7.3.1
NTSC（National Television System Committee） NTSC（美国国家电视系统委员会） 10.8.2
Numerical methods: 数值方法:
bisection method 二分法
boundary-value problems 边界值问题 A.14.2
Cramer's rule Cramer 定律 A.14.4
Euler method 欧拉方法 A.14.6
false-position method 伪定位法 A.14.4
finite differences 有限差 A.14.2
Gaussian elimination 高斯消去法 A.14.5
Gauss-Seidel method Gauss-Seidel 法 A.14.1
integral evaluations 积分估值 A.14.1
initial-value problems 初值问题 A.14.3
least-squares data fitting 数据集的最小二乘曲线拟合 A.14.4
linear-equation sets 线性方程组 A.14.6
LU decomposition LU 分解法 A.14
Monte Carlo methods Monte Carlo 积分技术 A.14.1
Newton-Raphson method Newton-Raphson 算法 A.14.3
nonlinear equations 非线性方程组 A.14.2
ordinary differential equations 常微分方程 A.14.2
partial differential equations 偏微分方程 A.14.4
random-number generator 随机数生成器 A.14.5
root finding 求根 A.14.3
Runge-Kutta algorithm Runge-Kutta算法 A.14.2
Simpson's rule Simpson 定律 A.14.4
trapezoid rule 梯形定律 A.14.3
NURB（Nonuniform rational B-spline） NURB（非均匀有理B样条） 8.15
Nyquist:
sampling frequency 取样频率 4.17
sampling interval 取样间隔 4.17

O

Object: 对象:
nonrigid（flexible） 非刚性（柔性） 8.26
picture component 图形子部分 3.22
rigid 刚性 13.7.3
Object geometry 对象的几何要素 3.13
Object representations: 对象表示:
beta splines beta 样条 8.14
Bézier splines Bézier 样条 8.1
boundary（B-rep） 边界表示 8
B-splines B 样条 8.12
BSP trees BSP 树 8.22
blobby surfaces 柔性对象 8.7
CSG methods 结构实体几何法 8.2
cubic spline interpolation 三次样条插值 8.9
data visualization 数据集可视化 8.27
density functions 密度函数 8.7
ellipsoid 椭球面 8.4.2
fractal curves and surfaces 分形曲线和曲面 8.23
Gaussian bumps 高斯凸起 8.7

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

constant-intensity surface rendering 常数强度表面绘制 10.20.11
copying texture patterns 复制纹理图案 10.21.7
core library 核心库 2.9.1
cubic-surface function（teapot） 三次曲面茶壶函数 8.6.2
current matrix 当前矩阵 5.17.2
current raster position 当前光栅位置 3.19.1
current texture state 当前纹理状态 10.21.9
curve functions 曲线函数 3.8
data types 数据类型 2.9.1
default refresh buffer 默认刷新缓存 3.19.2
default rendering mode 填充模式 4.9.1
depth buffer 深度缓存 3.19.2
depth-buffer functions 深度缓存函数 9.14.2
depth-cueing function 深度提示函数 9.14.4
destination color 目标颜色 4.3.3
directional light sources（spotlights） 方向光源（投射光源） 10.20.5
display callback functions 显示回调函数 2.9.5
display lists 显示列表 3.23
display-window management（see also GLUT） 显示窗口管理（又见GLUT） 2.9.4
double buffering 双缓存 3.19.2
edge flags 边界标志 4.14.3
far clipping plane 远裁剪平面 7.10.2
feedback mode 反馈模式 11.6.7
fill-area attribute functions 填充区属性函数 4.14
fill-area functions 填充区函数 3.16
flat surface rendering 平表面绘制 10.20.11
front buffer 前缓存 3.19.2
front-face function 前向面函数 4.14.4
front-left color buffer 前－左颜色缓存 3.19.2
general perspective-projection function 通用透视投影函数 7.10.4
geometric transformation functions 几何变换函数 5.17
global lighting parameters 全局光照参数 10.20.6
GLX（extension to X Windows） OpenGL的X窗口系统扩充 2.9.2
Gouraud surface rendering Gouraud表面绘制 10.20.11
halftoning operations 半色调操作 10.20.12
header files 头文件 2.9.3
hierarchical modeling 层次建模 14.4
homogeneous coordinates 齐次坐标 3.3
homogeneous texture coordinates 齐次纹理坐标 10.21.15
IBM/OS interface（PGL） IBM/OS 计算机的OpenGL接口 2.9.2
illumination functions 光照函数 10.2
illumination model 光照模型 10.20.8
illumination parameters 光照参数 10.2
image-compositing functions 图像混合函数 4.3.3
imaging subset 成像子集 4.3.3
infinitely distant light source 无穷远光源 10.20.2
interactive input functions（see also GLUT） 支持交互式输入设备的函数（又见GLUT） 11.6
interlaced array 交错数组 4.3.4
interpolation-fill methods 插值填充法 4.14.2
introductory sample program 介绍性示例程序 2.1
left buffer 左缓存 3.19.2
light-source colors 光源颜色 10.20.3
light-source properties 光源特性 10.20.1
light-source types 光源类型 10.20.2
linear-texture functions 线纹理函数 10.21.1
line attribute functions 线属性函数 4.8
line functions 画线函数 3.4
local light source 近光源 10.20.2
matrix stacks 矩阵栈 5.17.3
menu functions（see also GLUT） 菜单函数（又见GLUT） 11.7
mip maps 纹理缩减图案 10.21.11
modelview matrix 建模观察矩阵 5.17.2
name stack 名称栈 11.6.7
name-stack functions 名称栈函数 11.6.7
naming texture patterns 纹理图案命名 10.21.9
near clipping plane 近裁剪平面 7.10.2
normal-vector interpolation lists 法向量插值列表 10.20.11
one-dimensional（linear）texture space 一维纹理空间 10.21.1
optional clipping planes 任选裁剪平面 7.12
orthogonal-projection transformations 正投影变换 7.10.2
orthogonal-projection function 正投影 2.9.5
perspective-projection functions 透视投影函数 7.10.4
PGL（IBM OS/2 Presentation Manager interface） IBM OS/2 计算机的 OpenGL 接口 2.9.2
pick buffer 拾取缓冲区 11.6.7
pick-buffer array function 拾取缓冲区数组函数 11.6.7
pick identifiers 拾取标识符 11.6.7
picking operations（see also GLU） 拾取操作（又见GLU） 11.6.7
pick window 拾取窗口 11.6.7
pixel-array functions 像素阵列函数 3.19
pixmap function 像素图函数 3.19.2
point attribute functions 点属性函数 4.7

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

point functions　画点函数　3.3
point light-source functions　点光源函数　10.20.1
polygon-culling functions(back-face removal)　多边形剔除函数　9.14.1
polygon edge flags　多边形边界标志　4.14.3
polygon fill-area attribute functions　多边形填充区属性函数　4.14
polygon fill-area functions　多边形填充区函数　3.16
polyhedron functions　多面体函数　8.2
primitives　基本元素　2.9.5
projection mode　投影模式　2.9.5
proxy textures　代理纹理　10.21.13
quadric-surface functions(see also GLUT; GLU)　二次曲面函数(又见GLUT; GLU)　8.6.1
query functions　查询函数　4.19
radial-intensity attenuation coefficients　光线强度衰减系数　10.20.4
raster operations　光栅操作　3.19.3
raster transformations　光栅变换　5.7
reduction texture patterns　纹理缩减图案　10.21.11
refresh buffer　刷新缓存　2.9.5
regular polyhedron functions　规则多面体函数　8.2.2
related libraries　相关库　2.9.2
rendering modes　绘制模式　10.20.11
RGB and RGBA modes　RGB 模式与 RGBA 模式　4.3.1
right buffer　右缓存　3.19.2
selection mode　选择模式　11.6.7
server attribute stack　服务器属性栈　4.2
source color　源颜色　4.3.3
specular-reflection parameters　镜面反射　10.20.6
spotlights　投射光源　10.20.5
state variables　状态变量　4.1
stencil buffer　模板缓存　3.19.2
subpatterns(textures)　子图案(纹理)　10.21.10
surface-illumination parameters　表面散射　10.20.6
surface-rendering functions　表面绘制函数　10.20.11
surface-texture functions　表面纹理函数　10.21.2
symbolic constants　符号常量　2.9.1
symmetric perspective-projection function　对称透视投影函数　7.10.4
teapot functions　茶壶函数　8.6.2
texture borders　纹理边界　10.21.1
texture color options　纹理图案的颜色选项　10.21.4
texture coordinates　纹理坐标　10.21.1
texture-coordinate arrays　纹理坐标数组　10.21.8
texture functions　填充图案函数　4.14.1
texture homogeneous coordinates　齐次纹理坐标　10.21.15
texture-mapping options　纹理映射选项　10.21.5
texture mode　纹理模式　5.17.2
texture names　纹理名　10.21.9
texture options　纹理选项　10.21.16
texture proxy　代理纹理　10.21.13
texture reduction patterns　纹理缩减图案　10.21.11
texture state(current)　纹理状态(当前)　10.21.9
texture subpatterns　纹理子图案　10.21.10
texture wrapping　纹理环绕　10.21.6
three-dimensional viewing functions　三维观察函数　7.1
three-dimensional (volume) texture space　三维(体) 纹理空间　10.21.3
transparency functions　透明性函数　10.20.10
trimming curves　修剪函数　8.18.5
two-dimensional reference frame　二维参考系统　3.2
two-dimensional(surface) texture space　二维(表面) 纹理空间　10.21.2
two-dimensional viewing functions　二维观察函数　2.9.5
unit surface-normal vectors　单位表面法向量　10.20.11
Utility (GLU)　实用库　2.9.2
Utility Toolkit (GLUT)　实用函数工具包　2.9.2
vertex arrays　顶点数组　3.17
viewport function　视口函数　6.4.3
view-position vector　视点向量　7.10.1
visibility detection functions　可见性检查函数　9.14
volume-texture functions　体纹理函数　10.21.3
WGL (Windows interface)　Windows 系统的 OpenGL 接口　2.9.2
wire-frame methods　线框图方法　4.14.3
wire-frame visibility methods　线框面可见性方法　9.14.3
X Window system interface(GLX)　X Window 系统的 OpenGL 接口　2.9.2
Opaque object　不透明物体　10.4
Open Inventor　OpenGL 的面向对象工具包　2.9.2
Optical touch panel　光学触摸板　2.4.8
Order (spline curve continuity)　阶 (样条曲线连续性)　8.8.2
Ordered dither　按序抖动　10.9.2
Ordinary differential equations　常微分方程　A.14.4
Orthogonal basis vectors　正交基　A.4.2
Orthogonal coordinates　正交坐标系　A.1.5
Orthogonal projection:　正交投影:
axonometric　轴测　7.6.1
elevations　立面图　7.6
isometric　等轴测　7.6.1
normalization transformation　规范化变换　7.6.4

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

normalization transformation matrix 规范化变换矩阵 7.6.4
normalized view volume 规范化观察体 7.6.4
plan view 平面图 7.6
projection coordinates 投影坐标系 7.6.2
view volume 观察体 7.6.3
Orthonormal basis vectors 正交标准基 A.4.2
Out code 外部码 6.7.1
Outline font 轮廓字体 3.2
Output primitives（see also Graphics output primitives）输出图元（又见图形输出元素） 2.8.2
Outside surface 分离表面 9.8
Overhauser spline Overhauser 样条 8.9.3
Overlapping surface 重叠表面 9.8

P

Paintbrush programs 画笔程序 1.6
Painter's algorithm（depth sorting） 画家算法（深度排序） 9.6
Panning effects 摇镜头效果 6.1
Parabola 抛物线 3.11.1
Parallel algorithms: 并行算法:
area filling 区域填充 4.9
curve generation 曲线生成 3.12
line displays 线段显示 3.6
Parallelepiped view volume 平行管道观察体 7.6.4
Parallel projection 平行投影 7.1.2
axonometric 轴测平行投影 7.6.1
cabinet 斜二测投影 7.7.2
cavalier 斜等测投影 7.7.2
elevation view 立面图 7.6
isometric 等轴测 7.6.1
normalization transformation 规范化变换 7.6.4
oblique 斜平行投影 7.7
orthogonal 正平行投影 7.6.3
plan view 平面图 7.6
principal axes 主轴 7.6.1
shear transformation 错切变换 7.7.1
vector 投影向量 7.7.3
view volume 观察体 7.6.3
Parametric continuity（spline） 参数连续性（样条） 8.8.2
Parametric representations 参数表示 A.9
circle 圆 3.9
curve 曲线 A.9
ellipse 椭圆 3.1
ellipsoid 椭球面 8.4.2
parabola 抛物线 3.11.1
sphere 球面 8.4
spline 样条 3.11.2
straight line segment 直线段 6.5
surface 曲面 A.9
torus 环面 8.4.3
Parametrized animation system 参数系统 13.5
Parity（odd-even）rule 奇偶性规则 3.15.5
Partial differential equations 偏微分方程 A.14.5
Partial derivative 偏导数 A.10
Particle systems 微粒系统 8.25
Partitioning pictures 图形分割 3.22
Passive-matrix LCD 无源矩阵 LCD 2.1.5
Path（text） 路径 4.15
Path vector（ray tracing） 路径向量（光线跟踪） 10.11.1
Pattern fill 图案填充 4.14.1
Pattern mapping（see also Texture mapping） 图案映射（又见纹理映射） 10.17
PCX PC Paintbrush 文件格式 15.4.10
Peano curve Peano 曲线 8.23.3
Peano space Peano 空间 8.23.4
Pel 像素 2.1.2
Pen and brush attributes 画笔或画刷的属性 4.5.3
Pen plotter 笔式绘图仪 2.5
Penumbra shadow Penumbra 阴影 10.11.9
Perfect reflector 理想反射器 10.3.3
Periodic animation motions 周期性动画运动 13.8
Period（monochromatic light wave） 周期 12.1.1
Persistence 余辉 2.1.1
Perspective projection 透视投影 7.1.2
center of 投影中心 7.8
coordinates 坐标系 7.8.1
cone of vision 视觉圆锥体 7.8.4
field-of-view angle 视场角 7.8.6
frustum 棱台 7.8.4
homogeneous-coordinate representation 齐次坐标表示 7.8.5
matrix 矩阵 7.8.5
normalized transformation coordinates 规范化变换坐标 7.8.8
normalized transformation matrix 规范化变换矩阵 7.8.8
normalized view volume 规范化观察体 7.8.8
oblique frustum 斜棱台 7.8.7
oblique transformation matrix 斜变换矩阵 7.8.7
one-point 一点 7.8.3
principal vanishing point 主灭点 7.8.3
pyramid of vision 视觉棱锥体 7.8.4
reference point 投影参考点 7.8
shearing matrix 错切矩阵 7.8.7
special cases 特殊情况 7.8.2
symmetric frustum 对称棱台 7.8.6
three-point 三点 7.8.3
two-point 两点 7.8.3

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

vanishing points　灭点　7.8.3
view volume　观察体　7.8.4
Perturbation function（bump mapping）　扰动函数（凹凸映射）　10.18
PET（Position emission tomography）　定位发射造影术　1.8
PGL（Presentation manager interface to OpenGL）　2.9.2
Phase angle　相位角　13.7.1
PHIGS（Programmer's Hierarchical Interactive Graphics Standard），PHIGS+　程序员级的分层结构交互图形系统，扩充PHIGS　2.8.3
Phong specular-reflection model　Phong镜面反射模型　10.3.3
Phong surface rendering　Phong表面绘制　10.10.3
Phosphor　荧光　2.1.1
Photon　量子光能　2.1.1
Photon energy　光子能　10.12.1
Photon mapping　光子映射　10.14
Physically based:　基于物理的：
animations　动画　13.5
modeling　建模　8.26
Pick:　拾取:
coordinates　坐标　11.2.6
distance　距离　11.2.6
highlighting　醒目显示　11.2.6
input device　输入设备　11.2
object　对象　11.2.6
window　窗口　11.2.6
Picking operations in OpenGL　OpenGL拾取操作　11.6.7
PICT　Apple公司图画数据格式　15.4.8
Picture element（pixel）　图形元素（像素）　2.1.2
Picture partitioning　分割图形　3.22
Piecewise approximation（spline）　分段逼近（样条）　3.11.2
Pie chart　饼图　1.1
Pitteway-Watkins antialiasing　Pitteway-Watkins算法　4.17.7
Pivot point（rotation）　旋转点　5.1.2
Pixblt（color-pixel block transfer）　彩色像素块移动　5.6
Pixel　像素　2.1.2
Pixel-array primitives　像素阵列图元　3.18
Pixel coordinates　像素坐标　3.1.1
Pixel mask　像素掩模　3.18
Pixel-order scanning（texture）　像素次序扫描（纹理）　10.17.2
Pixel patterns（halftone）　像素模式（半色调）　10.9.1
Pixel phasing　像素移相　4.17
Pixel ray　像素光线　10.11.1
Pixel weighting mask　子像素的加权掩模　4.17.2
Pixmap　像素图　2.1.2
Planck's constant　普朗克常数　10.12.1
Plane:　平面:
back face　后向面　3.15.8
clipping　裁剪　7.6.3
coefficients　系数　3.15.7
complex　复　A.6
equations　方程　3.15.7
front face　前向面　3.15.8
normal vector　法向量　3.15.8
parameters　参数　3.15.7
Plan view　平面图　7.6
Plasma-panel display　离子体板显示器　2.1.5
Plotters　绘图仪　2.5
PNG　可移植网络图形格式　15.4.4
Point:　点:
attributes　属性　4.4
clipping　裁剪　6.5
in coordinate space　坐标系中　A.2.1
properties　性质　A.2.1
Point light source　点光源　10.1.1
Point size（character）　字大小　4.15
Polar coordinates　极坐标系　A.1.3
Polar form（complex number）　（复数的）极坐标形式　A.6.5
Polyhedra　多面体　8.1
properties　性质　A.13
ray-tracing intersections　光线跟踪求交　10.11.3
Polygon（see also Fill area）　多边形（又见填充区域）　3.15
active edge list　活化边表　4.1
area　面积　A.12.1
attributes　属性　4.9
back face　后向面　3.15.8
centroid　质心　A.12.2
characteristic　特征　8.8.1
classifications　分类　3.15.1
concave　凹　3.15.1
control　控制　8.8.1
convex　凸　3.15.1
convex angle　凸角　3.15.4
degenerate　退化　3.15.1
edge　边　3.15.1
edge flag（OpenGL）　边标志　4.14.3
edge table　边表　3.15.6
edge vector　边向量　3.15.2
fill methods（see also Fill-area algorithms）　填充方法（又见填充区算法）　4.9
front face　前向面　3.15.8
geometric data table　几何数据表　3.15.6

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

inside-outside tests（see also Plane） 内－外测试（又见平面） 3.15.5
interior angle 内角 3.15.1
mesh 网格 3.14
normal vector 法向量 3.15.8
plane equations 平面方程 3.15.7
plane parameters 平面参数 3.15.7
ray intersection 光线求交 10.11.4
side 边 3.15.1
simple 简单 3.15
sorted edge table 有序边表 4.1
splitting concave polygons 分割凹多边形 3.15.3
splitting convex polygons 分割凸多边形 3.15.4
standard 标准 3.15
stitching effects 缝线的效果 4.14.3
surface-facet table 面片表 3.15.6
vertex table 顶点表 3.15.6
vertices 顶点 3.15
tables 表 3.15.6
Polygon clipping: 多边形裁剪:
parallel methods 并行算法 6.8.1
Sutherland-Hodgman Sutherland-Hodgman多边形裁剪算法 6.8.1
three-dimensional 三维 7.11.4
two-dimensional 二维裁剪 6.7
Weiler-Atherton Weiler-Atherton多边形裁剪算法 6.8.2
Polygon-detail facets 多边形表面细节面片 10.16
Polygon rendering: 多边形绘制:
constant intensity 恒定强度 10.10.1
fast Phong 快速 Phong 10.10.4
flat 平面 10.10.1
Gouraud Gouraud 10.10.2
intensity interpolation 强度插值 10.10.2
Mach boundaries 马赫带 10.10.2
normal-vector interpolation 法向量插值 10.10.3
Phong Phong 10.10.3
Polyline 折线 3.4
bevel join 斜切连接 4.5.1
display algorithms 显示算法 3.1.1
miter join 斜角连接 4.5.1
round join 圆连接 4.5.1
Polymarker 多点标记 3.2
Polynomial curve(see also Spline curve） 多项式曲线（又见样条曲线） 3.11.2
Popularity color reduction 基于出现频率的降色法 15.2.2
Position emission tomography(PET） 定位发射造影术 1.8
Positioning methods（interactive） 定位方法 11.4.1
Position vector 位置向量 A.4.1
Postfiltering（see also Antialiasing） 后过滤（又见反走样） 4.17
Precision（text attribute） 文字精度属性 4.15
Prefiltering（see also Antialiasing） 前滤波（又见反走样） 4.17
Presentation Manager interface to OpenGL（PGL） OpenGL的表示管理接口 2.9.2
Pressure-sensitive joystick 压力感应式操纵杆 2.4.4
Primary color 基色 12.2.1
Primitives（see also Graphics output primitives） 图元 2.8.2
Principal axes 主轴 7.6.1
Principal vanishing point 主灭点 7.8.3
Printers 打印机 2.5
Procedural texture mapping 过程式纹理映射 10.17.5
Production rules 过程规则 8.24
Progressive refinement（radiosity） 逐步求精（辐射度） 10.12.3
Projecting square line cap 突方帽的粗线 4.5.1
Projection: 投影:
center of 投影中心 7.8
frustum 棱台 7.8.4
plane 平面 7.1.1
reference point 投影参考点 7.8
vector（oblique） 向量（斜） 7.7.3
view volume 观察体 7.6.3
Projection transformation: 投影变换:
axonometric 轴测 7.6.1
cabinet 斜二测投影 7.7.3
cavalier 斜等测投影 7.7.3
isometric 等轴测 7.6.1
oblique parallel 斜平行 7.7
oblique perspective 斜透视 7.8.7
orthogonal 正 7.6
orthographic 正交 7.6
parallel 平行 7.1.2
perspective 透视 7.1.2
principal axes 主轴 7.6.1
symmetric perspective 对称的透视 7.8.6
Proportional font 比例宽度字体 3.2
Pseudo-color methods 伪彩色方法 8.27.1
Psychological color characteristics 颜色的心理学特征 12.1.2
Pure color 纯色 12.2.2
Pure imaginary number 纯虚数 A.6
Purity（light） （光的）纯度 12.1.2
Purple line（in chromaticity diagram） （色度图中）紫色线 12.3.3
Pyramid of vision（perspective projection） 视觉棱锥体（透视投影） 7.8.4

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Q

Quadric curves　二次曲线　8.4
Quadric surfaces　二次曲面　8.4
Quadrilateral mesh　四边界网格　3.17
Quadtree　四叉树　8.21
Quantum energy levels　量子能级　2.1.1
Quaternion　四元数　A.7
　addition　加法　A.7
　fractal constructions　分形构造　8.23.6
　imaginary terms　虚数项　A.7
　inverse　逆　A.7
　magnitude　值　A.7
　multiplication　乘法　A.7
　ordered-pair representation　有序对表示　A.7
　rotations　旋转　5.11.3
　scalar multiplication　标量乘法　A.7
　scalar part　标量部分　A.7
　vector part　向量部分　A.7

R

Radial intensity attenuation（light source）　辐射强度衰减（光源）　10.1.3
Radiance　辐射　10.12.1
Radiant:　辐射的：
　energy（radiance）　（辐射）能　10.12.1
　energy transfer　能量转移　10.12.2
　exitance　（辐射）出率　10.12.1
　flux　通量　10.12.1
　intensity　强度　10.12.1
　power　能量　10.12.1
Radiosity（radiant exitance）　辐射度　10.12
　basic model　基本模型　10.12.2
　equation　方程　10.12.2
　form factors　形状因子　10.12.2
　hemicube　半立方体　10.12.2
　hemisphere　半球　10.12.2
　progressive refinement　逐步求精　10.12.3
　reflectivity factor　反射因子　10.12.2
　surface enclosure　表面的闭包　10.12.2
Random dither（noise）　随机抖动（噪音）　10.9.2
Random midpoint-displacement methods　随机中点位移方法　8.23.7
Random number generator　随机数发生器　A.14.3
Random-scan monitor　随机扫描显示器　2.1.3
Random walk　随机走动　8.23.6
Raster:　光栅:
　animations　动画　13.1.2
　file（buffer）　文件　2.1.2
　font　字体　3.2
　operations　工作原理　2.1.1
Raster-scan display（monitor）　光栅扫描显示器　2.1.2
　aspect ratio　纵横比　2.1.2
　beam retrace　电子束回扫　2.1.2
　bilevel　双层　2.1.4
　bitmap　位图　2.1.2
　bit planes　平面数　2.1.2
　color　颜色　2.1.2
　color buffer　颜色缓存　2.1.2
　depth　深度　2.1.2
　frame　帧　2.1.2
　frame buffer　帧缓存　2.1.2
　gray scale　灰度　2.1.2
　horizontal retrace　水平回扫　2.1.2
　interlacing　隔行　2.1.2
　pixel　像素　2.1.2
　pixmap　像素图　2.1.2
　refresh buffer　刷新缓存　2.1.2
　refresh rates　刷新频率　2.1.1
　resolution　分辨率　2.1.1
　scan line　扫描行　2.1.2
　vertical retrace　垂直回扫　2.1.1
Raster-scan system　光栅扫描系统　2.2
　cell encoding　单元编码　2.2.2
　display controller　显示控制器　2.2
　display coprocessor　显示协处理器　2.2.2
　display processor　显示处理器　2.2.2
　graphics controller　图形控制器　2.2.2
　run-length encoding　行程长度编码　2.2.2
　scan conversion　扫描转换　2.2.2
　video controller　视频控制器　2.2
Raster stair-step line　阶梯效应线段　3.5
Raster transformations　光栅变换　5.6
Rate-of-change integral theorems　变化率积分定理　A.11
Rate-of-change operators　变化率算子　A.10
Rational spline　有理样条　8.15
Raw data　原始数据　15.1
Raw raster file　原始光栅文件　15.1
Ray casting:　光线投射：
　constructive solid geometry　结构实体几何　8.2
　visible-surface detection　可见面判别　9.1
Ray tracing　光线跟踪　10.11
　adaptive sampling　适应性采样　10.11.8
　adaptive subdivision　适应性细分　10.11.6
　angle of refraction　折射角　10.11.1
　antialiasing　反走样　10.11.8
　basic algorithm　基本算法　10.11.1
　bounding volumes　包围体　10.11.5
　bundles　多组　10.11.6
　camera focusing effects　照相机的聚焦效果　10.11.7
　camera-lens effects　照相机镜头的效果　10.11.9

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

cell traversal 单元贯穿 10.11.6
codes 代码 10.11.9
cone tracing 锥体跟踪 10.11.8
distributed rays 分布式光线 10.11.9
eye ray（pixel ray） 视线（像素光线） 10.11
extended light source 附加光源 10.11.9
jittering 抖动 10.11.9
light-buffer method 光线缓存方法 10.11.6
motion blur 动感模糊 10.11.9
path vector 路径向量 10.11.1
pixel（primary）ray 初始光线 10.11.1
polyhedra intersections 多面体求交 10.11.4
in radiosity model 辐射度模型 10.12.3
ray equation 光线方程 10.11.2
reflection ray 反射光线 10.11
refraction ray 折射光线 10.11.1
reverse light ray 逆向光线 10.11.1
secondary rays 从属光线 10.11.1
shadow ray 阴影光线 10.11.1
space subdivision 空间分割 10.11.6
sphere intersections 球面求交 10.11.3
stochastic sampling 随机采样 10.11.9
supersampling 过采样 10.11.8
surface intersection calculations 表面求交计算 10.11.2
transmission vector 透射方向 10.11.1
tree 树 10.11.1
uniform subdivision 均匀分割 10.11.6
Readable typeface 可读性较好的字体 3.2
Real part（imaginary number） （复数的）实部 A.6
Real-time animation 实时动画 2.2.1
Rectangular parallelepiped view volume 矩形平行管道观察体 7.6.4
Reduction patterns（texture mapping） 缩减图案（纹理映射） 10.17.4
Reference point（perspective projection） 投影参考点（透视） 7.8.1
Reflection（light）: （光）反射:
angle of incidence 入射角 10.3.2
coefficients 系数 10.3.2
diffuse 漫反射 10.3.2
Fresnel laws Fresnel 反射定律 10.3.3
halfway vector 半角向量 10.3.3
Lambertian 朗伯 10.3.2
ray 光线 10.11.1
specular（Phong model） 镜面（Phong 模型） 10.3.3
Reflection mapping 反射映射 10.13
Reflection transformation: 反射变换:
axis 坐标轴 5.5.1
plane 反射平面 5.14.1
raster methods 光栅方法 5.6
three-dimensional 三维反射 5.14.1
two-dimensional 二维 5.5.1
Reflectivity 反射性 10.3.2
Reflectivity factor（radiosity） 反射因子（辐射度） 10.12.2
Refraction: 折射:
angle 角度 10.4.2
diffuse 漫反射 10.4
double 双重 10.4.2
index 率 10.4.2
ray 光线 10.4.2
Snell's law Snell 定律 10.4.2
transparency coefficient 透明因子 10.4.3
vector 向量 10.4.2
Refresh: 刷新:
buffer（see also Frame buffer） 缓存（又见帧缓存） 2.1.2
CRT（see also Cathode-ray tube） 阴极射线管（又见阴极射线管） 2.1
display file 显示文件 2.1.3
rate（CRT） 刷新频率 2.1.1
Region codes（clipping）: 区域码（裁剪）:
three-dimensional 三维 7.11.2
two-dimensional 二维裁剪 6.6
Registered procedure 注册 2.9.5
Relative coordinates 相对坐标 3.1.2
Rendering（see also Surface rendering） 绘制（又见表面绘制） 7.1.5
Request input mode 请求输入模式 11.3.1
Resistive touch panel 电阻式光学触摸板 2.4.8
Resolution: 分辨率:
display device 显示设备 2.1
halftone approximations 半色调近似 10.9.1
Retrace（electron beam） 回扫（电子光束） 2.1.1
REYES REYES 表面绘制系统 9.4
RGB: RGB:
chromaticity coordinates 色度坐标 12.4
color model 颜色模型 2.1.4
monitor（see also Video monitor） 监视器（又见视频监视器） 2.1.4
Riemannian space 黎曼空间 A.4.3
Right-handed Cartesian system 右手笛卡儿坐标系 2.8.1
Right-hand rule 右手定律 A.2.5
Rigid body: 刚体:
degrees of freedom 自由度 13.5
transformation 变换 5.1
Rigid motion 刚体运动 5.4.9
Roots: 根:
nonlinear equations 非线性方程 A.14.2

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

of complex numbers　复数　A.6.4
Rotation:　旋转:
angle　角度　5.1.2
in animations　在动画中　5.4.8
axis of　旋转轴　5.1.2
axis vector　轴向量　5.11.2
composition　复合变换　5.4
coordinate axis　坐标轴　5.11
general spatial axis　一般空间坐标轴　5.11.1
inverse　逆变换　5.3
matrix constructions　构造矩阵　5.4.10
matrix representation　矩阵表示　5.2
pivot point　旋转点　5.1.2
quaternion methods　四元数方法　5.11.3
raster methods　光栅方法　5.6
three-dimensional　三维　5.11
two-dimensional　二维　5.1.2
x axis　*x* 坐标轴　5.11.1
y axis　*y* 坐标轴　5.11.1
z axis　*z* 坐标轴　5.11.1
Rotational polygon-splitting method　旋转多边形分割法　3.15.3
Round line cap　圆帽线　4.5.1
Round-off and approximation errors (animation)　近似取整误差(动画)　5.4.8
Round polyline join　圆折线连接　4.5.1
Row vector (matrix)　(矩阵) 行向量　A.5
Rubber-band methods　橡皮条方法　11.4.5
Runga-Kutta algorithm　Runga-Kutta 算法　A.14.4
Run-length encoding　行程长度编码　2.2.2

S

Sample input mode　取样模式　11.3.1
Sampling(see also Antialiasing)　取样(又见反走样)
adaptive　适应性采样　10.11.8
area　取样区域　10.11.7
line segment　线段　3.5.1
Nyquist frequency　Nyquist 取样频率　4.17
Nyquist interval　Nyquist 取样间隔　4.17
prefiltering　前滤波　4.17
postfiltering　后过滤　4.17
rates (animation)　帧率(动画)　13.1.1
supersampling　过取样　4.17
weighting masks　加权掩模　4.17.2
Sans-serif typeface　无衬线字体　3.2
Saturation (light)　饱和度(光)　12.1.2
Scalar data visualization　数据集可视化　8.27
Scalar product (vectors)　(矢量的) 标量积　A.2.4
Scaling:　缩放:
composition　复合　5.4.1
differential　差值　5.1.3
factors　系数　5.1.3
fixed position　基准位置　5.4.5
general scaling directions　通用定向缩放　5.4.6
inverse　逆　5.3
matrix representation　矩阵表示　5.2
nonuniform (differential)　(非一致) 差值　5.1.3
parameters (factors)　参数　5.1.3
raster methods　光栅方法　5.6
three-dimensional　三维　5.12
two-dimensional　二维　5.1.3
uniform　一致缩放　5.1.3
Scan conversion　扫描转换　3.1
Scan line　扫描行　2.1.2
Scan-line algorithms:　扫描方法:
circle generation　圆生成　3.9.1
coherence properties　相关特征　4.1
convex-polygon fill　凸多边形的扫描线填充　4.11
curved-boundary fill methods　曲线边界区域的扫描线填充　4.12
ellipse generation　椭圆生成　3.1
polygon fill　多边形填充　3.1
straight-line segments　直线段　3.4
visible-surface detection　可见面判别　9
Scan-line interlacing　隔行扫描线　2.1.2
Scan-line number　扫描行号　3.1.1
Scanner　扫描仪　2.4.7
Scanning(texture patterns)　扫描(纹理图案)　10.17.2
Scientific visualization (see also Data visualization)　科学计算可视化(又见数据可视化)　1.4
Screen coordinates:　屏幕坐标:
three-dimensional　三维　3.1.1
two-dimensional　二维　3.1.1
Scripting animation system　剧本系统　13.5
Secondary ray　从属光线　10.11.1
Second-order continuity:　二次连续性:
geometric　参数　8.8.2
parametric　段(图形子部分)　3.22
Segment (picture subsection)　结构 (图形子部分)　3.22
Self-affine fractals　自仿射分形　8.23.2
Self-inverse fractals　自逆分形　8.23.2
Self-similar fractals　自相似分形　8.23.2
Self-squaring fractals　自平方分形　8.23.2
Serif typeface　有衬线字体　3.2
Server　服务器　2.6
setPixel procedure　setPixel 函数　3.1.1
Shades (color)　明暗　12.2.2
Shading (see Illumination models; Surface rendering)　阴影(见光照模型, 表面绘制)　10
Shadow mask　荫罩法　2.1.4

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Shadow ray 阴影光线 10.11.1
Shadows: 阴影:
modeling 建模 10.6
penumbra 半影 10.11.9
umbra 本影 10.11.9
Shape grammars 形状语法 8.24
Shear transformation: 错切变换:
axis 轴 5.5.2
matrix 矩阵 5.5.2
parameters 错切参数 5.5.2
in oblique parallel projections 斜投影 7.7
in oblique perspective projections 透视投影 7.8
three-dimensional 三维 5.12
two-dimensional 二维 5.5.2
x-direction *x* 方向 5.1.3
y-direction *y* 方向 5.1.3
z-direction *z* 方向 5.12
Shift vector（see also Translation） 位移向量（又见变换） 5.1.1
Side（polygon） 边（多边形） 3.15
SIGGRAPH（Special Interest Group in Graphics） 计算机学会图形学专业委员会 2.7
Similarity dimension（fractal） 自相似维数（分形） 8.23.3
Simple polygon 简单多边形 3.15
Simpson's rule Simpson 定律 A.14.3
Simulations 模拟
Simulators 模拟系统 1.5
Simultaneous linear-equation solving 并列线性方程组求解 A.14.1
Singular matrix 奇异矩阵 A.5.5
Slope-intercept line equation 斜率截距直线方程 3.5.1
Snell's law Snell 定律 10.4.2
Snowflake（fractal） 雪花形状（分形） 8.23.4
Soft fill 软填充 4.9.2
Soft-object model 柔性对象模型 8.7
Software standards 软件标准 2.8.3
Solid angle 立体角 A.1.7
Solid modeling: 实体建模:
constructive solid geometry 结构实体几何 8.2
sweep constructions 扫描构造 8.19
Solid texture 实体图案 10.17.3
Sonic digitizer 声音数字化仪 2.4.6
Sorted edge table 有序边表 4.1
Spaceball 空间球 2.4.3
SpaceGraph system SpaceGraph 系统 2.1.6
Space-partitioning methods（ray tracing）: 空间分割方法（光线跟踪）:
adaptive 适应性细分 10.11.6
light buffer 光线缓存 10.11.6
ray bundles 光线束 10.11.6
uniform 均匀分割 10.11.6
Space-partitioning representations 空间分区表示 8
Spectral color 光谱颜色 12.1.1
Spectral radiance 光谱辐射能 10.12.1
Spectrum（electromagnetic） 频谱（电磁） 12.1.1
Specular reflection 镜面反射 10.2
angle 镜面反射角 10.3.3
coefficient 镜面反射系数 10.3.3
exponent 镜面反射指数 10.3.3
Fresnel laws Fresnel 反射定律 10.3.3
halfway vector 半角向量 10.3.3
Phong model Phong 模型 10.3.3
vector 向量 10.3.3
Specular transmission（refraction） 镜面折射 10.4.1
Speed of light 光速 12.1.1
Sphere: 球面:
equations 方程 8.1
ray intersection calculations 光线求交计算 10.11.5
Spherical coordinates 球面坐标 A.1.6
SPIFF 静止图像交换文件格式 15.4.1
Spiral 螺旋线 3.25
Spline curve 样条曲线 3.11.2
approximation 逼近 8.18
basis functions 基函数 8.8.4
basis matrix 基本矩阵 8.8.4
beta-spline beta 样条 8.14
Bézier 8.14
bias parameter 偏离参数 8.9.4
blending functions 混合函数 8.8.4
boundary conditions 边界条件 8.8
B-spline B 样条 8.12
cardinal cardinal 样条 8.12.3
Catmull-Rom Catmull-Rom 样条 8.9.3
characteristic polygon 特征多边形 8.8.1
continuity conditions 连续性条件 8.8
continuity parameter 连续性参数 8.9.4
control graph 控制图 8.8.1
control polygon 控制多边形 8.8.1
control points 控制点 8.8.1
conversions 转换 8.16
convex hull 凸壳 8.8.1
cubic interpolation 三次插值 8.9
displaying 显示 8.17
geometric continuity 几何连续性 8.8.3
Hermite Hermite 插值 8.9.2
interpolation 插值 8.8
knot vector 节点向量 8.12.1
Kochanek-Bartels Kochanek-Bartels 样条 8.9.4
local control 局部控制 8.12.1
matrix representation 矩阵表示 8.8.4
natural 自然 8.9.1

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

NURB　非均匀有理B样条　8.15
Overhauser　Overhauser样条　8.9.3
parametric continuity　参数连续性　8.8.2
rational　有理　8.15
tension parameter　张量参数　8.9.4
Spline generation:　样条生成:
Horner's method　Horner方法　8.17.1
forward-difference method　向前差分计算　8.17.2
subdivision methods　细分方法　8.17.3
Spline surface　样条曲面　8.13
Bézier　Bézier　8.13
B-spline　B样条　8.14
Cartesian-product representation　笛卡儿积表示　8.8.5
displaying　显示　8.17
trimming　修剪　8.8.6
Split-screen effects　分画面效果　6.3.4
Splitting concave polygons:　分割凹多边形:
rotational method　旋转法　3.15.3
vector method　向量方法　3.15.3
Splitting convex polygons　分割凸多边形　3.15.4
Spotlights　投射光源　10.3
Spring:　弹簧:
constant　常数　8.26
network（nonrigid body）　网络（非刚性对象）　8.26
Square matrix　方阵　A.5
Squash and stretch（animation）　挤压和拉伸（动画）　13.3
Staging（animation）　分级（动画）　13.3
Stair-step line　阶梯线　3.5
Standard graphics object　标准图形对象　3.14
Standard polygon　标准多边形　3.15
State:　状态:
machine　状态机　4
parameters　状态参数　4
system　状态系统　4
variables　状态变量　4
Statistically self-similar fractal　统计自相似分形　8.23.2
Steradian　立体弧度　A.1.7
Stereoscopic viewing　立体感视图　2.1.7
Stitching（polygon edge gaps）　缝线　4.14.3
Stochastic sampling　随机采样　10.11.9
Stoke's theorem　斯托克斯定理　A.11.1
Storyboard（animation）　情节板（动画）　13.2
Streamlines　流线　8.27.2
String input device　字符串输入设备　11.2.3
Stroke font　笔划字　3.2
Stroke input device　笔划输入设备　11.2.2
Stroke-writing（vector）display　笔画（向量）显示器　2.1.3
Structure（picture subsection）　结构（子图）　3.23.1
Stylus（digitizer）　触笔（数字）　2.4.6
Subdivision methods:　细分方法:
adaptive ray tracing　适应性光线跟踪　10.11.6
BSP tree　BSP树　8.22
fractal generation　分形生成　8.23
octree　八叉树　8.22
spline generation1　样条生成　8.23
uniform ray tracing　均匀光线跟踪　10.11.6
Substitutional algorithm　替代算法　15.3.2
Subtractive color model（CMY）　CMY颜色模型　12.6.1
Superellipse　超椭圆　8.5.1
Superellipsoid　超椭球面　8.5.2
Superquadrics　超二次曲面　8.5
Supersampling　过取样　4.17
Surface:　表面:
blobby　柔性对象　8.7
curved（see also Curved surface）　曲面　8.3
detail　细节　10.15
explicit representation　显式表示　A.8
fractal　分形　8.23
implicit representation　隐式表示　A.8
nonparametric representation　非参数表示　A.8
parametric representation　参数表示　A.8
plane　平面　8.4
quadric　二次曲面　8.4
spline（see also Spline surface）　样条（又见样条曲面）　8.8
superquadric　超二次曲面　8.5
tessellation　细分　3.14
trimming curve　修剪曲线　8.8.6
weighting　加权　4.17
Surface clipping　表面裁剪　7.11.3
Surface contour plots　曲面的层位线显示　9.12.2
Surface detail　表面细节:　10.15
bump mapping　凹凸映射　10.18
environment mapping　环境映射　10.13
frame mapping　帧映射　10.19
polygon facets　多边形面片　10.16
procedural texturing　过程式纹理映射　10.17.5
texture mapping　纹理映射　10.17
Surface enclosure（radiosity）　表面闭包（辐射度）　10.12.2
Surface-facet table　面片表　3.15.6
Surface lighting effects（see also Illumination model）　表面光照效果（又见光照模型）　10.2
Surface normal vector　平面的法向量　3.15.8

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Surface rendering（see also Surface detail） 表面绘制（又见表面细节） 10.1
constant-intensity 恒定强度 10.10.1
fast Phong 快速 Phong 明暗处理 10.10.4
flat 平面绘制 10.10.1
Gouraud Gouraud 明暗处理 10.10.2
intensity interpolation 强度插值表面绘制 10.10.2
Mach bands 马赫带 10.10.2
normal-vector interpolation 法向量插值绘制 10.10.3
Phong Phong 明暗处理 10.10.3
photon mapping 光子映射 10.14
polygon methods 多边形绘制算法 10.1
radiosity 辐射度 10.12
ray-tracing 光线跟踪 10.11
Surface tessellation 表面细分 3.14
Surface tiling 区域平铺 4.9.1
Surrounding surface 包围表面 9.8
Sutherland-Hodgman polygon clipping Sutherland-Hodgeman 多边形裁剪 6.8.1
Sweep representations 扫描表示 8.19
Symbol: 符号:
hierarchies 层次 14.1.2
instance 实例 14.1.1
modeling 建模 14.1
Symmetry: 对称:
circle 圆 3.9.1
in curve-drawing algorithms 曲线显示方法 3.11
ellipse 椭圆 3.1
System intensity levels 系统强度等级 10.8

T

Table: 表格:
attribute 属性 3.15.6
edge 边 3.15.6
geometric 几何 3.15.6
surface-facet 面片 3.15.6
vertex 顶点 3.15.6
Tablet（see also Digitizer） 数据板（又见数字化仪） 2.4.6
Targa Targa 图形适配器 15.4.11
Task planning 任务安排 1.1
TCP/IP TCP/IP 2.7
Teapot 茶壶 8.6.2
Tension parameter（spline） 张量参数（样条） 8.9.3
Tensor 张量 A.3
contraction 缩减 8.27.3
data visualization 数据场的可视化 8.27.3
dimension 维数 A.3
metric 度量 A.4.3
rank 阶 A.3
transformation properties 转换性质 A.3
Terrain（fractal） 地面 8.23.7
Tessellated surface 细分的表面 3.14
Texel 纹理元 10.17
Text（see also Character） 文字（又见字符）
alignment 字符对齐 4.15
attributes 字符属性 4.15
clipping 字符裁剪 6.1
path 字符路径 4.15
precision 精度 4.15
Texture mapping: 纹理映射:
array 纹理数组 10.17
coordinates 纹理坐标 10.17
functions 纹理函数 10.17
image-order scanning 图像次序扫描 10.17.2
inverse scanning 逆扫描 10.17.2
mip maps 纹理缩减图案 10.17.4
one-dimensional（linear） 一维纹理（线） 10.17
pixel-order scanning 像素次序扫描 10.17.2
procedural 过程式纹理 10.17.5
r coordinate *r* 坐标 10.17.3
reduction patterns 缩减图案 10.17.4
s coordinate *s* 坐标 10.17.1
solid 实体图案 10.17.3
space 纹理空间 10.17
t coordinate *t* 坐标 10.17.2
texel 纹理元 10.17
three-dimensional（volume） 三维纹理（体） 10.17.3
two-dimensional（surface） 二维纹理（表面） 10.17.2
Texture scanning 纹理扫描 10.17.2
TGA Truevision 图形适配器格式 15.4.11
Thin-film electroluminescent displays 薄膜光电显示器 2.1.5
Thin-lens equation 薄透镜公式 10.11.7
Three-point perspective projection 三点透视投影 7.8.3
TIFF 标记图像文件格式 15.4.3
Tiling（surface） 区域平铺 4.9.1
Time chart 时间图 1.1
Timing（animation） 定时 13.3
Tints（color） 色泽 12.2.2
Tint fill 色彩填充 4.9.2
Tomography 层析 X 射线造影术 1.8
Tones（color） 色调 12.2.2
Topline（character） 顶线 4.15
Topological: 拓扑:
covering（fractal） 覆盖 8.23.3
dimension 维数 8.23.3
Torus 环面 8.4.3
Touch panels 触摸板 2.4.8
Trackball 跟踪球 2.4.3

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

Training applications　培训　1.5
Transformations:　变换:
　affine　仿射　5.16
　basic geometric　基本几何　5.1
　commutative　交换　5.4.7
　composite　复合　5.4.1
　computational efficiency　计算效率　5.4.8
　coordinate system　坐标系　5.8
　geometric　几何　2.8.2
　inverse（see also individual transformations）　逆变换（又见单一变换）　5.3
　matrix representations　矩阵表示　3.1.1
　modeling　建模　2.8.2
　noncommutative　不可交换　5.4.7
　parallel projection　平行投影　7.1.2
　perspective projection　透视投影　7.1.2
　raster methods　光栅方法　5.6
　reflection　反射　5.5
　rigid body　刚体　5.1.1
　rotation　旋转　5.1.2
　scaling　缩放　5.1.3
　shear　错切　5.5.2
　three-dimensional geometric　三维几何　5.9
　three-dimensional viewing　三维观察　7.1
　translation　平移　5.1.1
　two-dimensional geometric　二维几何　5.1
　two-dimensional viewing　二维观察　6
　viewing　观察　2.8.2
　viewport　视口　7.9
　window-to-viewport　窗口到视口　6.1
　workstation　工作站　6.3.4
　world-to-viewing coordinate　世界坐标系转换到观察坐标系　2.8.1
Translation:　平移:
　composition　复合　5.4.3
　distances　平移距离　5.1.1
　inverse　逆　5.3
　matrix representation　矩阵表示　5.4.1
　raster methods　光栅方法　5.6
　three-dimensional　三维　5.1
　two-dimensional　二维　5.1.1
　vector　向量　5.1.1
Translucent material　半透明材料　10.4.1
Transmission vector（refraction）　透射向量（折射）　10.4.2
Transparency（see also Refraction; Ray tracing）　透明（又见折射；光线跟踪）
　basic model　基本的透明模型　10.4.3
　coefficient　系数　10.4.3
　factor　因子　4.9.2
　Snell's law　Snell 定律　10.4.2
　vector　透射向量　10.4.2
Transparent material　透明材料　10.4.2
Transpose（matrix）　矩阵转置　A.5.3
Trapezoid rule　梯形定律　A.14.3
Triangle meshes　三角形网络　3.16
Trimming curve　修剪曲线　8.8.6
Tristimulus vision theory　三刺激视觉理论　12.4
True-color system　真彩色系统　2.1.4
Two-point perspective projection　两点透视投影　7.8.3
Typeface（see also Font）　字体　3.2

U

Umbra shadow　本影　10.11.9
Under-sampled animation displays　采样过低动画放映　13.9
Uniform B-splines　均匀的 B 样条　8.12.1
Uniform color reduction　均匀降色法　15.2.1
Uniform scaling　一致缩放　5.1.3
Uniform spatial subdivision:　均匀空间细分:
　octree　八叉树　8.21
　ray tracing　光线追踪　8.22
Unit cube（view volume）　单位立方体（观察体）　7.6.4
Up vector（character）　向上向量（字符）　4.15
URL（Uniform Resource Locator）　统一资源定位器　2.7
User:　用户:
　dialogue　对话　11.8.1
　help facilities　帮助功能　11.8.3
　interface（see also Graphical user interface）　界面（又见图形用户界面）　11.8.2
　model　模型　11.8.1
uvn coordinate system　*uvn* 坐标系统　7.3.3

V

Valuator input device　定值设备　11.2.4
Vanishing point　灭点　7.8.3
Variable:　变量:
　dependent　因变量　A.8
　independent　自变量　A.8
Varifocal mirror　变焦反射镜　2.1.6
Vector:　向量:
　addition　向量加　A.2.3
　basis　基向量　A.4
　column（matrix）　列向量（矩阵）　A.5
　cross product　叉积　A.2.5
　data visualization　数据集可视化　8.27
　dimension　维数　A.3
　direction angles　方向角　A.2.2
　direction cosines　方向余弦　A.2.2
　dot product　点积　A.2.4

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

edge 边向量 3.15.2
halfway 半角向量 10.3.3
inner product 向量内积 A.2.4
knot 节点 8.12.1
light source 光源 10.1
magnitude（length） 大小（长度） A.2.2
oblique parallel projection 斜平行投影 7.7
operations 操作 A.5
position 位置向量 A.4.1
product 标量积 A.2.4
properties 性质 A.2
in quaternion representation 四元数表示 A.7
row（matrix） 行（矩阵） A.5
scalar multiplication 标量乘 A.2.3
scalar product 标量积 A.2.4
space 空间 A.3
specular reflection 镜面反射 10.3.3
surface normal 法向量 A.10.1
translation 平移 5.1.1
transmission（refraction） 透射（折射） 10.4.2
viewer（lighting model） 视点（光照模型） 10.3.3
Vector file 向量文件 2.1.3
Vector method（polygon splitting） 向量方法（多边形分割） 3.15.3
Vector monitor（see also Random-scan monitor） 向量显示器（又见随机扫描监视器） 2.1.3
Vertex（polygon） 顶点（多边形） 3.15.1
Vertex array（OpenGL） 顶点数组（OpenGL） 3.17
Vertex table（polygon） 顶点表（多边形） 3.15.6
Vertical retrace 垂直回扫 2.1.2
Vibrating mirror 振动镜 2.1.6
Video controller 视频控制器 2.2
Video display（see also Video monitor） 视频显示器（又见视频监视器） 2.1
Video lookup table 视频查找表 10.8.2
Video monitor（see also Cathode-ray tube） 视频监视器（又见CRT） 2.1
active-matrix 激活矩阵 2.1.5
calligraphic 笔迹显示器 2.1.3
color CRT 彩色CRT 2.1.4
composite 组合色 2.1.4
CRT design CRT设计 2.1
electroluminescent 光电显示器 2.1.5
emissive 发射显示器 2.1.5
graphics server 图形服务器 2.6
flat-panel 平板显示器 2.1.5
full-color 全彩色 2.1.4
gas-discharge 气体放电显示器 2.1.5
high-definition 高清晰度系统 2.1.1
large-screen 大屏幕 2.3
LCD（liquid crystal device） LCD（液晶显示器） 2.1.5
LED（light-emitting diode） LED（发光二极管） 2.1.5
multi-panel 多板显示屏 2.3
nonemissive 非发射显示器 2.1.5
passive-matrix 无源矩阵 2.1.5
plasma panel 等离子体显示板 2.1.5
random-scan 随机扫描 2.1.3
raster-scan 光栅扫描 2.1.2
refresh CRT 刷新式CRT 2.1.1
resolution 分辨率 2.1.1
RGB RGB颜色模型 2.1.4
server 服务器 2.6
stereoscopic 立体感 2.1.7
storage CRT 存储CRT 2.1.1
thin-film electroluminescent 薄膜光电显示器 2.1.5
three-dimensional 三维观察设备 2.1.6
true-color 真彩色 2.1.4
vector 向量 2.1.3
workstations 工作站 2.3
View angle（field-of-view） 观察角度（视场角） 7.8.6
Viewing: 观察:
exploded and cutaway displays 拆散和剖切面视图 7.1.6
panning 摇镜头 6.1
stereoscopic 立体视图 7.1.7
three-dimensional 三维 7.1.7
two-dimensional 二维 6.1
zooming 拉镜头 6.1
Viewing coordinates: 观察坐标:
left-handed 左手系 A.1.5
three-dimensional 三维 7.1.7
two-dimensional 二维 6.1
uvn *uvn*观察坐标系统 7.3.3
Viewing pipeline: 观察流水线:
three-dimensional 观察 7.2
two-dimensional 二维 7.2
Viewing transformations（see also Projection） 观察变换（又见投影）
camera parameters 照相机参数 7.1.1
clipping algorithms 三维裁剪算法 7.11
clipping planes 裁剪平面 7.2
clipping window 裁剪窗口 6.1
display window 显示窗口 6.1
far clipping plane 远裁剪平面 7.6.3
frustum 棱台 7.8.4
near clipping plane 近裁剪平面 7.6.3

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

normalized coordinates　规范化设备坐标系　6.1
normalized projection coordinates　规范化投影坐标　7.9
normalized square　规范化正方形　6.3.2
normalized viewport　规范化视口　6.3.1
normalized view volume　规范化观察体　7.8.8
oblique parallel projection　斜平行投影　7.7
oblique perspective projection　斜透视投影　7.8.7
orthogonal projection　正投影　7.6
parallel projection　平行投影　7.5
perspective projection　透视投影　7.8
pipeline　流水线
projection plane　投影平面　7.1.1
projections　投影　7.1.1
screen coordinates　屏幕坐标系　7.9, A.1.1
split-screen effects　分画面效果　6.3.4
symmetric perspective projection　对称透视投影　7.1
three-dimensional　三维　7.1.1
two-dimensional　二维　6.1
uvn coordinates　*uvn* 观察坐标系　7.3.3
view plane　观察平面　7.3.3
view point　观察点　7.3.4
viewport transformation　视口变换　7.9
view up vector　观察向上向量　7.3.2
view volume　观察体　7.2
workstation transformation　工作站变换　6.3.4
world-to-viewing coordinates　世界坐标系到观察坐标系的变换　7.4
View plane　观察平面　7.3.3
clipping window　裁剪窗口　7.6.3
normal vector　法向量　7.3.1
position　观察位置　7.3
Viewport（see also Clipping window）　视口（又见裁剪窗口）　6.1
normalized　规范化　6.3
in screen coordinates　屏幕坐标系　6.3.2
three-dimensional（see also View volume）　三维（又见观察体）　7.3.4
View reference point　观察参考点　7.8.1
View-up vector:　观察向上向量:
three-dimensional　三维　7.3.4
two-dimensional　二维　6.1
View volume　观察体　7.2
normalized　规范化观察体　7.6.4
perspective projection　透视投影　7.8
parallelepiped　平行管道　7.6.3
parallel projection　平行投影　7.7
unit cube　单位立方体　7.6.4
Virtual reality:　虚拟现实:
applications　应用　1.3
systems　虚拟现实系统　2.1.7
modeling language（VRML）　建模语言（VRML）　2.8
Visible-line detection（see also Depth cueing）　可见线判别（又见深度提示）　9.13
Visible-surface detection　可见面的判定　7.1.4
A-buffer method　A 缓存算法　9.4
algorithm classification　算法分类　9.1
algorithm comparisons　算法的比较　9.11
area-subdivision method　区域细分算法　9.8
back-face elimination　后向面消除　9.2
BSP-tree method　BSP 树　9.7
curved surfaces　曲面　9.12
depth-buffer method　深度缓存算法　9.3
depth cueing　深度提示　9.13.2
depth-sorting method　深度排序算法　9.6
image-space methods　像空间　9.1
object-space methods　物空间　9.1
octree methods　八叉树　8.21
painter's algorithm（depth sorting）　画家算法（又见深度排序）　9.6
ray-casting method　光线投射算法　9.1
scan-line method　扫描线算法　9.5
surface contour plots　面的层位线　9.12.2
wire-frame methods　线框可见性算法　9.13
Vision（tristimulus theory）　视觉（三刺激理论）　12.4
Visualization:　可视化:
applications　应用　1.4
methods（see also Data visualization）　方法（又见数据可视化）　8.27
Voice input systems　语音输入系统　2.4.10
Volume calculations（CSG）　结构实体几何　8.2
Volume element　体素　8.21
Volume rendering　体绘制　8.27.1
Volume texture patterns　体纹理图案　10.17.3
Voxel　体元　8.21
VRML（Virtual-reality modeling language）　VRML（虚拟现实建模语言）　2.8

W

Warn model（light source）　Warn模型（光源）　10.1.6
Wavelength（light）　波长（光）　12.1.1
Weighted sampling　加权取样　4.17.2
Weighting:　加权:
mask（pixel）　加权掩模　4.17.2
surface　加权表面　4.17.4
Weiler-Atherton polygon clipping　Weiler-Atherton 多边形裁剪算法　6.8.2
WGL（Windows-to-OpenGL interface）　WGL（窗口系统到OpenGL的接口）　2.9.2

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。

www.Linuxidc.com

White light 白色光 12.1.1
Winding number 环绕数 3.15.5
Window（see also Clipping window; Display window; Viewing transformation; Viewport） 窗口（又见裁剪窗口、显示窗口） 6.1
Window manager 窗口管理 1.9
Windows to OpenGL interface（WGL） 窗口系统到OpenGL的接口（WGL） 2.9.2
Window-to-viewport mapping 窗口到视口的映射 6.1
Wire frame: 线框图:
depth-cueing methods 深度提示 7.1.3
displays 线框图显示 3.14
visibility algorithms 线框图可见性算法 9.13
Workstation 工作站 2.3
Workstation transformation 工作站变换 3.3.4
World coordinates 世界坐标系 2.8.1
World-to-viewing-coordinate transformation 世界坐标系到观察坐标系的变换 7.4
World Wide Web 万维网 2.7

X

x-axis rotation 绕 *x* 轴旋转 5.11.1
XBM X Window 系统位图格式 15.4.5
x-direction shear *x* 方向错切 5.5.2
XPM X Window 系统像素图格式 15.4.5
XWindow System X 窗口系统 2.9.2
XYZ color model XYZ 颜色模型 12.3.1

Y

y-axis rotation 绕 *y* 轴旋转 5.11.1
YC_rC_b color model YC_rC_b 颜色模型 12.5.3
y-direction shear *y* 方向错切 5.5.2
YIQ color model YIQ 颜色模型 12.5.2
YUV color model YUV 颜色模型 12.5.3

Z

z-axis rotation *z* 轴旋转 5.11.1
z buffer（depth buffer） *z* 缓存（深度缓存） 9.3
z-direction shear 相对于 *z* 轴的错切 5.14.2
Zero-order continuity: 0 阶连续性:
geometric 几何 8.8.1
parametric 参数 8.8.2
Z mouse Z 鼠标 2.4.3
Zooming effects 拉镜头的效果 6.1

Linux公社（www.LinuxIDC.com）是专业的Linux系统门户网站，实时发布最新Linux资讯。