TURING 图灵计算机科学丛书

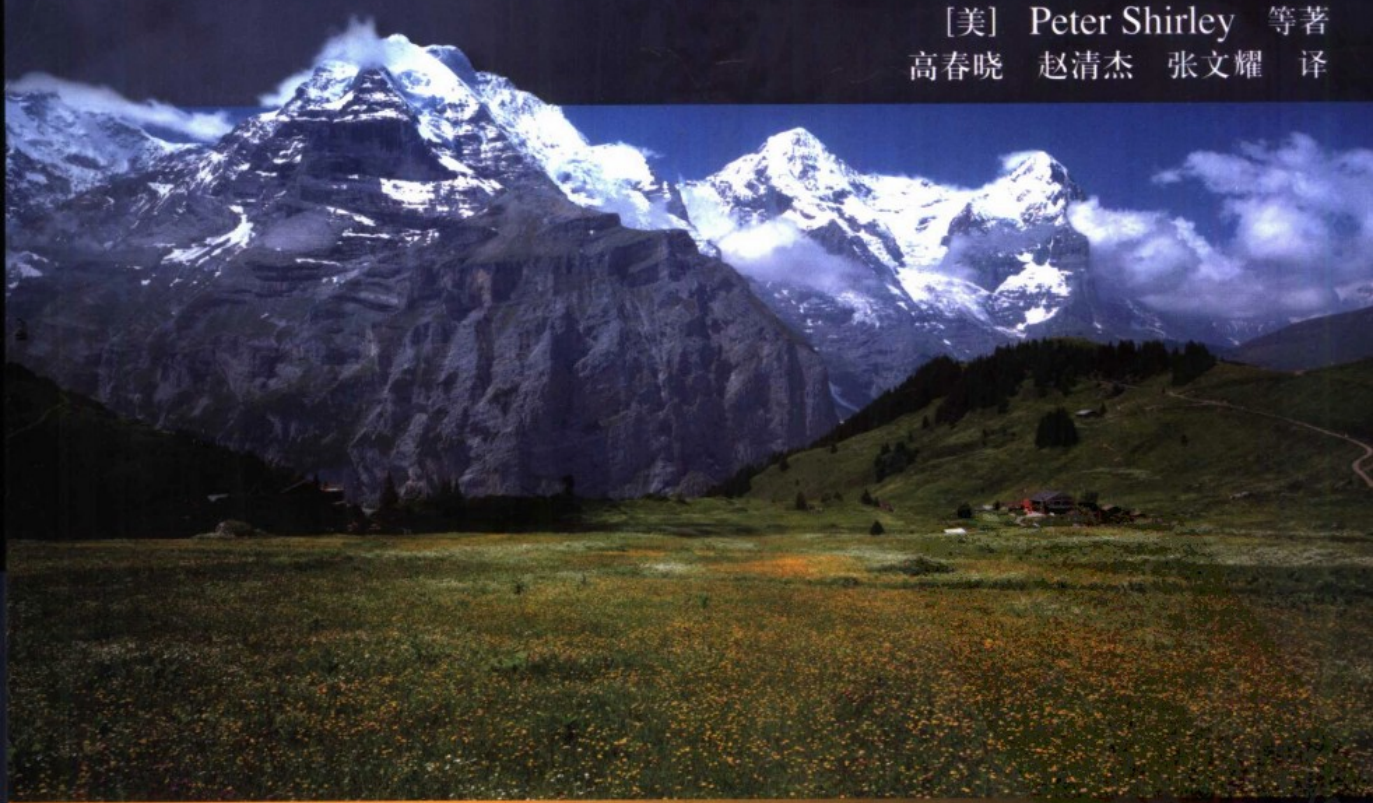
计算机图形学

（第2版）

Fundamentals of Computer Graphics

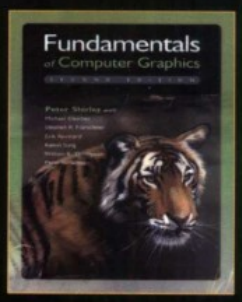
Second Edition

[美] Peter Shirley 等著
高春晓 赵清杰 张文耀 译



人民邮电出版社
POSTS & TELECOM PRESS

计算机图形学
（第2版）



Fundamentals of
Computer Graphics
Second Edition

在计算机图形学的建立和发展过程中，美国犹他大学曾发挥了关键性的作用。1968年，计算机图形学先驱同时也是本领域第一位图灵奖得主Ivan Sutherland在此任教，先后吸引和培养了包括图灵奖得主Alan Kay在内的一大批图形学大师。犹他大学也因此成为世界计算机图形学首屈一指的学术重镇。

现在，犹他大学的计算机图形学教授Peter Shirley将这一伟大传统凝聚在本书之中。他在书中采取突出原理的方法，全面介绍了计算机图形学及相关的基础理论（包括数学和物理理论），有助于读者对这一学科的内在脉络有深入的理解，而不是过早地纠缠于具体的、变化很快的各种图形API。与此同时，作者注意吸收近年来层出不穷的新理论、新方法，很好地将理论和最新的业界实践结合起来，使读者能够学以致用。第2版作者还邀请了康奈尔大学、威斯康星大学等名校的同行就各自的擅长领域撰写了7章，更极大地丰富了原书的内容。

正因为具备以上特色，本书出版之后受到广泛推崇，成为计算机图形学领域教材的后起之秀，迅速被全球数十所高校用作教材或者推荐教材。其中包括美国麻省理工学院、加州大学伯克利分校、卡耐基-梅隆大学、斯坦福大学、哥伦比亚大学、佐治亚理工学院、加拿大多伦多大学、我国香港大学等众多名校。



Peter Shirley

计算机图形学领域世界知名的学者，尤以光线跟踪方面的研究闻名世界。曾担任*ACM Transactions on Graphics*和*Journal of Graphics Tools*副主编，多次担任SIGGRAPH程序委员会委员。他是犹他大学计算机科学系教授，在伊利诺伊大学厄巴纳-尚佩恩分校获得计算机科学博士学位。除本书之外，他还著有*Realistic Ray Tracing*。

本书相关信息请访问：**图灵网站** http://www.turingbook.com
读者/作者热线：(010) 88593802
反馈/投稿/推荐信箱：contact@turingbook.com

分类建议 计算机/计算机科学/图形学

人民邮电出版社网址 www.ptpress.com.cn

ISBN 978-7-115-15867-3

9 787115 158673 >

ISBN 978-7-115-15867-3/TP

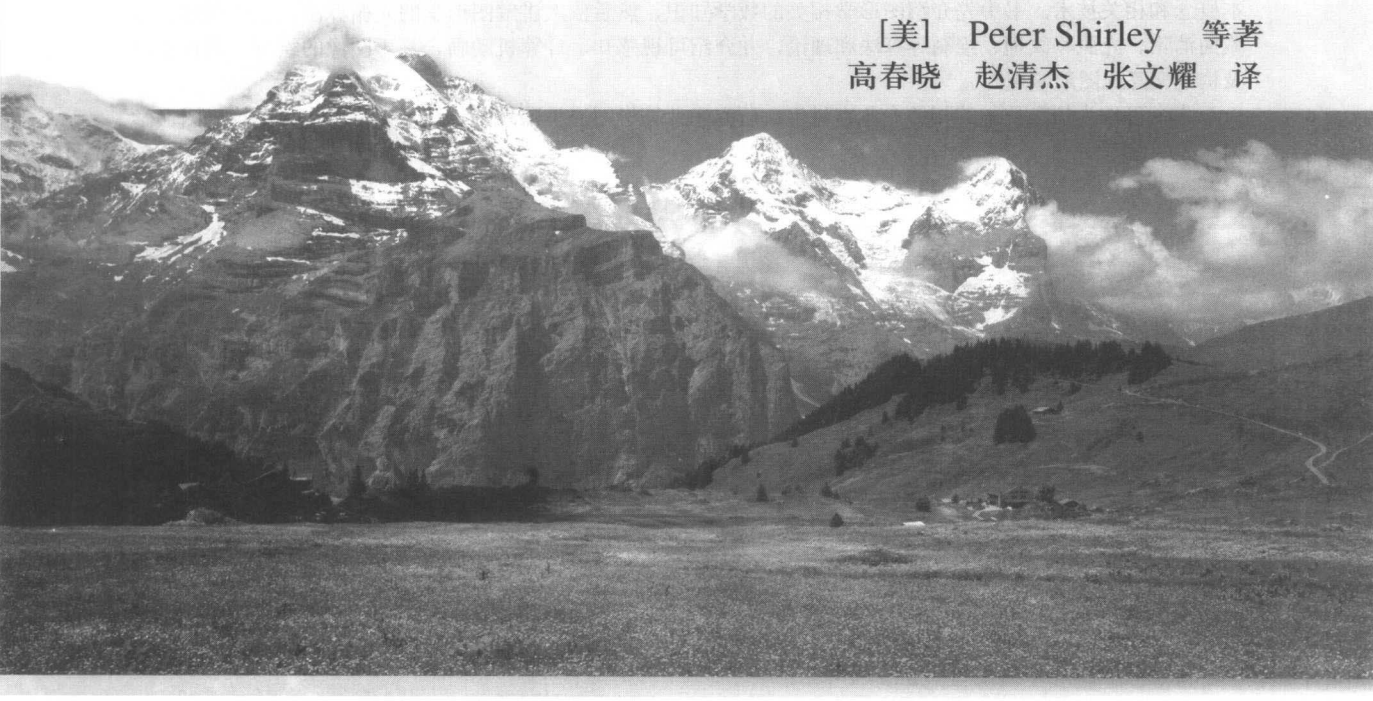
定价：49.00 元

TURING 图灵计算机科学丛书

计算机图形学
（第2版）

Fundamentals of Computer Graphics
Second Edition

[美] Peter Shirley 等著
高春晓 赵清杰 张文耀 译



人民邮电出版社
北京

图书在版编目（CIP）数据

计算机图形学：第2版/（美）舍利（Shirley, P.）等著；高春晓，赵清杰，张文耀译.
—北京：人民邮电出版社，2007.6
（图灵计算机科学丛书）

ISBN 978-7-115-15867-3

Ⅰ. 计… Ⅱ. ①舍…②高…③赵…④张… Ⅲ. 计算机图形学 Ⅳ. TP391.41

中国版本图书馆CIP数据核字（2007）第024186号

Original English edition: Fundamentals of Computer Graphics, Second Edition
@ 2005 A K Peters, Ltd. All Rights Reserved.
本书中文版由A K Peters公司授权人民邮电出版社独家出版。
版权所有，侵权必究。

内 容 提 要

本书是国外高校采用率很高的计算机图形学教材，共分为26章，全面系统地讲解了计算机图形学的基本概念和相关技术。书中先介绍图形学相关的数学知识，然后依次讲解图形学的光栅算法、三维观察、隐藏面消除、光照、纹理、绘制等算法和理论，并介绍可视感知、计算机动画、基于图像的绘制、可视化以及构建交互式图形应用等。

本书可作为信息技术等相关专业本科生、研究生计算机图形学课程的教材，也可以作为计算机图形学工作者的参考用书。

图灵计算机科学丛书

计算机图形学（第2版）

◆ 著 [美] Peter Shirley 等
译 高春晓 赵清杰 张文耀
责任编辑 杨海玲 黄 倩

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京密云春雷印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092 1/16
印张：26.5 彩插：4
字数：743千字 2007年6月第1版
印数：1-4 000册 2007年6月北京第1次印刷

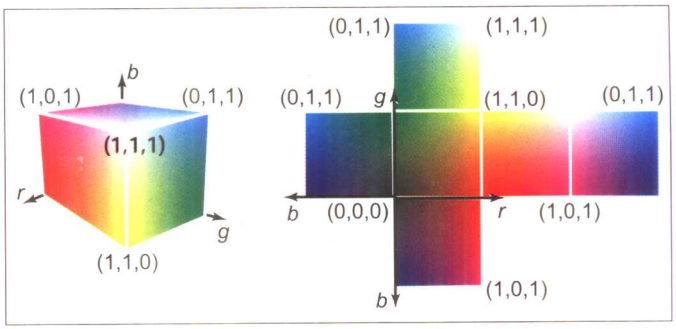
著作权合同登记号 图字：01-2006-0314号

ISBN 978-7-115-15867-3/TP

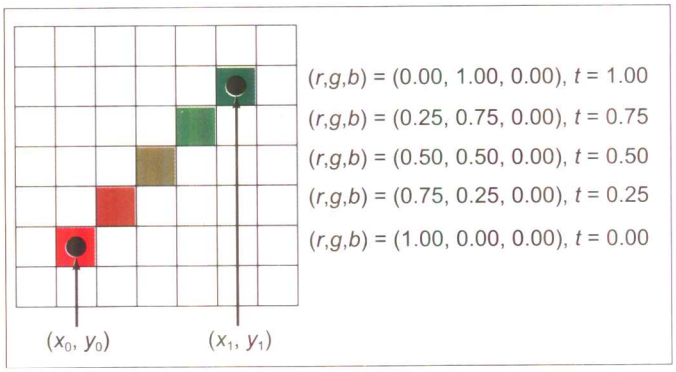
定价：49.00元

读者服务热线：(010)88593802 印装质量热线：(010)67129223

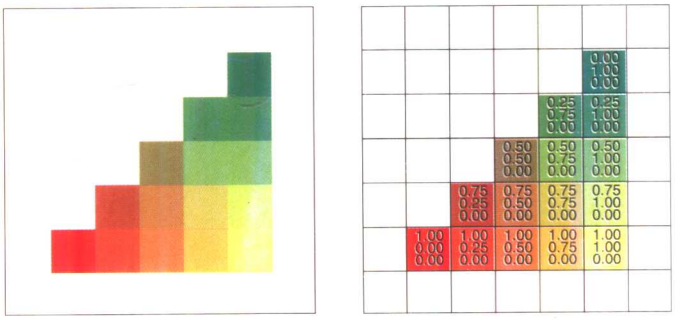
彩图 I RGB 颜色立方体的三维形式及其展开面。任何 RGB 颜色都是立方体中的一个点（另见图 3-4）



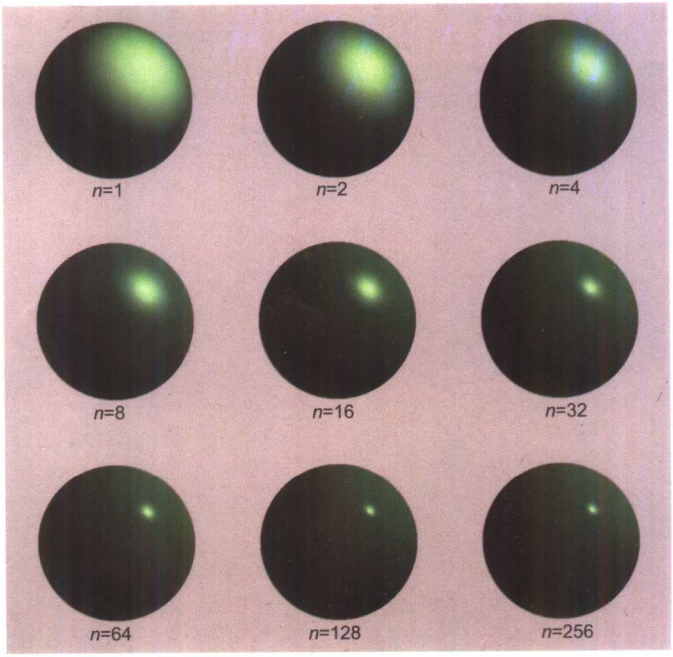
彩图 II 一条从红变到绿的彩色直线。中点像素的颜色是半红半绿，即“深黄色”（另见图 3-9）

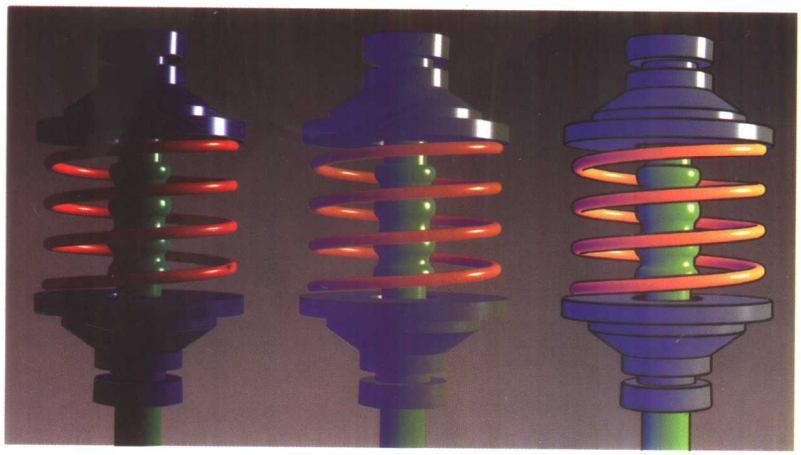


彩图 III 用重心插值的彩色三角形。注意在每行、每列及沿着每条边的颜色分量的变化是线性的。实际上，沿着每条线的变化都是线性的，如对角线（另见图 3-10）

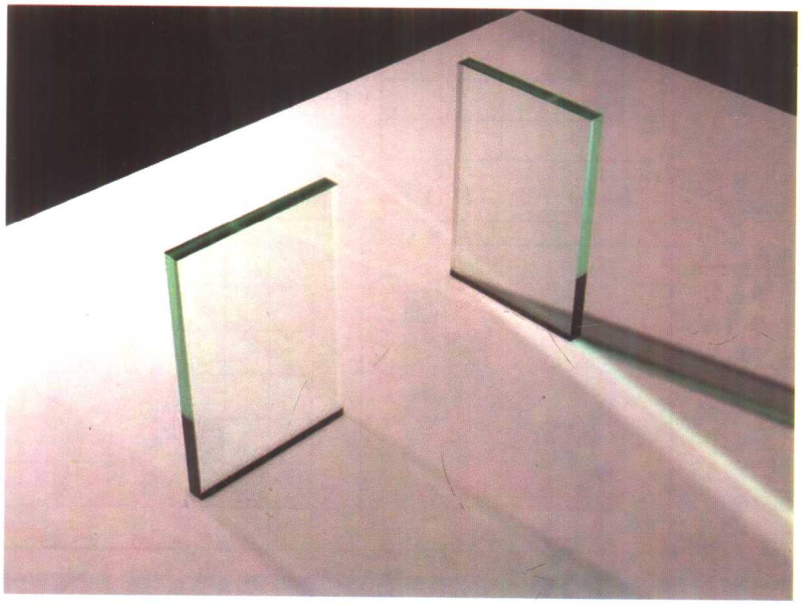


彩图 IV Phong 指数作用在高光特性上的效果。这里使用式（9-5）计算高光。另外还有漫反射分量，它使物体具有光泽，但呈现非金属外观。图片经 Nate Robins 授权使用（另见图 9-6）





彩图 V 左：Phong 光照图像。中：没有轮廓线，冷暖明暗处理的效果不好。右：具有轮廓线的冷暖明暗处理。图片经 Amy Gooch 授权使用（另见图 9-9）

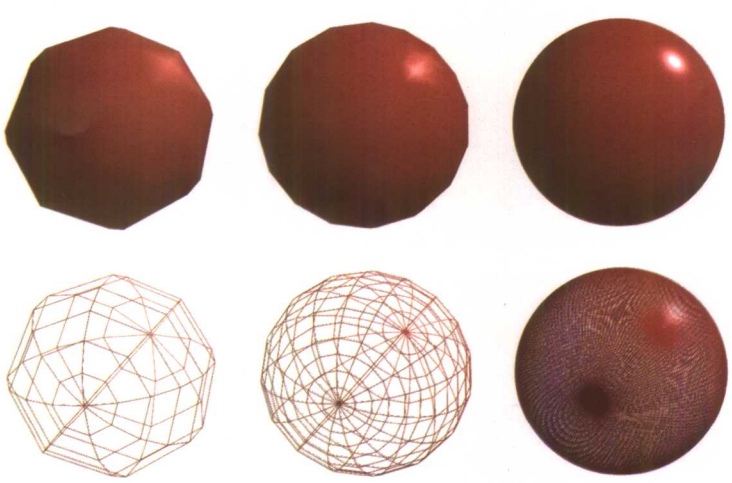


彩图 VI 玻璃的颜色受到完全内部反射和 Beer 定律的影响。光线透过和反射的量由菲涅尔公式决定。地面上复杂的光照是通过第 23 章中的粒子跟踪方法计算的（另见图 10-11）

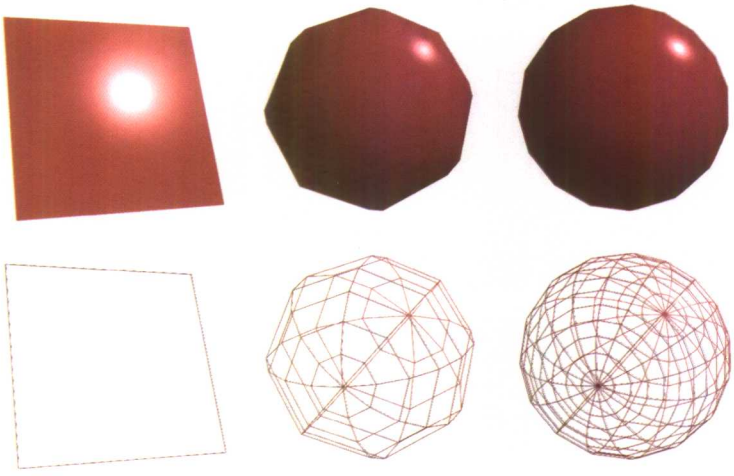


彩图 VII 一个景深的示例。酒瓶阴影中焦散现象是通过第 23 章中的粒子跟踪方法计算的（另见图 10-34）

彩图 VIII 每个球体仅使用一个顶点着色器计算 Phong 明暗处理。因为计算是基于每个顶点偏差进行的，只有在用于建立球体模型的几何模型的数量大大增加后，Phong 高光看起来才是正确的（另见图 17-7）



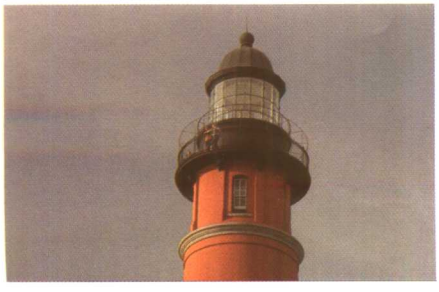
彩图 IX 运行 17.3.4 节的片段着色器得到的结果。注意在最左边的模型的确出现了 Phong 高光，这个模型是用一个多边形表示的。实际上，因为光照是在片段上而不是在每个顶点上计算，所以较粗糙的镶嵌球体模型也出现了适当的 Phong 明暗处理效果（另见图 17-8）







彩图 X 每个通道的伽马校正能够降低图像的饱和度。左图是令 s=0.5 减小饱和度的结果，右图是没有减小饱和度的结果（s=1）（另见图 22-11）



彩图 XI 用来演示颜色变换技术的图像。结果显示在彩图 XII 和彩图 XIV 中（另见图 22-12 和图 22-30）





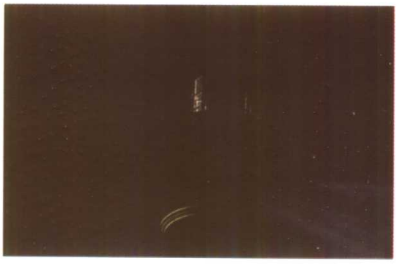
彩图 XII 左边的图像用来调整彩图 XI 所示图像的颜色，结果显示在右边（另见图 22-13）



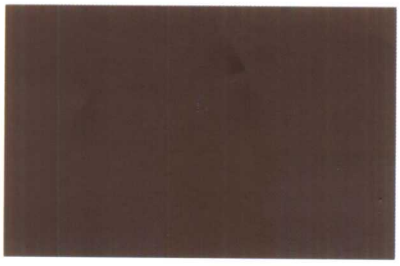


彩图 XIII 针对颜色校正的线性插值。在左边的图像中参数 c 设为 0.0，在右边的图像中 c 为 1.0（另见图 22-24）

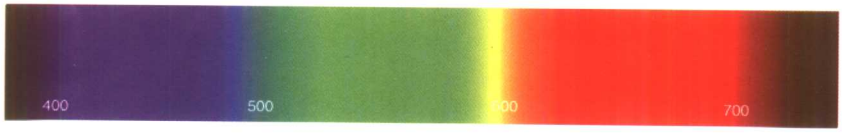




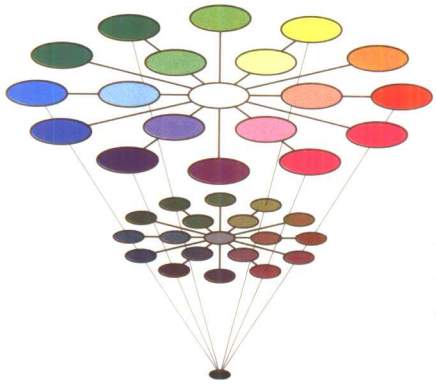
彩图 XIV 左边的图像用来将彩图 XI 中的图像变换成夜景，该夜景显示在右边（另见图 22-31）



彩图 XV 使用彩图 XI 所示图像模拟的夜景（另见图 22-30）



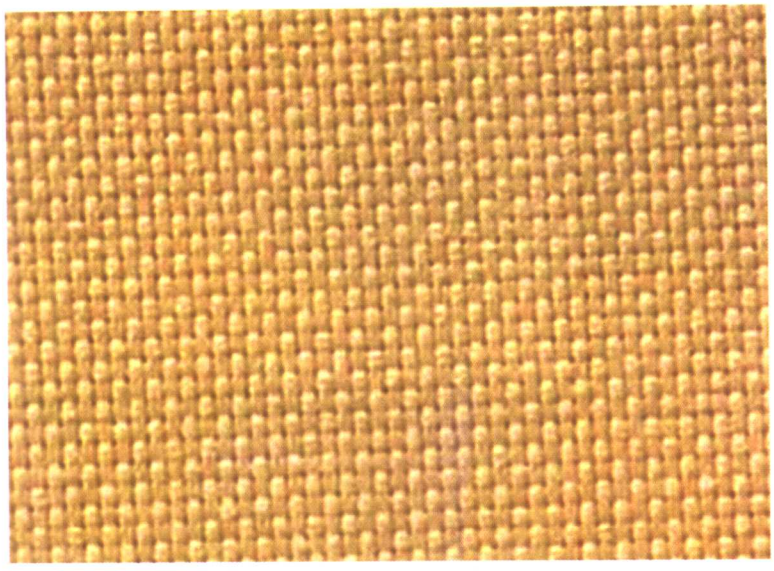
彩图 XVI 可见光谱。波长单位是纳米（nm）



彩图 XVII HSV 颜色空间。色调围绕圆环变化，饱和度随半径变化，值随着高度变化



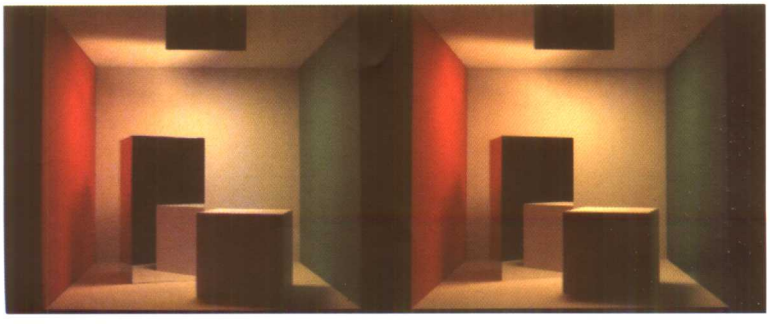
彩图 XVIII 哪种颜色更接近红色：绿色还是紫色？



彩图 XIX 当用颜色显示时，图 21-29 显示的效果更强烈。图片经 Albert Yonas 授权使用



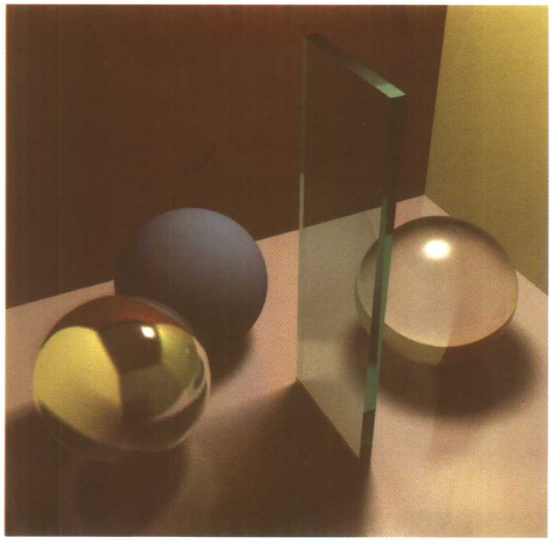
彩图 XX 空中透视，其中大气的效果降低了对比度并且使颜色偏蓝，它提供远距离的深度线索

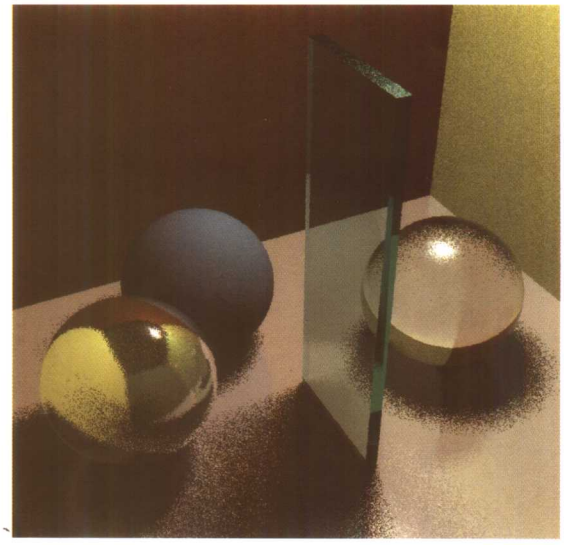


彩图 XXI 照片与绘制结果的比较。图片经 Sumant Pattanaik 以及康奈尔大学计算机图形学项目授权使用（另见图 23-9）



彩图 XXII 图像显示了极度运动模糊效果。阴影使用分布式光线跟踪，因为它们在图像中移动。由 Joseph Hamdorf 和 Young Song 建模，由 Eric Levin 绘制







彩图 XXIII 每个像素使用 1 个采样、16 个采样和 256 个采样的分布式光线跟踪计算出的图像。图片经 Jason Waltman 授权使用

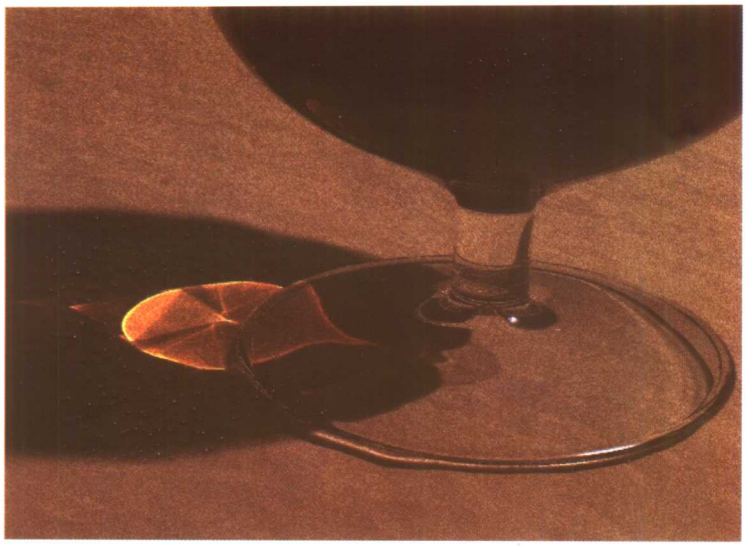




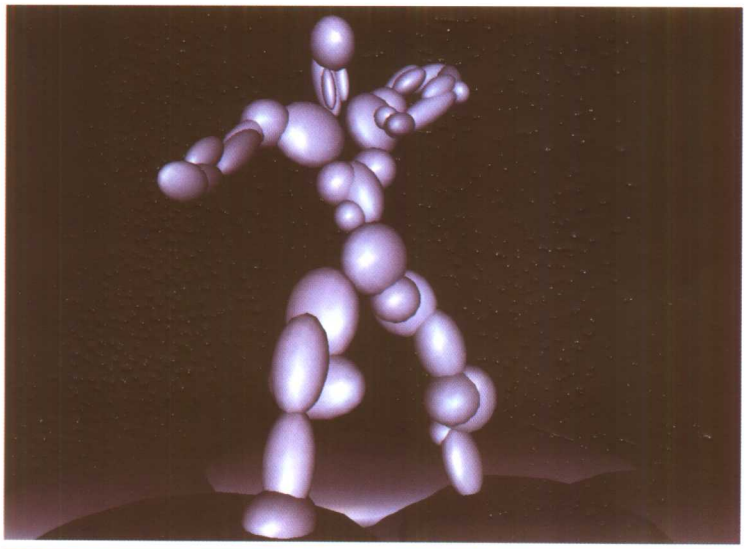
彩图 XXIV 上：使用漫反射明暗处理模型。下：使用“A Practical Model for Sub-surface Light Transport”（Jensen et al., Proceedings of SIGGRAPH 2001）中的技术，允许子表面散射。图片经 Henrik Jensen 授权使用



彩图 XXV 光线跟踪和光子映射的室内图像。大部分光照是间接的。图片经 Henrik Jensen 授权使用



彩图 XXVI 阴影中明亮的颜色模式是一种“聚光线”，是光线通过玻璃聚焦的结果。它是通过光子跟踪方法计算得到的。图片经 Henrik Jensen 授权使用





彩图 XXVII 上：一个椭圆体集合近似表示模型。下：椭圆体用于创建一个类似重力的隐式函数，然后把它替换掉。图片经 Eric Levin 授权使用

人民邮电出版社
POSTS & TELECOM PRESS

精选大师名作
打造专业品牌

TURING
图灵教育

人民邮电出版社图灵公司专注于引进国外经典教材与优秀科技图书，出版国内高校教师编写的专业教材，专业方向包括：计算机、数学、统计学和电子电气等。合作伙伴包括Prentice-Hall、Addison-Wesley、Wiley、McGraw-Hill、剑桥大学出版社、Elsevier、IEEE Press、Wrox、SIAM等多家世界知名出版公司，具有丰富的选题资源和雄厚的出版力量。

目录、前言等出版物详细信息，请浏览www.turingbook.com



书　名：软件工程(第3版)
原书名：Software Engineering: Theory and Practice
作　者：Shari Lawrence Pfleeger, Joanne M.Atlee
译　者：杨卫东
书　号：978-7-115-15829-1
定　价：59.00元
出版时间：2006年03月



书　名：Beginning Java Objects 中文版：从概念到代码
原书名：Beginning Java Objects: From Concepts to Code
作　者：Jacquie Barker
译　者：万波
书　号：7-115-15440-6
定　价：78.00元
出版时间：2006年12月



书　名：TSP：领导开发团队
原书名：TSP:Leading a Development Team,1E
作　者：Watts S.Humphrey
译　者：张家才 江贺 车皓阳
书　号：7-115-13919
定　价：35.00元
出版时间：2006年12月



书　名：Java程序设计与问题解决：基础篇(英文版·第4版)
原书名：Java:An Introduction to Problem Solving and Programming
作　者：Walter Savitch
书　号：7-115-15288-8
定　价：59.00元
出版时间：2006年11月



书　名：Java程序设计与问题解决：高级篇(英文版·第4版)
原书名：Java:An introductio n to Problem Solving and Programming
作　者：Walter Savitch
书　号：7-115-15289-6
定　价：59.00元
出版时间：2006年11月



书　名：数理逻辑（第2版）(中英文版)
原书名：AMathemati cal Introduction to Logic,2E
作　者：Herbert B.Enderton
译　者：沈复兴 陈磊 孙运传
中文版书号：7-115-14311-0
英文版书号：7-115-14145-2
中文版定价：29.00元
英文版定价：39.00元
中文版出版时间：2006年12月
英文版出版时间：2005年12月



书　名：Java程序设计语言(第4版)(中英文版)
原书名：The Java Programming Language
作　者：Ken Arnold, James Gosling, David Holmes
译　者：陈昊鹏 章程 张思博 李楠
中文版书号：7-115-15297-7
英文版书号：7-115-14762-0
中文版定价：69.00 元
英文版定价：99.00 元
中文版出版时间：2006年10月
英文版出版时间：2006年6月



书　名：UNIX环境高级编程(第2版)(中英文版)
原书名：Advanced Programming in the UNIX Environment, 2nd Edition
作　者：W.Richard Stevens, Stephen A.Rago
译　者：尤晋元 张亚英 戚正伟
中文版书号：7-115-14731-0
英文版书号：7-115-14484-2
中文版定价：99.00元
英文版定价：99.00元
中文版出版时间：2006年5月
英文版出版时间：2006年3月



书　名：数据挖掘导论（中英文版）
原书名：IntroductiontoDa ta Mining
作　者：Pang-NingTan , Michael Steinbach, Vipin Kumar
译　者：范明 范宏建
中文版书号：7-115-14698-5
英文版书号：7-115-14144-4
中文版定价：49.00元
英文版定价：59.00元
中文版出版时间：2006年5月
英文版出版时间：2006年1月



书　名：现代编译原理—C 语言描述(中英文版)
原书名：ModernCompil er Implementation in C
作　者：AndrewW.App el
译　者：赵克佳 黄春 沈志宇
中文版书号：7-115-14552-0
英文版书号：7-115-13771-4
中文版定价：45.00元
英文版定价：59.00元
中文版出版时间：2006年4月
英文版出版时间：2005年9月



书　名：UML面向对象建模与设计(第2版)(中英文版)
原书名：Object-Oriented Modeling and Design with UML, Second Edition
作　者：Michael Blaha, James Rumbaugh
译　者：车皓阳 杨眉
中文版书号：7-115-14223-8
英文版书号：7-115-14076-6
中文版定价：45.00元
英文版定价：55.00元
中文版出版时间：2006年1月
英文版出版时间：2006年1月



书　名：数据结构与问题求解——Java语言描述（第3版）
原书名：DataStructures and Problem Solving Using Java
作　者：Mark Allen Weiss
译　者：翁惠玉 严骏
书　号：7-115-14988-7
定　价：49.00元
出版时间：2006年8月



书　名：TCP/IP基础教程：基于实验的方法
原书名：TCP/IP Essentials: A Lab-Based Approach
作　者：Shivendra S.Panwar, Shiwen Mao, Jeong-dong Ryoo, Yihan Li
译　者：陈涓 等
书　号：7-115-15386-8
定　价：29.00元
出版时间：2006年11月



书　名：数据库系统：面向应用的方法
原书名：Database Systems: An Application-Oriented Approach, Introductory Version
作　者：Michael Kifer, Arthur Bernstein, Philip M.Lewis
译　者：陈立军 赵加奎 邱海艳 帅猛
书　号：7-115-15269-1
定　价：49.00元
出版时间：2006年10月



书　名：计算机网络实验教程
原书名：Computer Networking: Internet Protocols in Action
作　者：Jeanna Matthews
译　者：李毅超 曹跃 王钰
书　号：7-115-14250-5
定　价：29.00元
出版时间：2006年1月



书　名：数据结构与算法分析——C++描述(第3版)(中英文版)
原书名：Data Structures and Algorithm Analysis in C++
作　者：MarkAllenW eiss
译　者：张怀勇
中文版书号：7-115-13923-7
英文版书号：7-115-15233-0
中文版定价：49.00元
英文版定价：59.00元
中文版出版时间：2006年11月
英文版出版时间：2006年9月



书　名：UML用户指南(第2版)
原书名：The Unified Modeling Language User Guide
作　者：Grady Booch, James Raumbaugh, Ivar Jacobson
译　者：邵维忠 麻志毅 马浩海 刘辉
书　号：7-115-14833-3
定　价：49.00 元
出版时间：2006年6月



书　名：算法分析与设计
原书名：Algorithm Design: Foundations, Analysis, and Internet Examples
作　者：Michael T.Goodrich, Roberto Tamassia
译　者：霍红卫
书　号：7-115-15054-0
定　价：55.00元
出版时间：2006年9月



书　名：计算机组成：结构化方法(第5版)
原书名：Structured Computer Organization
作　者：Andrew S.Tanenbaum
译　者：刘卫东 宋佳兴 徐格
书　号：7-115-14891-0
定　价：65.00元
出版时间：2006年7月



书　名：C语言程序设计(英文版·第3版)
原书名：Programming in C
作　者：Stephen G.Kochan
书　号：7-115-14763-9
定　价：55.00元
出版时间：2006年6月



书　名：系统分析与设计(英文版·第3版)
原书名：Essentials of System Analysis and Design Third Edition
作　者：Joseph S. Valacich; Joey F. George; Jeffrey A. Hoffer
书　号：7-115-15154-7
定　价：55.00元
出版时间：2006年8月



书　名：数据结构与算法分析—C 语言描述(英文版·第2版)
原书名：DataStructures and Algorithm Analysis in C
作　者：MarkAllenW eiss
改　编：陈越
书　号：7-115-13984-9
定　价：49.00元
出版时间：2005年8月

北京图灵文化发展有限公司　电话：010-88592340，88593802　传真：010-88593803　网址http://www.turingbook.com
地　址：北京市海淀区西四环北路140号京鼎原商务楼405室　邮编：100097　E-mail：contact@turingbook.com

译 者 序

随着计算机图形学及其相关技术的飞速发展，我们跨入了一个三维时代，各种扣人心弦的三维游戏、震撼人心的虚拟场景不断冲击着人们的感官和体验。计算机图形学已经成为计算机科学中最为活跃的领域之一，在世界范围内得到了普遍的重视。

计算机图形学作为一门建立在计算机科学、数学、物理、心理学以及艺术等学科基础上的综合学科，主要是20世纪60年代以后发展起来的。在此过程中，犹他大学发挥了关键性的作用。1968年，计算机图形学先驱同时也是本领域第一位图灵奖得主Ivan Sutherland受David Evans之邀来此任教，先后吸引和培养了一批图形学的大师，包括图灵奖得主Alan Kay，z缓冲等技术的发明者Edwin Catmull（后创办Pixar动画公司），提出了反走样技术的Frank Crow，开发了Warnock算法的John Warnock（后与人合创Adobe公司），Gouraud着色技术的发明者Henri Gouraud，几何流水线之父Jim Clark（创办SGI公司），提出了Phong模型和着色技术的越南裔学者裴祥风（Bui Tuong Phong）等。在ACM SIGGRAPH颁发的计算机图形学界最高荣誉Steve A. Coons奖的获奖者中，犹他大学的教授和校友拥有5席，几乎占据一半。犹他大学也因此成为世界计算机图形学首屈一指的学术重镇。

现在，犹他大学的计算机图形学教授Peter Shirley将这一伟大传统凝聚在本书之中。作者在书中采取突出原理的方法，全面介绍了计算机图形学及相关的基础理论（包括数学和物理理论），有助于读者对这一学科的内在脉络有深入的理解，而不是过早地纠缠于具体的、变化很快的各种图形API。与此同时，作者注意吸收近年来层出不穷的新理论、新方法，很好地将理论和最新的业界实践结合起来，使读者能够学以致用。第2版作者还邀请了康奈尔大学、威斯康星大学等名校的同行就各自的擅长领域撰写了7章，更极大地丰富了原书的内容。

全书内容按照由浅到深的结构进行组织，分为26章。第1章到第6章介绍计算机图形学涉及的代数、几何、矩阵等数学知识，以及物理学、信号处理等相关知识。这使得阅读本书成为一个流畅的过程，基本上不必再为相关知识查阅其他书籍。第7章阐述两种基本的投影观察方法。第8章讨论两种经典的隐藏面消除算法。第9章介绍表面明暗处理的典型方法。第10章详细讨论光线跟踪及其加速算法，并介绍利用光线跟踪实现的特殊效果，如阴影、景深、光泽反射和运动模糊等。第11章的内容包括二维和三维纹理映射、凹凸映射以及环境映照等。第12章给出完整的图形流水线。第13章、第14章和第15章介绍图形学中典型的数据结构、采样方法以及曲线建模方法，这些是计算机图形学的重要内容，但在一般的教材中往往被忽略了。计算机动画技术的相关介绍在第16章。第17章介绍的顶点着色器和片段着色器体现了图形学发展的一个新方向。在掌握了上述各章的基础上，读者就可以建立简单的交互式图形学应用了。第18章用示例介绍了设计开发这样的应用的过程和方法。第19章到第25章进一步讨论图形学的高级主题：光照与光照模型、颜色与视觉感知、基于图像的绘制等。最后一章介绍了图形学的一个重要应用领域可视化技术。教师可以对后一部分内容有选择地进行讲解，这部分内容对于工程人员和高级程序员也具有很高的参考价值。书中还包括了难度适中的习题，并提供了参考文献和进一步阅读的推荐文献。

正因为具备以上特色，本书出版之后受到广泛推崇，已经被全球数十所高校用作教材或者推荐教材。其中包括美国麻省理工学院、加州大学伯克利分校、卡耐基-梅隆大学、斯坦福大学、哥伦比亚大学、佐治亚理工学院、加拿大多伦多大学、我国香港大学等众多名校。

翻译此书我们非常荣幸，同时也深感责任重大。但由于时间仓促，译文中欠妥和纰漏之处恐难避免，恳请读者不吝赐教和指正。

译　者

前　言

本书是以我在印第安纳大学和犹他大学多年教授的图形学课程的基础上编写的。根据内容，图形学方面的教材可分为两类：一类是讲解图形学基本原理和底层细节的；另一类是讲解如何使用现代图形学API如OpenGL、Direct3D以及Java3D的。本书属于前一种类型。这两种类型的课程我都讲过，经验表明，学习第一类课程的学生，在理解图形学原理甚至熟练使用API方面，都要强于学习第二类课程的学生。有很多理由可以证明这个结论，当然这对于其他教师或者学校也许并不完全适用。但我认为，无论是图形学还是其他学科（甚至体育运动），传授基本原理都是正确的选择。

如何使用本书

前面9章的内容对应一个学期的课程，让学生了解图形流水线以及基本的光线跟踪等内容。本书让学生全面掌握图形学的原理性知识并予以实现，它不是那种“学习OpenGL”式的教材。这里讲的图形流水线与硬件实现的图形流水线是一致的。学习本书的同学，会发现OpenGL和其他常见的API在许多方面都是类似的。

本书的第二部分内容是一些更高级的主题，安排顺序不是很严格。因此这些内容可以用于各种形式的第二学期的课程，也可以在一学期的课程中安排几周讲述其中的一些内容。

在第一学期，建议布置以下作业：

(1) 第2章后的数学作业，然后至少有一次课堂测试。

(2) 直线光栅化。

(3) 重心颜色插值的三角形光栅化。

(4) 画正射线框图。

(5) 画透视线框图。

(6) 带平面阴影三角形的BSP树以及z剪切很小且具有鼠标驱动视点选择的线框边。

(7) z剪切很小的有限精度z缓冲器实现。

这些作业之后，教师可以布置光线跟踪方面的作业，或者让学生在z缓冲器上增加阴影图、Phong光照、裁剪及纹理，或者让学生用三维API编程。

关于封面

本书英文原版的封面图片是J. W. Baker创作的*水中的老虎*（在画布上喷成的丙烯画，16"×20"，www.jwbart.com）。

在1998年的康奈尔大学研讨班上，Alain Fournier（1943—2000）借用老虎做了一次精彩的演讲。演讲中，他生动地描述了一只老虎的运动。他最后这样总结自己的观点：

在过去35年里，计算机图形学的建模和绘制已经取得了巨大的进步，但是我们仍然不能自动模拟出老虎在河中游泳的壮观景象。关于自动，我的意思是指不需要艺术家/专家去做仔细的人工调节。

坏消息是我们还有很长的路要走。

好消息仍然是我们还有很长的路要走。

在线资源

本书的配套网址是http://www.cs.utah.edu/~shirley/fcg2/。我将在网上公布勘误表，并且给课堂上使用本书的读者建立链接。尽管我没有提供本课程的幻灯片，但是Rich Riesenfeld已经慷慨地同意把他的幻灯片共享给大家，在本书网站上将有到此幻灯片的链接。书中的大部分图形都是Abobe Illustrator格式的，如果需要我很乐意将具体图形转化成可移植格式。欢迎与我联系：shirley@cs.utah.edu。

本版变化

与第1版相比，书中的内容有较多小变化，但也有两个较大的变化：增加了参考文献的数目，增添了同事们新写的内容。这些同事头脑清晰、善于思考和交流，我邀请他们每人写一章，以使本书内容能覆盖各个关键主题。他们中的多数曾经在课堂上用过本书，对书中的方法很熟悉。参考文献无意包罗万象，只是为了给读者提供进一步学习的资源。我相信其中肯定有遗漏，希望能收到关于遗漏某本关键参考书的反馈信息。新加的章节有：

- **信号处理**：Stephen Marschner，康奈尔大学（第4章）。
- **曲线**：Michael Gleicher，威斯康星大学（第15章）。
- **计算机动画**：Michael Ashikhmin，纽约州立大学石溪分校（第16章）。
- **使用图形硬件**：Peter Willemsen，明尼苏达大学德卢斯分校（第17章）。
- **构建交互式图形应用程序**：Kelvin Sing，华盛顿大学博塞尔分校（第18章）。
- **视觉感知**：William B.Thompson，犹他大学（第21章）。
- **色调再现**：Erik Reinhard，中佛罗里达大学（第22章）。

致谢

以下人士对本书提供了帮助：Josh Andersen、Zeferino Andrade、Michael Ashikhman、Adam Berger、Adeel Bhutta、Solomon Boulos、Stephen Chenney、Michael Coblenz、Greg Coombe、Frederic Cremer、Brian Curtin、Dave Edwards、Jonathon Evans、Karen Feinauer、Amy Gooch、Eungyoung Han、Chuck Hansen、Andy Hanson、Dave Hart、John Hart、Helen Hu、Vicki Interrante、Henrik Wann Jensen、Shi Jin、Mark Johnson、Ray Jones、Kristin Kerr、Dylan Lacewell、Mathias Lang、Philippe Laval、Marc Levoy、Howard Lo、Ron Metoyer、Keith Morley、Eric Mortensen、Tamara Munzner、Koji Nakamaru、Micah Neilson、Blake Nelson、Michael Nikelsky、James O'Brien、Steve Parker、Sumanta Pattanaik、Matt Pharr、Peter Poulos、Shaun Ramsey、Rich Riesenfeld、Nate Robins、Nan Schaller、Chris Schryvers、Tom Sederberg、Richard Sharp、Sarah Shirley、Peter-Pike Sloan、Tony Tahbaz、Jan-Phillip Tiesel、Bruce Walter、Alex Williams、Amy Williams、Chris Wyman、Kate Zebrose。

书中借用了Ching-Kuang Shene及David Solomon的著作中的示例。Henrik Jensen、Eric Levin、

Matt Pharr和Jason Waltman慷慨地提供了大量图片。Brandon Mansfield对光线跟踪部分的层次包围体内容做了很多改进。Carrie Ashust、Jean Buckley、Molly Lind、Pat Moulis和Bob Shirley为本书提供了很多后勤支持。Miranda Shirley为我们带来很多快乐。

非常感谢J. W. Baker帮忙设计出我想要的封面。他是个天才艺术家，跟他合作非常愉快。

在编写本书时参考了很多著作，大部分在各章的“文献注记”中做了说明。还有少数对本书内容和表述有帮助的没有注明，我在这里把它们列举出来，感谢这些作者的帮助。这些资料包括我学生时期读过的两本经典计算机图形学教材：*Computer Graphics: Principles & Practice*（Foley, Van Dam, Feiner, & Hughes, 1990）和*Computer Graphics*（Hearn & Baker, 1986）。其他书包括Alan Watt的两本经典教材（Watt, 1993, 1991）、Hill的*Computer Graphics Using OpenGL*（Francis S. Hill, 2000）、Angel的*Interactive Computer Graphics: A Top-Down Approach With OpenGL*（Angel, 2002）、Hughes Hoppe的华盛顿大学学位论文（Hoppe, 1994）以及Rogers的两本经典图形学教材（D. F. Rogers, 1985, 1989）。

本书是在苹果笔记本电脑上用LaTeX软件写成的。图形是作者用Adobe Illustrator包做出的。感谢创建者开发出这么好用的软件。

感谢犹他大学，使我能够利用学术休假时间完成此书。

特别感谢Alice Peter和Klaus Peter，感谢他们鼓励我写出了本书的第1版，感谢他们高超的编辑技巧，感谢他们精益求精的态度。在第2版中除了纠正公式和语言上的错误外，他们在本书最后阶段的数周内日以继夜地工作，付出了大量心血。毫无疑问，如果没有他们的努力，本书是不可能完成的。

Peter Shirley于盐湖城
2005年4月

目　录

第 1 章　引言 …… 1
1.1　图形学领域 …… 1
1.2　主要应用 …… 1
1.3　图形学 API …… 2
1.4　三维几何模型 …… 2
1.5　图形流水线 …… 3
1.6　数值问题 …… 3
1.7　效率 …… 4
1.8　软件工程 …… 5
1.8.1　单精度与双精度 …… 5
1.8.2　内联 …… 5
1.8.3　成员函数与非成员运算符 …… 6
1.8.4　包含保护符 …… 6
1.8.5　编译调试 …… 7
1.8.6　实验性调试 …… 7
文献注记 …… 7
第 2 章　数学知识 …… 9
2.1　集合与映射 …… 9
2.1.1　逆映射 …… 10
2.1.2　区间 …… 10
2.1.3　对数 …… 11
2.2　解二次方程 …… 12
2.3　三角学 …… 12
2.3.1　角 …… 12
2.3.2　三角函数 …… 13
2.3.3　三角函数公式 …… 14
2.4　向量 …… 15
2.4.1　向量运算 …… 16
2.4.2　向量的笛卡儿坐标 …… 16
2.4.3　点积 …… 17
2.4.4　叉积 …… 18
2.4.5　标准正交基与坐标系 …… 19
2.4.6　从单向量建立正交基 …… 20
2.5　二维隐式曲线 …… 20
2.5.1　二维梯度 …… 21
2.5.2　隐式二维直线 …… 23
2.5.3　隐式二次曲线 …… 25
2.6　二维参数曲线 …… 26
2.6.1　二维参数直线 …… 26
2.6.2　二维参数圆 …… 27
2.7　三维隐式曲面 …… 27
2.7.1　隐式曲面的法向量 …… 27
2.7.2　隐式平面 …… 27
2.7.3　隐式曲面上的三维曲线 …… 28
2.8　三维参数曲线 …… 28
2.9　三维参数曲面 …… 29
2.10　线性插值 …… 30
2.11　三角形 …… 30
2.11.1　二维三角形 …… 30
2.11.2　三维三角形 …… 33
常见问题 …… 34
文献注记 …… 34
习题 …… 34
第 3 章　光栅算法 …… 35
3.1　光栅显像 …… 35
3.2　显示器亮度和 γ 值 …… 36
3.3　RGB 颜色 …… 37
3.4　α 通道 …… 38
3.5　直线绘制 …… 39
3.5.1　基于隐式方程绘制直线 …… 39
3.5.2　基于参数方程绘制直线 …… 42
3.6　三角形光栅化 …… 43

3.7 简单反走样技术……46
3.8 图像捕捉与存储……46
3.8.1 扫描仪和数码摄像机……47
3.8.2 图像存储……47
常见问题……47
习题……48
第 4 章 信号处理……49
4.1 数字音频：一维采样……50
4.2 卷积……51
4.2.1 滑动平均……52
4.2.2 离散卷积……52
4.2.3 把卷积看作移位滤波器之和……55
4.2.4 与连续函数的卷积……56
4.2.5 离散-连续卷积……58
4.2.6 多维卷积……59
4.3 卷积滤波器……61
4.3.1 各种卷积滤波器……61
4.3.2 滤波器的性质……63
4.4 图像信号处理……66
4.4.1 离散图像滤波……66
4.4.2 图像采样中的反走样技术……67
4.4.3 重构与重采样……68
4.5 采样理论……71
4.5.1 傅里叶变换……72
4.5.2 卷积与傅里叶变换……74
4.5.3 傅里叶变换举例……75
4.5.4 采样理论中的狄拉克脉冲……76
4.5.5 采样与走样……76
4.5.6 理想滤波器与实用滤波器……80
习题……81
第 5 章 线性代数……82
5.1 行列式……82
5.2 矩阵……84
5.2.1 矩阵运算……84
5.2.2 矩阵形式的向量运算……86
5.2.3 矩阵与行列式……86
5.2.4 计算逆矩阵……88
5.2.5 线性方程组……89
5.2.6 特征值及矩阵对角化……89
5.2.7 奇异值分解……91
常见问题……91
文献注记……92
习题……92
第 6 章 矩阵变换……93
6.1 基本二维变换……93
6.1.1 缩放……93
6.1.2 切变……94
6.1.3 旋转……95
6.1.4 反射……96
6.1.5 二维变换组合……97
6.1.6 二维变换分解……98
6.2 基本三维变换……101
6.2.1 任意三维旋转……101
6.2.2 法向量变换……102
6.3 平移……103
6.4 变换矩阵的逆……106
6.5 坐标变换……106
常见问题……108
文献注记……108
习题……108
第 7 章 观察……109
7.1 绘制标准视体……109
7.2 正射投影……111
7.3 透视投影……114
7.4 透视变换的性质……118
7.5 视域……119
常见问题……119
文献注记……120
习题……120
第 8 章 隐藏面消除……121
8.1 BSP 树……121
8.1.1 BSP 树算法概述……121
8.1.2 构造 BSP 树……124
8.1.3 分割三角形……126
8.1.4 优化 BSP 树……127
8.2 z 缓冲器……127
8.2.1 z 缓冲器算法……127
8.2.2 整数 z 缓冲器……128
常见问题……129

习题 …… 129

第 9 章 表面明暗处理 …… 130

9.1 漫反射明暗处理 …… 130
9.1.1 朗伯明暗处理模型 …… 130
9.1.2 环境明暗处理 …… 131
9.1.3 基于顶点的漫反射明暗处理 …… 131
9.2 Phong 明暗处理 …… 132
9.2.1 Phong 光照模型 …… 132
9.2.2 表面法线向量插值 …… 134
9.3 艺术化明暗处理 …… 134
9.3.1 线图 …… 134
9.3.2 冷暖明暗处理 …… 135
常见问题 …… 136
习题 …… 136

第 10 章 光线跟踪 …… 137

10.1 基本光线跟踪算法 …… 137
10.2 计算观察光线 …… 138
10.3 光线与物体相交 …… 139
10.3.1 光线与球相交 …… 139
10.3.2 光线与三角形相交 …… 140
10.3.3 光线与多边形相交 …… 142
10.4 光线跟踪程序 …… 142
10.5 阴影 …… 144
10.6 镜面反射 …… 144
10.7 折射 …… 145
10.8 实例化 …… 147
10.9 次线性的光线与物体相交 …… 148
10.9.1 包围盒 …… 149
10.9.2 层次包围盒 …… 151
10.9.3 均匀空间子分法 …… 154
10.9.4 二叉空间划分法 …… 155
10.10 构造实体几何 …… 156
10.11 分布式光线跟踪 …… 156
10.11.1 反走样 …… 157
10.11.2 软阴影 …… 157
10.11.3 景深 …… 159
10.11.4 光泽反射 …… 160
10.11.5 运动模糊 …… 160
常见问题 …… 161
习题 …… 161

第 11 章 纹理映射 …… 162

11.1 三维纹理映射 …… 162
11.1.1 三维条纹纹理 …… 162
11.1.2 纹理数组 …… 163
11.1.3 实体噪声 …… 165
11.1.4 扰动 …… 166
11.2 二维纹理映射 …… 167
11.3 镶嵌模型 …… 168
11.4 光栅化三角形的纹理映射 …… 169
11.5 凹凸纹理 …… 172
11.6 置换映射 …… 172
11.7 环境映照 …… 173
11.8 阴影图 …… 174
常见问题 …… 174
文献注记 …… 175
习题 …… 175

第 12 章 完整的图形流水线 …… 176

12.1 裁剪 …… 176
12.2 流水线中裁剪部分的位置 …… 176
12.2.1 在透视除法后的裁剪（选项 3） …… 177
12.2.2 在变换前裁剪（选项 1） …… 178
12.2.3 在齐次坐标系中裁剪（选项 2） …… 179
12.2.4 用平面裁剪 …… 179
12.3 扩展的图形流水线 …… 180
12.3.1 剔除 …… 180
12.3.2 光照和明暗处理 …… 180
12.4 背向面删除 …… 180
12.5 三角形条带和三角形扇区 …… 180
12.6 保留状态 …… 181
12.7 完整的图形流水线 …… 181
常见问题 …… 182
文献注记 …… 182
习题 …… 182

第 13 章 图形学的数据结构 …… 183

13.1 三角形网格 …… 183
13.2 翼边数据结构 …… 183
13.3 场景图 …… 184
13.4 平铺多维数组 …… 186

13.4.1 二维数组的一层平铺……187
13.4.2 示例：三维数组的两层平铺…188
常见问题……189
文献注记……189
习题……189

第 14 章 采样……190
14.1 积分……190
14.1.1 度量和均值……191
14.1.2 示例：二维平面中直线的度量……191
14.1.3 示例：三维空间中直线的度量……193
14.2 连续概率……194
14.2.1 一维连续概率密度函数……194
14.2.2 一维期望……194
14.2.3 多维随机变量……194
14.2.4 方差……195
14.2.5 估计均值……196
14.3 蒙特卡罗积分……196
14.4 选择随机点……197
14.4.1 求反函数……198
14.4.2 拒绝……200
14.4.3 Metropolis……200
14.4.4 示例：在正方形上选择随机直线……201
常见问题……203
文献注记……204
习题……204

第 15 章 曲线……205
15.1 曲线……205
15.1.1 参数化和重新参数化……206
15.1.2 分段的参数表达式……207
15.1.3 样条……208
15.2 曲线的性质……209
15.3 多项式片段……210
15.3.1 多项式的表示……210
15.3.2 线段……211
15.3.3 超线段……212
15.3.4 三次方程的基矩阵……214
15.3.5 混合函数……214
15.3.6 插值多项式……214
15.4 片段连接……215
15.4.1 节点……215
15.4.2 使用独立的片段……216
15.4.3 连接线段……216
15.5 三次多项式……217
15.5.1 自然三次多项式……218
15.5.2 埃尔米特三次多项式……219
15.5.3 基数三次多项式……219
15.6 拟合曲线……221
15.6.1 贝塞尔曲线……222
15.6.2 B 样条曲线……226
15.6.3 非均匀 B 样条曲线……230
15.6.4 非均匀有理 B 样条曲线……232
15.7 小结……233
文献注记……233
习题……233

第 16 章 计算机动画……235
16.1 动画原理……236
16.1.1 时间控制……236
16.1.2 动作布局……236
16.1.3 动画技术……237
16.1.4 人工控制与自动化方法……238
16.2 关键帧……238
16.2.1 运动控制……240
16.2.2 旋转插值……242
16.3 变形……244
16.4 角色动画……245
16.4.1 面部动画……249
16.4.2 运动捕捉……250
16.5 基于物理的动画……251
16.6 过程技术……253
16.7 对象群……255
文献注记……257

第 17 章 使用图形硬件……258
17.1 什么是图形硬件……258
17.2 从硬件角度描述几何模型……259
17.3 几何模型转化为像素……263
17.3.1 设计流水线……264
17.3.2 基本执行模型……265

17.3.3　顶点着色器示例……267
17.3.4　片段着色器示例……269
17.3.5　GPU 上的通用计算……270
常见问题……271
文献注记……271
习题……271

第 18 章　构建交互式图形应用程序……272

18.1　击球程序……272
18.2　编程模型……274
18.2.1　控制驱动的编程……274
18.2.2　事件驱动的编程……276
18.2.3　事件驱动的击球程序……280
18.2.4　实现文献注记……282
18.2.5　小结……284
18.3　模型-视图-控制器体系结构……284
18.3.1　模型-视图-控制器框架……284
18.3.2　将 MVC 应用于击球程序……285
18.3.3　利用 MVC 扩充击球程序……289
18.3.4　MVC 组件之间的交互作用……290
18.3.5　应用 MVC 概念……291
18.4　实现示例……291
18.4.1　使用 GUI API……292
18.4.2　使用图形 API……293
18.4.3　实现细节……294
18.5　结果的运用……298
18.5.1　示例 1：PowerPoint……298
18.5.2　示例 2：Maya……299
文献注记……300
习题……300

第 19 章　光照……302

19.1　辐射测量……302
19.1.1　光子……302
19.1.2　光谱能量……302
19.1.3　功率……303
19.1.4　辐射照度……304
19.1.5　辐射率……304
19.1.6　BRDF……306
19.2　输送方程……307
19.3　光度测定……308
常见问题……309
文献注记……309
习题……309

第 20 章　颜色……310

20.1　光和光检测器……310
20.2　三刺激颜色理论……311
20.3　CIE 三刺激值……312
20.4　色度……312
20.5　暗视亮度……313
20.6　RGB 显示器……314
20.7　近似颜色操作……314
20.8　对立颜色空间……315
常见问题……316
文献注记……316
习题……316

第 21 章　视觉感知……317

21.1　视觉科学……317
21.2　视觉灵敏度……318
21.2.1　亮度和对比度……318
21.2.2　颜色……321
21.2.3　动态范围……323
21.2.4　视域和灵敏度……324
21.2.5　运动……326
21.3　空间视觉……328
21.3.1　参考坐标系和测量尺度……328
21.3.2　眼球运动线索……329
21.3.3　双目视差……330
21.3.4　运动线索……331
21.3.5　绘画线索……333
21.4　物体、位置和事件……336
21.4.1　物体识别……336
21.4.2　大小与距离……338
21.4.3　事件……340
21.5　图片感知……341

第 22 章　色调再现……343

22.1　分类……345
22.2　动态范围……346
22.3　颜色……346
22.4　图像形成……348
22.5　基于频率的算子……348

22.6 梯度域算子……349
22.7 空间算子……350
22.8 除法……351
22.9 S 形曲线……352
22.10 其他方法……356
22.11 夜景色调映射……357
22.12 讨论……358

第 23 章 全局照明……359

23.1 朗伯场景的粒子跟踪……359
23.2 路径跟踪……362
23.3 精确的直接光照……363
23.3.1 数学框架……363
23.3.2 球形照明器的采样……364
23.3.3 非漫射照明器……366
常见问题……366
文献注记……367
习题……368

第 24 章 反射模型……369

24.1 现实世界的材质……369
24.1.1 光滑绝缘材质和金属……369
24.1.2 粗糙表面……370
24.1.3 散射材质……370
24.1.4 透明材质……370
24.1.5 分层材质……370
24.2 反射模型的实现……371
24.3 镜面反射模型……372
24.4 光滑分层模型……372
24.5 粗糙分层模型……374
24.5.1 各向异性的镜面 BRDF……375
24.5.2 各向异性 Phong 模型的漫射项……375
24.5.3 模型的实现……376
常见问题……377
文献注记……377
习题……377

第 25 章 基于图像的绘制……378

25.1 光场……378
25.2 根据一组图像创建一幅新图像……379
常见问题……380
习题……381

第 26 章 可视化……382

26.1 二维标量场……382
26.2 三维标量场……383
26.2.1 等值面……383
26.2.2 直接体绘制……386
常见问题……389
习题……389

参考文献……390

索引……401

第1章 引 言

计算机图形学是利用计算机技术创建和处理图像的理论、方法和技术。本书采取了比较底层的视角，主要讲解图形生成算法。学习计算机图形学，需要知道一些关于具体硬件、文件格式以及一两种API[1]方面的知识。这些知识将随着相应领域的快速发展而变化，因此本书不涉及其中的具体细节。建议读者阅读软硬件环境的有关文档，把它们作为本书的补充材料。所幸计算机图形学已经形成了比较多的标准术语和概念，因此本书讨论的内容适用于多种环境。本章将定义一些基本术语，介绍一些历史背景以及有关计算机图形学方面的信息来源。

1.1 图形学领域

对任何领域进行分类都不容易，但是多数从事图形学研究的专业人士会认同以下几大方向，并认为它们同属计算机图形学的范畴：

1

- **建模**。对形状和外观性质进行数学定义，并且能够存储在计算机中。例如，一只咖啡杯可以表示为有序的三维点集，这些三维点按一定的插值规则连接起来，还可以建立咖啡杯受光线照射的反射模型。
- **绘制（rendering）**。该术语来自艺术领域，根据三维计算机模型生成带阴影的图像。
- **动画**。利用图像序列，产生物体运动的视觉效果的一种技术。这里也要用到建模和绘制，但控制时间这个关键问题在基本建模和绘制中一般是不考虑的。

还有很多用到计算机图形学的领域，它们是否属于图形学范畴还存在争议，但至少是相近的研究领域。这些相近的研究领域如下：

- **人机交互**。在输入设备（如鼠标、写字板等）、应用程序和以图形或者其他感官方式向用户发送的反馈之间建立接口。历史上，该领域和图形学关系非常密切，因为图形学研究人员最早接触这些输入/输出设备，不过现在这些设备已经随处可见了。
- **虚拟现实**。试图让用户置身于三维虚拟世界。为了产生这种效果，系统至少有立体显示，并对头部运动产生反应。真正的虚拟现实系统还要有声音反馈和力反馈。由于这一领域需要先进的图形技术和显示技术，因此与图形学的关系通常很密切。
- **可视化**。试图通过视觉显示让用户看得更明白。可视化问题一般都会涉及图形学。
- **图像处理**。对二维图像进行处理，图形学和视觉领域都要用到。
- **三维扫描**。利用测距技术建立三维测量模型。这种模型在创建细节丰富的视觉图像方面很有用处，而模型处理常常要用到图形学算法。

1.2 主要应用

几乎各行各业都多少要用到计算机图形学，但主要用户涉及以下行业：

2

1. 应用程序接口（API）是实现画线等基本操作的软件接口，当前较流行的API包括OpenGL、Direct3D和Java3D。

- **视频游戏**。视频游戏越来越多地使用复杂的三维模型和绘制算法。
- **动画片**。动画片一般直接根据三维模型生成。很多传统的二维动画片采用由三维模型产生的背景，并允许其中有连续运动的视点，这样可以为动画片制作者节省大量时间。
- **电影特效**。电影特效几乎用到了计算机图形学技术的所有方面。现代电影几乎都用数字合成技术，在拍好的前景中添加背景。还有很多电影根据三维模型通过计算机生成前景。
- **CAD/CAM**。表示计算机辅助设计与计算机辅助制造。该行业利用计算机技术在计算机内设计零件和产品，然后用这些虚拟设计指导生产过程。例如，先用三维计算机建模软件包设计出机械零件，然后由计算机控制的生产设备自动生产出零件。
- **仿真**。可以认为仿真是精确的视频游戏。例如，飞行仿真器使用复杂的三维图形学技术模拟一架飞机的飞行过程。在一些安全性极为重要的领域（如驾驶）的初期训练中，这种仿真技术非常有用。还可用于专业训练，如特殊情况下的消防演习，如果建立实际演习场景则花费太大，或者太危险。
- **医学成像**。根据病人的检查数据创建有意义的图像。例如，核磁共振成像数据按密度值的三维矩形阵列存储。可利用计算机图形学创建有明暗的图像，以帮助医生从数据中找出有用的信息。
- **信息可视化**。那些本来不可视的数据，可以将它们以图像的方式显示出来。例如，10种股票价格随时间的变化趋势，本来没有明显的视觉描述，但采用图形化技术就可以帮助人们在数据中找到规律。

1.3　图形学 API

3

使用图形函数库的关键在于处理应用程序接口（API）。API是一种软件接口，它为应用程序如何访问系统功能（例如在窗口中画图）提供了一种模型。一般来说，在设计图形程序时存在的两个关键问题是，处理如同“画三角形”这样的图形调用，以及处理如按下一个按钮这样的用户交互。

多数API都具有使用回调（callback）的用户界面工具包。回调指通过函数指针或者虚函数向低层函数传递参数的过程。例如，为了将一个行为与按下按钮联系起来，一个低层函数与按钮按下进行动态关联。按这种方式，用户界面工具包就可以处理按钮按下这一事件，程序员可以让它与任何行为相关联。

当前主要有两种API模式。第一种是Java的集成方法，其中图形与用户界面工具包被集成在一起，而且都是可移植的包（package），这样的包是完全标准化的，并作为语言的一部分得到支持。第二种的代表是Direct3D和OpenGL，其中画图命令是软件库的一部分，软件库与某种语言（如C++语言）绑定，而用户界面软件是独立的实体，一般随系统不同而不同。在后一种方法中，编写可移植的代码并不容易，但简单程序可以在特定系统的事件处理之上利用可移植库。

不管选用哪种API，基本图形调用大都一样，本书的概念都适用这些调用。

1.4　三维几何模型

图形程序的关键是使用三维几何模型。这些模型采用球、立方体、圆锥、多边形等基本数学形状来描述三维物体。最常用的模型由三维三角形组成，这些三角形共享顶点，常被称作三角形网格。这些网格模型有时由美工设计人员用交互式建模程序生成，有时要借用距离扫描设备得到数据。不管哪种情况，这些模型一般都包含很多小三角形，所以，为了适应这种数据集，要对程

序进行优化处理。

1.5 图形流水线

现在的计算机几乎都有三维图形流水线（graphic pipline）。这是一个特别的软硬件子系统， 4
能高效绘出透视图中的三维图元。一般来说这些系统都经过优化，以适应处理三维带有共享顶点的三角形。流水线基本运算是把三维顶点位置映射到二维屏幕位置，并对三角形进行明暗处理，使它们看起来比较真实并且按适当的前后层次显示。

尽管具有前后层次画三角形的方法一度是计算机图形学领域的研究重点，但是现在这个问题一般都通过z缓冲器技术解决。z缓冲器用特殊的存储缓冲区解决了蛮力算法中存在的问题。

图形流水线中的几何运算，在四维坐标空间几乎都可以实现。四维坐标由一般的三维几何坐标和利于处理透视问题的第4维齐次坐标组成。这些四维坐标主要用于4×4的矩阵和四维向量。因此图形流水线中包含很多对矩阵和向量进行高效处理和组合运算的机制。该四维坐标系是计算机科学中最精妙、最漂亮的构造之一，当然也是学习计算机图形学遇到的最大智力挑战。每本图形学书籍的第一部分都有相当多的内容是讨论坐标系的。

现在多数图形流水线的速度都与正在画的三角形的数量有关。因为交互能力要比视觉效果更重要，所以在生成模型的过程中需要将三角形数量减到最少。另外，如果模型出现在远处，所需要的三角形数量就比模型在近处时少。这就意味着，在表示模型时要采用不同的细节等级（LOD）。

1.6 数值问题

很多图形学程序实际就是三维数值代码。在这类程序中，数值问题常常是非常重要的问题。在过去，用一种稳健并且可移植的方式处理这种问题很困难，因为机器之间有不同的内部数字表示方法，更糟糕的是处理异常的方式不兼容。幸运的是，现代计算机几乎都服从IEEE浮点标准（IEEE Standards Association，1985），这就为编程者处理数值问题带来了方便。

当对数值算法编码时，IEEE浮点表示具有很多重要特性，但图形学多数情况下只使用其中少 5
数几个特性。首先也是最重要的一点，就是了解IEEE浮点实数中有三个“特殊值”：

- **无穷大(∞)**：比任何其他有效数都大的有效数。
- **负无穷大(-∞)**：比任何其他有效数都小的有效数。
- **无效数(NaN)**：这是一个无效数，由结果未定义的运算得出，例如 0 除以 0。

IEEE浮点标准的制定者做了一些规定，这对编程者来说非常方便。很多规定在处理异常运算如0为除数时，都与上面的三个特殊值有关。这时异常被记载在日志中，不过很多情况编程者都可以忽略不看。对任意正实数a，下面的规则成立：

$$+a/(+\infty) = +0$$
$$-a/(+\infty) = -0$$
$$+a/(-\infty) = -0$$
$$-a/(-\infty) = +0$$

要注意一点，IEEE浮点标准将-0与+0区别开来。在多数图形学程序中，这种区别关系不大，但对于更经典的数值算法，应该牢记这一点。

其他涉及无穷大的运算，与期望的结果一样。对于正实数a，有

$$\infty+\infty=+\infty$$
$$\infty-\infty=\text{NaN}$$
$$\infty\times\infty=\infty$$
$$\infty/\infty=\text{NaN}$$
$$\infty/a=\infty$$
$$\infty/0=\infty$$
6
$$0/0=\text{NaN}$$

涉及无穷大的布尔表达式规则如下，与期望的结果一样：

(1) 所有有限的有效数都小于$+\infty$。

(2) 所有有限的有效数都大于$-\infty$。

(3) $-\infty$小于$+\infty$。

包含无效数（NaN）的表达式规则很简单，具体如下：

(1) 任何包含NaN的算术表达式，结果是NaN。

(2) 任何包含NaN的布尔表达式，结果为假。

也许IEEE浮点标准最实用的是对除数为0的情况的处理。对于任意的正实数a，下面包含除数为0的规则成立：

$$+a/+0=+\infty$$
$$-a/+0=-\infty$$

如果在代码中出现-0，就必须要小心了。但是如果编程者利用IEEE浮点规则，很多数值代码就会变得非常简单了。例如，考虑表达式

$$a=\frac{1}{\frac{1}{b}+\frac{1}{c}}$$

这种表达式出现在电阻和透镜中。如果除以0导致程序崩溃（在IEEE浮点标准出现之前，很多系统会出现这种情况），那么就需要在代码中加入两个*if*语句，来判断b或c是否为0或者非常小。有了IEEE浮点标准，如果b或c为0，我们将得到a为0的结果，这正是我们想要的。另一种避免特别检查的通用技术是，利用NaN的布尔特性。考虑下面的代码段：

```
a = f(x)
if(a>0)then
  do something
```

7 其中，函数f可能会返回类似∞或者NaN这样的值。当$a=\text{NaN}$或者$a=-\infty$时，*if*语句为假；当$a=+\infty$时，*if*语句为真，因此不需要进行特别检查。这样就使程序更加简单，更加稳健，并且更加高效。

1.7 效率

没有能使代码更高效的超级规则。高效率是对各方面进行认真权衡后得到的，而权衡的标准随系统体系结构的不同而不同。在不远的将来，一个好的程序员应该把更多心思放在存取方式上，而不是数值运算上。而10年前的情况正好相反。出现这种情况的原因是，内存速度的提高跟不上处理器速度的提高。随着这种趋势的继续，优化存取方式显得越来越重要。

一种优化代码效率的方法如下，从中挑选需要的步骤进行：

(1) 以最直接的方式编写代码。尽量直接计算数据而不存储它。

(2) 在优化模式下编译。

(3) 使用性能分析软件找到系统瓶颈。

(4) 检查数据结构，找出调整数据位置的方法。如果可能，使数据单元大小和目标体系结构的高速缓存/页面大小匹配。

(5) 如果性能分析表明数值计算中存在瓶颈，则检查编译器生成的汇编代码，找出效率低的原因。重写源代码以解决发现的任何问题。

这些步骤中最重要的是第一步。很多“优化”使代码可读性更差却不能提高效率。另外，把时间花在预先的代码优化上要比纠正bug或增加功能更好。另外，在读旧的文献时要谨慎。因为过去的诀窍，如使用整数代替实数，已经不可取了，因为现在的CPU可以按照与处理整数同样的速度处理浮点数。在所有情况下，在具体机器和编译器上优化后，都要经过性能分析，以确定优化有没有效果。

1.8 软件工程

任何图形程序的关键部分是针对几何实体（如向量和矩阵）以及图形实体（如RGB颜色和图像）都具有比较好的类或者例程。这些例程应尽可能简洁和高效。多数图形编程人员都使用 8
C++，因此应该讨论一下C++。这里有一个重要的问题，位置与位移是否应该写在不同的C++类中，因为它们的操作是不同的，例如位置乘以1/2是没有几何意义的，而位移乘以1/2是有意义的（Goldman，1985）。对此各人看法不同，但我始终坚信KISS（keep it simple，stupid）原则，根据这一原则采用两个类的好处还不足以抵消增加的复杂性[相反的观点见（DeRose，1989）]。这说明应该编写的基本类包括：

- vector2：二维向量类，存储了x和y分量。它将这些分量存储在长度为2的数组中，以支持索引操作。同时应该包括向量加法、向量减法、点积、叉积、标量乘法和标量除法。
- vector3：三维向量类，与vector2类似。
- hvector：含4个分量的齐次向量（见第7章）。
- rgb：含3个分量的RGB颜色。同时应该包括RGB加法、RGB减法、RGB乘法、标量乘法和标量除法。
- transform：4×4变换矩阵。应该包括矩阵乘法，以及求位置、方向与表面法向量的成员函数。像第6章所讲的那样，这些都是不同的操作。
- image：由RGB像素构成的二维阵列，带一个输出操作。

此外，你可以添加区间、标准正交基、坐标系等类，也可以不加。也可以考虑单位长度向量，但我发现这样做得不偿失。在后面几节中列出了一些基本结论。

1.8.1 单精度与双精度

现代体系结构提出，减少内存使用并保持一致的内存存取是提高效率的关键。这表明应使用单精度类型。但是为了避免数值问题，应该采用双精度算法。对二者的权衡取决于实际程序，不过在类定义中设置默认值是个好习惯。我建议对几何运算使用双精度类型，而对颜色运算使用单 9
精度类型。当内存使用率较高（用于如三角形网格模型）时，建议存储成单精度数据，但如果是通过成员函数存取数据，则应转换成双精度类型。

1.8.2 内联

内联是实用类（如RGB）提高效率的关键。几乎所有的RGB和向量函数都应该是内联的。要

对代码进行性能分析，以保证内联确实正确。非实用代码及其他大的函数不应该是内联的，除非性能分析器表明它们占用很多运行时间。即使后来才将它们定义为内联，也不会影响代码的运行速度。注意，在多数系统中，内联函数必须定义在头文件中。对于成员函数，则可以在声明的同时进行定义，例如：

```
class vector3 {
      .
      .
      .
    double lengthSquared { return x()*x()+y()*y()
        +z()*z(); }
};
```

1.8.3　成员函数与非成员运算符

对于两向量相加这样的运算，我们或者将它们声明为向量类的成员，或者是类外部的运算函数。我建议把类似这样的运算声明在类外部。因为对于类似用双精度数乘以向量（与向量乘以双精度数相对）这样的运算，只能是声明在类外部。既然存在必须是非成员的情况，我们就干脆将运算函数一直声明为非成员，以保持一致。定义这种运算时，代码越简洁越好。例如：

```
inline vector3 operator+(vector3 a, vector3 b) {
    return vector3( a.x() + b.x(), a.y()
        + b.y(), a.z() + b.z() );
}
```

注意，对于非内联运算函数以及一些编译器，参数传递采用const引用形式，可以避免一些
10 数据复制：

```
inline vector3 operator+(const vector3& a, const vector3& b) {
    return vector3( a.x() + b.x(), a.y()
        + b.y(), a.z() + b.z() );
}
```

1.8.4　包含保护符

所有类声明都应该具有包含保护符（include guard）。包含保护符的名字应遵从一些简单的命名约定。例如：

```
#ifndef VECTOR3H
#define VECTOR3H

class vector3 {
      .
      .
      .

};

#endif
```

多次包含头文件，这在实际中几乎是不可避免的。增加包含保护符，就可以避免多次包含头文件的问题。注意，当VECTOR3H已经定义时，仍会打开头文件并读取一行。对于大型类库，这个打开的文件要占用编译时间（Lakos，1996）。在这种情况下，有一种不太好但有效的解决办法是，在包含文件时进行一次检查：

```
#ifndef VECTOR3H
#include <vector3.h>
#endif
```

1.8.5 编译调试

在代码中应该包含很多断言（assert）宏。断言是一种宏，如果它包含的布尔表达式的结果为假，则终止程序。例如：

```
#include <assert.h>
    .
assert( fabs( v.length() - 1 ) < 0.00001 );
```

11

上面的代码用来判断v与单位长度是否接近。在程序开发与调试过程中，应该多使用断言。如果你曾经用过，就让它留在代码中。以后再出现bug时，它就会起作用。注意在优化运行中，需要定义预处理变量NDEBUG来关闭断言，这个功能一般与编译器标志-DNDEBUG一起完成。当在调试模式下进行编译时，变量可以设置为非法值，如NaN，这样当使用没有初始化的变量时会使程序崩溃。

1.8.6 实验性调试

四处问问就会发现，随着编程者越来越有经验，他们使用传统调试器的次数越来越少。原因之一是，用这类调试器调试复杂程序比调试简单程序更麻烦。原因之二是，最难发现的错误都是概念上的错误，即所实现的目标就错了，而采取步进运行观察变量值的调试方法经常会浪费大量时间，而且检测不出这种错误。

在图形学程序中，有另一种非常实用的调试方法可供选择。不过这种方法也有弊端，有点类似于计算机编程者刚开始工作时就被告诫不要做一些事情，如果这么做了就会感到没守规矩。这种方法就是：构建一幅图像，并观察有什么问题；然后，针对产生问题的可能原因做出一种假设，并进行验证。例如，在一个光线跟踪程序中，可能会产生很多随机的黑点（这是典型的“暗疮”问题，编写光线跟踪程序时经常会遇到这种问题）。这种情况下传统调试器就不起作用了，应该意识到阴影线正影响着被遮住的表面。我们可能会注意到黑点的颜色是周围环境的颜色，所以直接光照正是丢失的部分。直接光照在阴影中可能被中止了，因此可以假设这些点被错误地标识在阴影中。为了检验这一假设，可以停止遮挡检查并重新编译。结果可能表明这些是错误的阴影测试，这样我们可以继续进行检测。这种方法有时很实用的关键原因在于，我们不是必须找到错误值，或者真正确定是概念错误。相反，我们只是通过实验缩小概念错误的范围。一般仅需几次尝试就可以找到问题，这种调试是一种享受。

在程序崩溃时，用传统调试器可以查出崩溃的位置。然后再从头开始，用断言并重新编译，找出程序出错的位置。这些断言应该留在程序中，以检测将来可能会带入程序的错误。这再次避免了传统的步进调试过程，因为步进调试不会在程序中使用便利的断言。

12

文献注记

关于软件工程的讨论，参考了*Effective C++*系列（Meyers，1995，1997）、*Extreme Programming*运动（Beck & Andres, 2004）以及（Kernighan & Pike, 1999）的思想。实验性调试部分的内容基于与Steve Parker的讨论。有很多关于计算机图形学的年度会议，通过会议名称可以在网上搜到相关信息：

- ACM SIGGRAPH Conference。
- Graphics Interface Conference。
- Game Developers' Conference (GDC)。

- Eurographics Conference。
- Pacific Graphics Conference。
- Eurographics Symposium on Rendering。
- Solid Modeling Conference。
- IEEE Visualization Conference。

13

第2章 数学知识

图形学很大程度上就直接把数学表达式转换成代码。数学表达得越清晰，对应的代码就越容易理解。因此，本书很大程度上就在集中讨论使用恰当的数学知识实现各种目的。本章介绍我们在中学和大学数学中学过的各种数学知识，更多的是作为参考而不是指南。本章涉及的内容很杂。之所以提到这些内容，可能是因为它们在标准数学课程中比较少见，或者因为在图形学中占有举足轻重的地位，也可能因为在数学课中对它们的几何意义解释不够。除了对书中要用到的一些表示法进行说明外，本章加强了数学课程中跳过不讲的一些内容，例如三角形的重心坐标等。本章强调对具体内容进行直观和几何意义的解释，而不强调严格的数学推导过程。线性代数推迟到第5章介绍，也就是安排在矩阵变换之前。建议读者略读本章以大体熟悉一下各个标题，需要时再进行参考。本章后面的习题作为有关内容的补充练习。

2.1 集合与映射

映射，又称为函数，是数学及编程的基础。和程序中的函数类似，数学中的映射把属于某类型的自变量作为参数，映射为（返回）某特殊类型的对象。在程序中我们称“类型”，在数学中我们称集合。对象是集合的成员，我们用符号“$\in$”来表示这种关系，即：

$$a \in S$$

上式读作：a是集合S的一个成员。给定任意两个集合A和B，求它们的笛卡儿积我们可以得到第三个集合，记为$A \times B$。集合$A \times B$由所有可能的有序对(a, b)组成，其中$a \in A$，$b \in B$。为简便起见，我们用符号A^2表示$A \times A$。对笛卡儿积进行扩展，对于给定的3个集合，它们的成员可以构成有序三元组的集合。同样，对于给定的任意多个集合，可以构成有序任意元组的集合。

经常用到的集合包括：

- $\mathbb{R}$：实数集合。
- $\mathbb{R}^+$：非负实数集合（包括0）。
- $\mathbb{R}^2$：二维平面上的实数有序对集合。
- $\mathbb{R}^n$：n维笛卡儿空间中的点集合。
- $\mathbb{Z}$：整数集合。
- S^2：单位球面上的三维点集合（$\mathbb{R}^3$中的点）。

注意，尽管S^2由三维空间中的点组成，但由于它们位于可用两个变量参数表示的表面上，因此可以看作是二维集合。用箭头和冒号表示映射，即

$$f: \mathbb{R} \mapsto \mathbb{Z}$$

上式读作：存在一个函数f，可以将输入的实数映射成一个整数。其中，箭头左侧的集合称为函数的定义域，箭头右侧的集合称为目标。包含所有函数映像点的目标子集（即在$\mathbb{R}$中存在对应点的$\mathbb{Z}$中的点集），称为函数的值域。编程人员更喜欢这样的等价描述：存在一个函数f，它具有一

个实型参数，并返回一个整型数。也就是说，上面的集合表示法等价于通常的编程表示法：

$$\text{integer}\, f(\text{real}) \leftarrow \text{等价于} \rightarrow f:\mathbb{R}\mapsto\mathbb{Z}$$

[16] 因此，可以认为“冒号-箭头”表示法是编程语法，就这么简单。

2.1.1 逆映射

假设有一个函数 $f:A\mapsto B$，则可能存在一个反函数 $f^{-1}:B\mapsto A$。反函数的定义规则为：$f^{-1}(b)=a$，其中 $b=f(a)$。只有当每个 $b\in B$ 是函数 f 作用下的映像点，并且只存在唯一的点 a 使得 $f(a)=b$ 成立时，这样的反函数定义才有意义。这种映射或者函数称为双射。双射将每个 $a\in A$ 映射成唯一的 $b\in B$，并且对于每个 $b\in B$，只存在一个 $a\in A$ 使得 $f(a)=b$ 成立（见图2-1）。骑手与马之间的双射，表示每人只能骑一匹马，并且每匹马都有人骑。这两个函数可以表示为 *rider*(*horse*) 和 *horse*(*rider*)，它们互为反函数。不是双射的函数不存在反函数（见图2-2）。

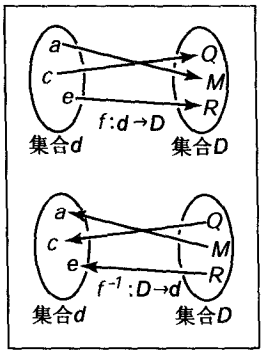


图2-1 双射 f 及其反函数 f^{-1}。注意 f^{-1} 也是双射

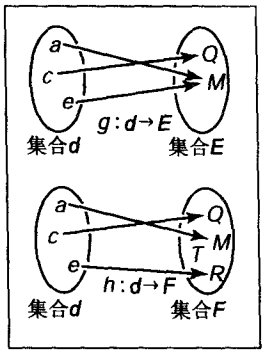


图2-2 函数 g 没有反函数，因为 d 中有两个元素映射到 E 中的同一个元素。函数 h 同样没有反函数，因为集合 F 中的元素 T 在集合 d 中没有对应的元素

双射举例。$f:\mathbb{R}\mapsto\mathbb{R}$，其中 $f(x)=x^3$。它的反函数是 $f^{-1}(x)=\sqrt[3]{x}$。从这个例子可以看出，数学表示符号有时会出现问题，因为 x 同时作为 f 以及 f^{-1} 的哑变量。更直观的方法是采用不同的哑变量：$y=f(x)$，$x=f^{-1}(y)$。这时就可以得到更直观的表达方式：$y=x^3$，$x=\sqrt[3]{y}$。平方函数 $sqr:\mathbb{R}\mapsto\mathbb{R}$ 不存在反函数，其中 $sqr(x)=x^2$。原因有两个，首先 $x^2=(-x)^2$，其次定义域中不存在映射为负数的成员。但是，如果把定义域和值域都限制到 $\mathbb{R}^+$，则可以定义反函数。$\sqrt{x}$ 就是有效的反函数。

2.1.2 区间

一般要对函数处理的实数范围进行规定，也就是要对区间进行定义。例如0和1之间、但不包括0和1的实数区间，表示为（0,1）。因为它不包括端点，因此称为开区间。相应的闭区间用方括号表示：[0,1]。也可以混合使用圆括号和方括号，如[0,1)表示包含0但不包含1的区间。我们假定当区间写为[*a*,*b*]时，就意味着 $a\leqslant b$。表示区间的三种常见方法如图2-3所示。经常要用到区间的笛卡儿积。例如，三维单位立方体内的点 x 可表示为 $x\in[0,1]^3$。

区间在集合运算中特别有用。集合运算包括交、并、差等。例如两个区间的交集等于它们共同拥有的点的集合。符号∩表示交运算，如[3,5)∩[4,6]=[4,5)。符号∪表示并运算，表示两个区间

[17] 所有元素组成的集合，如[3,5)∪[4,6]=[3,6]。差运算的结果与参数的顺序有关，这一点与交、并两种运算不同。差运算符用负号（即“-”）表示，结果返回左边区间中不属于右边区间的那些点。例如，[3,5) - [4,6]=[3,4)，[4,6] - [3,5)=[5,6]。这些运算可以很容易地用区间图形象表示（见图2-4）。

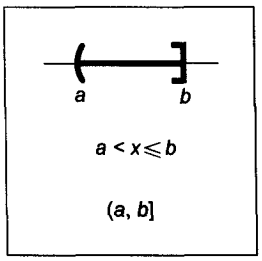


图2-3 三种表示从a到b且含b不含a的区间表示方法

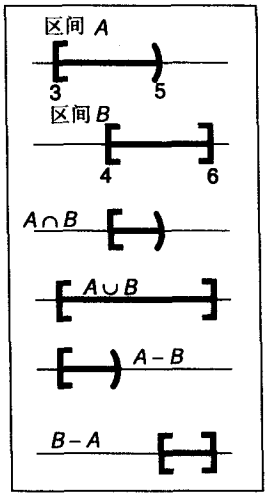


图2-4 区间[3,5)和[4,6]之间的运算

2.1.3 对数

计算器出现之前人们经常使用对数，而现在对数使用虽然已经不那么普遍了，但是在解决有指数项的方程时，对数还是非常有用的。根据定义，每种对数都有一个底数a。以a为底的x的对数可以写为$\log_a x$，定义为“为得到x，a应具有的次幂”，即

$$y = \log_a x \Leftrightarrow a^y = x$$

注意，以a为底的对数与a的指数函数互逆。从该基本定义可以得到以下几个推论：

$$a^{\log_a(x)} = x$$

$$\log_a(a^x) = x$$

$$\log_a(xy) = \log_a x + \log_a y$$

$$\log_a(x/y) = \log_a x - \log_a y$$

$$\log_a x = \log_a b \ \log_b x$$

进行对数运算时，数字e=2.718…用处很大。以e为底的对数称为自然对数。自然对数使用频繁，常采用ln来表示：

$$\ln x \equiv \log_e x$$

其中符号“≡”读作“恒等于”。与π一样，e在文中会出现很多次。除了e，很多领域都采用特殊的底数进行运算，并且在表达式中省略了对数的底，如$\log x$。例如，天文学家经常使用以10为底的对数，理论计算机科学家经常使用以2为底的对数。由于计算机图形学涉及很多领域的知识，我们将避免出现这种简写的情况。

对数和指数的导数可以说明自然对数为什么被称为“自然的”：

$$\frac{\mathrm{d}}{\mathrm{d}x}\log_a x = \frac{1}{x\ln a}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}a^x = a^x \ln a$$

当a=e时，上面式子中的常数项$\ln a$就变成了1。

2.2 解二次方程

二次方程具有下列形式：

$$Ax^2 + Bx + C = 0$$

其中x是一个未知的实数，A、B和C是已知的常数。如果将$y = Ax^2 + Bx + C$想像为二维平面上的xy曲线，那么方程的解就是曲线y值过零点的x坐标。$y = Ax^2 + Bx + C$是一条抛物线，它与x轴的交点个数可能是0、1或2个，由此可以确定方程的实数解有0、1或2个（见图2-5）。

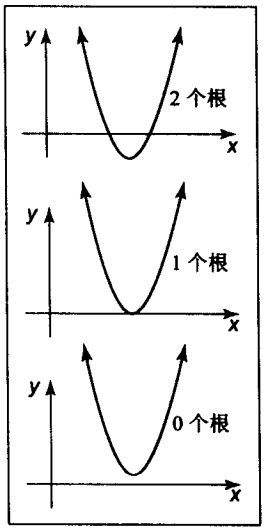


图2-5　二次方程的根的几何解释是抛物线与x轴的交点

为了得到二次方程的解析解，首先在方程两边同除以A：

$$x^2 + \frac{B}{A}x + \frac{C}{A} = 0$$

然后进行配方，得到：

$$\left(x + \frac{B}{2A}\right)^2 - \frac{B^2}{4A^2} + \frac{C}{A} = 0$$

将常数项移到方程的右边，两边进行开平方：

$$x + \frac{B}{2A} = \pm\sqrt{\frac{B^2}{4A^2} - \frac{C}{A}}$$

两边同时减去$B/(2A)$，然后以$2A$做分母进行通分就得到我们常见的形式：

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{2-1}$$

其中“±”表明有正负两个解。于是3±1等于“2或4”。注意决定实数解个数的式子是：

$$D \equiv B^2 - 4AC$$

此式称为二次方程的判别式。如果$D > 0$，则方程有两个实数解（又称为根）。如果$D = 0$，则有一个实数解（一个重根）。如果$D < 0$，则方程没有实数解。

例如，方程$2x^2 + 6x + 4 = 0$的根为$x = -1$和$x = -2$，而方程$x^2 + x + 1 = 0$没有实数解。这两个方程的判别式的值分别为$D = 4$和$D = -3$，根据其正负就可以确定解的个数。编程时，应该先判断D的值，如果D为负就返回“没有根”，而不再计算平方根。

2.3 三角学

图形学中的很多情况要用到基本的三角知识。三角知识可以帮助我们记忆基本定义。

2.3.1 角

尽管我们对“角”已经有了一点初步认识，还是来看看如何进行角的定义，以进一步将角的概念推广到球体上。从一点出发的两条射线形成一个角。如图2-6所示，由这两条射线形成的角存在两种可能的情况。角度定义为这两条射线在单位圆周上截出的圆弧长度。一般规定用较短的弧长来表示角度，其符号由两条射线的顺序来确定，这时所有的角度值都位于$[-\pi, \pi]$之间。

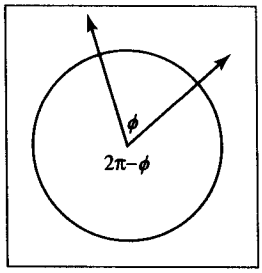


图2-6 两条射线将一单位圆周分成两段圆弧。每段圆弧的长度都可以用来表示两射线之间的有效夹角。一般用较短的一段弧长表示夹角ϕ，或者规定两条射线的顺序，用第一条射线按逆时针旋转到第二条射线所扫过的弧长来表示角度ϕ

每个角度都是两条射线在单位圆周上切出的圆弧的长度。由于单位圆的周长是2π，因此两射线夹角的两种情况之和等于2π。这时的弧长单位是弧度。另一种常用的角度单位是度，这时圆的周长是360度，因此一个π弧度的角度是180度，表示为180°。度与弧度之间的转换关系为

$$\text{度} = \frac{180}{\pi}\text{弧度}$$

$$\text{弧度} = \frac{\pi}{180}\text{度}$$

2.3.2 三角函数

已知一个直角三角形，边长分别为a、o、h，其中h为斜边（与直角相对），边长最长。毕达哥拉斯（Pythagorean）定理[1]描述了直角三角形中关于边长的重要关系：

$$a^2 + o^2 = h^2$$

从图2-7可以看出，这种关系是成立的。图中的大正方形面积为$(a+o)^2$，四个小三角形的总面积为$2ao$，中间小正方形的面积为h^2。

由于外部大正方形由四个三角形和内部小正方形构成，所以有$2ao+h^2=(a+o)^2$，从这个结果可以很容易推出上面的毕达哥拉斯定理。 20

定义ϕ的正弦、余弦及其他三角函数如下：

$$\sin\phi \equiv o/h$$
$$\csc\phi \equiv h/o$$
$$\cos\phi \equiv a/h$$
$$\sec\phi \equiv h/a$$
$$\tan\phi \equiv o/a$$
$$\cot\phi \equiv a/o$$

下面根据上述三角函数建立极坐标，即一个点用它到原点的距离和它相对x轴正向的方向角来表示，如图2-8所示。注意一般规定角度的范围是$\phi \in (-\pi, \pi]$，正角是指从x轴的正方向按逆时针旋转而成的角。在图形学中经常约定逆时针旋转为正，这一点一定要牢记。

三角函数是周期函数，多个自变量值对应同一个函数值，如$\sin(A)=\sin(A+2\pi)$。这就意味着这些函数在实数域$\mathbb{R}$内是不可逆的。为了使三角函数可逆，必须对取值范围进行限制。几乎所有的现代数学书籍（如（Plauger,1991））都做了这种限制。定义域和值域规定为

1. 毕达哥拉斯定理即勾股定理。——译者注

$$\begin{aligned}
&\text{asin}:[-1,1]\mapsto[-\pi/2,\pi/2]\\
&\text{acos}:[-1,1]\mapsto[0,\pi]\\
&\text{atan}:\mathbb{R}\mapsto[-\pi/2,\pi/2]\\
&\text{atan2}:\mathbb{R}^2\mapsto[-\pi,\pi]
\end{aligned}\tag{2-2}$$

最后一个函数atan2(s, c)是非常有用的。其中s与sin A成正比，c与cos A成正比，且比例因子相同，atan2(s, c)的函数值为A。假设该比例因子为正数。一种理解方式是，函数返回极坐标系下二维笛卡儿点(s, c)的角度，如图2-9所示。

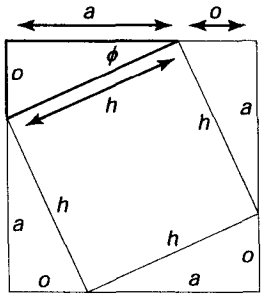


图2-7　毕达哥拉斯定理的几何解释

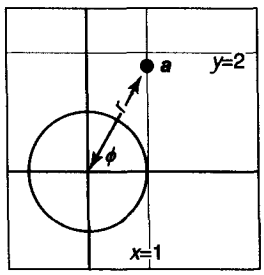


图2-8　点$(x_a, y_a)=(1,\sqrt{3})$的极坐标为(r_a, ϕ_a)=(2, π/3)

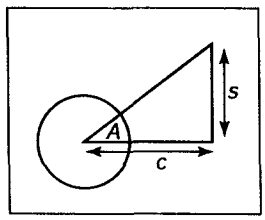


图2-9　函数atan2(s, c)返回角A，在图形学中这一函数非常有用

2.3.3　三角函数公式

这部分列出各种有用的三角函数公式。

诱导公式：

$$\begin{aligned}
\sin(-A)&=-\sin A\\
\cos(-A)&=\cos A\\
\tan(-A)&=-\tan A\\
\sin(\pi/2-A)&=\cos A\\
\cos(\pi/2-A)&=\sin A\\
\tan(\pi/2-A)&=\cot A
\end{aligned}$$

21

毕达哥拉斯公式：

$$\begin{aligned}
\sin^2 A+\cos^2 A&=1\\
\sec^2 A-\tan^2 A&=1\\
\csc^2 A-\cot^2 A&=1
\end{aligned}$$

和差公式：

$$\begin{aligned}
\sin(A+B)&=\sin A\cos B+\sin B\cos A\\
\sin(A-B)&=\sin A\cos B-\sin B\cos A\\
\sin(2A)&=2\sin A\cos A\\
\cos(A+B)&=\cos A\cos B-\sin A\sin B\\
\cos(A-B)&=\cos A\cos B+\sin A\sin B\\
\cos(2A)&=\cos^2 A-\sin^2 A\\
\tan(A+B)&=\frac{\tan A+\tan B}{1-\tan A\tan B}
\end{aligned}$$

$$\tan(A-B)=\frac{\tan A-\tan B}{1+\tan A\tan B}$$

$$\tan(2A)=\frac{2\tan A}{1-\tan^2 A}$$

半角公式：

$$\sin^2(A/2)=(1-\cos A)/2$$

$$\cos^2(A/2)=(1+\cos A)/2$$

积化和差公式：

$$\sin A\sin B=-(\cos(A+B)-\cos(A-B))/2$$

$$\sin A\cos B=(\sin(A+B)+\sin(A-B))/2$$

$$\cos A\cos B=(\cos(A+B)+\cos(A-B))/2$$

对于边长为a、b、c，对应角分别为A、B、C的任意三角形（见图2-10），下列公式都成立：

$$\frac{\sin A}{a}=\frac{\sin B}{b}=\frac{\sin C}{c}\quad\text{（正弦定理）}$$

$$c^2=a^2+b^2-2ab\cos C\quad\text{（余弦定理）}$$

$$\frac{a+b}{a-b}=\frac{\tan\left(\frac{A+B}{2}\right)}{\tan\left(\frac{A-B}{2}\right)}\quad\text{（正切定理）}$$

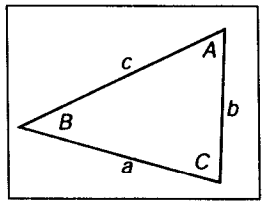


图2-10 三角定理对应的几何图形

三角形的面积同样可以通过三条边长表示出来：

$$\text{三角形面积}=\frac{1}{2}\sqrt{(a+b+c)(-a+b+c)(a-b+c)(a+b-c)}$$

22

2.4 向量

一个向量包含长度和方向，可以用一个箭头表示。如果两个向量有相同的长度和方向，那么它们是相等的向量，即使它们处在不同的位置上（见图2-11）。尽量把向量看成是一个箭头，而不是坐标或者数值。即使是在代码中，也应该把向量作为对象来处理。有时在程序中不得不把向量表示成数值，但只有低级向量运算才需要知道向量的数值表示（DeRose,1989）。在本书中，向量用粗体字母表示，如$\boldsymbol{a}$，向量长度表示为$\|\boldsymbol{a}\|$。单位向量是指长度为1的向量。零向量是指长度为0的向量。零向量没有定义方向。

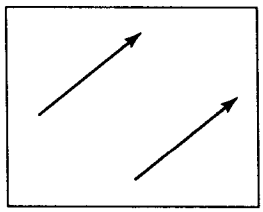


图2-11 这两个向量是相同的，因为它们有相同的长度和方向

向量可以表示很多事物。例如,可以用向量表示偏移量，或者称为位移。假如我们知道“财宝埋在离秘密会场东两步、北三步的地方”，我们就知道了偏移量，但是不知道从哪里开始。也可以用向量表示位置或点。位置可以用偏离另一个位置的位移来表示。一般先确定一个原点位置，其他位置就可表示为从原点位置出发的位移。注意，位置本身不是向量。后面我们将要讨论两个向量相加，但是将两个位置相加没有意义，除非是对位置进行加权平均的中间运算步骤（Goldman, 1985）。两个位移相加是有意义的，这就是为什么位移是向量的原因。

必须强调位置不是位移，它是从一个指定的原点位置的位移。位移也不是位置。

2.4.1 向量运算

我们知道可以对实数进行各种算术运算，对向量也一样。当且仅当两个向量具有相同的方向和长度时，两个向量才是相等的。两个向量的加法遵守*平行四边形法则*。求和时，把一个向量的箭尾与另一个向量的箭头相联，如图2-12所示。和向量与参与运算的两个向量构成了一个完整的三角形。向量加法满足交换律：

$$\boldsymbol{a}+\boldsymbol{b}=\boldsymbol{b}+\boldsymbol{a}$$

平行四边形法则使我们对位移有了直观的认识。想象一个人沿着一个向量从头走到尾，然后再沿
23 着另一个向量从头走到尾，产生的净位移就是平行四边形的对角线。同样可以求出向量的*一元负向量*：$-\boldsymbol{a}$（见图2-13），也就是与原向量长度相同但方向相反的向量。于是有向量*减法*：

$$\boldsymbol{b}-\boldsymbol{a}\equiv-\boldsymbol{a}+\boldsymbol{b}$$

减法也可以用平行四边形法则来表示，如图2-14所示。我们可以得到：

$$\boldsymbol{a}+(\boldsymbol{b}-\boldsymbol{a})=\boldsymbol{b}$$

向量也可以相乘。实际上，向量相乘的方式有很多。首先，可以用一个实数k倍乘向量，这时改变了向量的长度而不改变向量的方向。例如，向量$3.5\boldsymbol{a}$长度是向量$\boldsymbol{a}$的3.5倍，而方向与向量$\boldsymbol{a}$相同。求两向量之积的方法有很多，下面讨论其中的三种：点积、叉积和行列式运算。

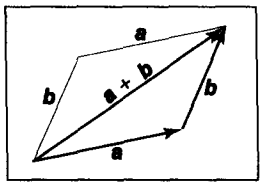


图2-12　两向量头尾相接实现加法运算，满足交换律

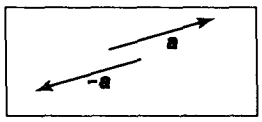


图2-13　向量$-\boldsymbol{a}$与向量$\boldsymbol{a}$长度相同、方向相反

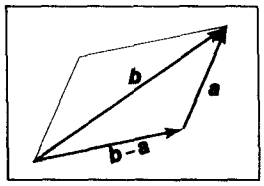


图2-14　向量减法，可以将第二个变量取负后进行加法运算

2.4.2 向量的笛卡儿坐标

任意一个二维向量可以由两个互不平行的非零向量组合而成。这两个向量所具有的这种性质称为*线性独立*。线性独立的两个向量就构成了一个二维*基*，这两个向量相应地称为*基向量*。例如，向量$\boldsymbol{c}$可以表示为两个基向量$\boldsymbol{a}$和$\boldsymbol{b}$的组合（图2-15）：

$$\boldsymbol{c}=a_c\boldsymbol{a}+b_c\boldsymbol{b} \tag{2-3}$$

注意权系数a_c和b_c是唯一的。非常有用的一种情况是，这两个向量是*正交向量*，也就是说它们互相垂直。如果它们是单位正交向量即*标准正交*时，就更有用了。如果已知两个标准正交向量$\boldsymbol{x}$和$\boldsymbol{y}$，就可以用它们表示笛卡儿坐标系中的其他所有向量，其中每个向量用一对实数表示。例如，向量$\boldsymbol{a}$可以表示为

$$\boldsymbol{a}=x_a\boldsymbol{x}+y_a\boldsymbol{y}$$

其中x_a和y_a是二维向量$\boldsymbol{a}$的实笛卡儿坐标（见图2-16）。注意这与式（2-3）在概念上并没有多少差别，只是式（2-3）中的基向量不是正交基。笛卡儿坐标系有个优点：根据毕达哥拉斯定理，$\boldsymbol{a}$的长度为

24

$$\|\boldsymbol{a}\|=\sqrt{x_a^2+y_a^2}$$

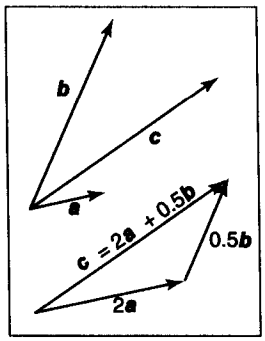


图2-15 任一二维向量c可以由两个互不平行的二维向量a和b的加权和表示

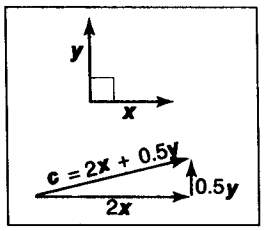


图2-16 向量的二维笛卡儿基

一般我们把a的坐标写成有序对（x_a，y_a），或者用列矩阵表示：

$$\boldsymbol{a}=\begin{bmatrix}x_a\\y_a\end{bmatrix}$$

两种书写方式都可以采用，具体看哪种方式写起来方便。有时也把向量写成行矩阵的形式，这时用$\boldsymbol{a}^T$表示：

$$\boldsymbol{a}^T=\begin{bmatrix}x_a & y_a\end{bmatrix}$$

也可以表示出笛卡儿坐标系下的三维、四维等向量。对于三维情况，我们要用到同时与x和y都正交的基向量z。

2.4.3 点积

两个向量相乘的最简单方式就是点积。向量a与b的点积用$a\cdot b$表示，通常称为标量积，因为它返回的是一个标量。向量的点积与两个向量的长度以及它们之间的夹角ϕ有关（见图2-17）：

$$\boldsymbol{a}\cdot\boldsymbol{b}=\|\boldsymbol{a}\|\ \|\boldsymbol{b}\|\ \cos\phi \tag{2-4}$$

在图形程序中，一般通过点积计算两向量间夹角ϕ的余弦值。

也可通过点积求一个向量到另一个向量的投影，投影的长度$\boldsymbol{a}\rightarrow\boldsymbol{b}$为（见图2-18）：

$$\boldsymbol{a}\rightarrow\boldsymbol{b}=\|\boldsymbol{a}\|\cos\phi=\frac{\boldsymbol{a}\cdot\boldsymbol{b}}{\|\boldsymbol{b}\|} \tag{2-5}$$

类似实数运算，点积运算遵循交换律、结合律及分配律：

$$\begin{aligned}\boldsymbol{a}\cdot\boldsymbol{b}&=\boldsymbol{b}\cdot\boldsymbol{a}\\ \boldsymbol{a}\cdot(\boldsymbol{b}+\mathbf{c})&=\boldsymbol{a}\cdot\boldsymbol{b}+\boldsymbol{a}\cdot\mathbf{c}\\ (k\boldsymbol{a})\cdot\boldsymbol{b}&=\boldsymbol{a}\cdot(k\boldsymbol{b})=k\boldsymbol{a}\cdot\boldsymbol{b}\end{aligned} \tag{2-6}$$

如果二维向量a和b以笛卡儿坐标的形式表示，我们就可以利用$x\cdot x=y\cdot y=1$以及$x\cdot y=0$推出它们的点积结果：

$$\begin{aligned}\boldsymbol{a}\cdot\boldsymbol{b}&=(x_a\boldsymbol{x}+y_a\boldsymbol{y})\cdot(x_b\boldsymbol{x}+y_b\boldsymbol{y})\\ &=x_ax_b(\boldsymbol{x}\cdot\boldsymbol{x})+x_ay_b(\boldsymbol{x}\cdot\boldsymbol{y})+x_by_a(\boldsymbol{y}\cdot\boldsymbol{x})+y_ay_b(\boldsymbol{y}\cdot\boldsymbol{y})\\ &=x_ax_b+y_ay_b\end{aligned}$$

同样，对于三维情况可以得出

$$\boldsymbol{a}\cdot\boldsymbol{b}=x_ax_b+y_ay_b+z_az_b$$

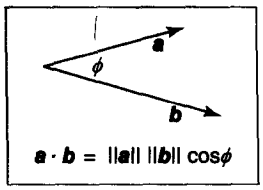


图2-17　点积与向量的长度和夹角有关，这是图形学中最重要的公式之一

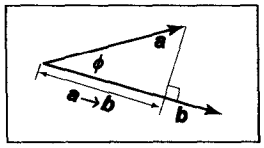


图2-18　a到b的投影等于用式（2-5）求出的长度

2.4.4　叉积

叉积$a \times b$常用于三维向量。一般叉积运算在“文献注记”部分提到的参考资料中有介绍。叉积$a \times b$返回一个三维向量，该向量与另两个向量a和b都是正交的。叉积结果得到的向量，其长度与$\sin\phi$相关：

$$\|a \times b\| = \|a\| \ \|b\| \ \sin\phi$$

幅值$\|a \times b\|$等于两向量a和b形成的平行四边形的面积，而且$a \times b$与a及b都正交（见图2-19）。注意叉积向量的方向只存在两种可能。定义x、y、z轴上的方向向量为

$$x = (1,0,0)$$
$$y = (0,1,0)$$
$$z = (0,0,1)$$

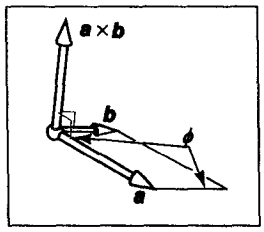


图2-19　叉积$a \times b$是一个三维向量，它与三维向量a及b都正交，其长度等于图中平行四边形的面积

按惯例$x \times y$的方向应在z轴的正向或者负向。虽然可以任意选择，但是一般标准是取

$$z = x \times y$$

以上三个三维单位向量的所有叉积如下：

$$x \times y = +z$$
$$y \times x = -z$$
$$y \times z = +x$$
$$z \times y = -x$$
$$z \times x = +y$$
$$x \times z = -y$$

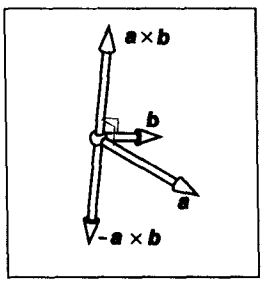


图2-20　叉积的右手法则。假设把你的右手掌底部放在a和b箭尾相交的地方，然后将箭头a推向箭头b，则你的大拇指就指向$a \times b$

26

根据$\sin\phi$的性质，我们知道一个向量与它本身的叉积结果为零向量，即$x \times x = 0$。注意：叉积不服从交换律，也就是说$x \times y \neq y \times x$。如果足够细心，我们还会注意到：根据上面的讨论，我们还画不出能准确反映坐标轴关系的笛卡儿坐标系。举个具体例子，如果我们把x和y放在人行道上，x指向东，y指向北，那么z将指向天空还是指向地面？一般惯例是z指向天空。这就是*右手坐标系法则*。伸开右手，让手掌指向x轴，然后让4个手指向y轴弯曲，这时大拇指的指向就是z轴的方向。这就是右手坐标系这一名称的来源，如图2-20所示。

叉积具有很有用的两个性质，即

$$a \times (b + c) = a \times b + a \times c$$

和

$$\boldsymbol{a}\times(k\boldsymbol{b})=k(\boldsymbol{a}\times\boldsymbol{b})$$

然而根据右手法则可知：

$$\boldsymbol{a}\times\boldsymbol{b}=-(\boldsymbol{b}\times\boldsymbol{a})$$

在笛卡儿坐标系中，我们可以利用展开式计算叉积：

$$\begin{aligned}\boldsymbol{a}\times\boldsymbol{b}&=(x_a\boldsymbol{x}+y_a\boldsymbol{y}+z_a\boldsymbol{z})\times(x_b\boldsymbol{x}+y_b\boldsymbol{y}+z_b\boldsymbol{z})\\&=x_ax_b\boldsymbol{x}\times\boldsymbol{x}+x_ay_b\boldsymbol{x}\times\boldsymbol{y}+x_az_b\boldsymbol{x}\times\boldsymbol{z}\\&\quad+y_ax_b\boldsymbol{y}\times\boldsymbol{x}+y_ay_b\boldsymbol{y}\times\boldsymbol{y}+y_az_b\boldsymbol{y}\times\boldsymbol{z}\\&\quad+z_ax_b\boldsymbol{z}\times\boldsymbol{x}+z_ay_b\boldsymbol{z}\times\boldsymbol{y}+z_az_b\boldsymbol{z}\times\boldsymbol{z}\\&=(y_az_b-z_ay_b)\boldsymbol{x}+(z_ax_b-x_az_b)\boldsymbol{y}+(x_ay_b-y_ax_b)\boldsymbol{z}\end{aligned}\tag{2-7}$$

用坐标形式则表示为

$$\boldsymbol{a}\times\boldsymbol{b}=(y_az_b-z_ay_b,z_ax_b-x_az_b,x_ay_b-y_ax_b)\tag{2-8}$$

2.4.5 标准正交基与坐标系

建立坐标系是几乎任何图形学程序中的核心任务之一。这一任务的关键就是计算标准正交基。对任意两个二维向量$\boldsymbol{u}$和$\boldsymbol{v}$，只要它们是正交的（夹角为直角），并且它们的长度为单位长度，那么它们就构成了一对标准正交基。于是有

$$\|\boldsymbol{u}\|=\|\boldsymbol{v}\|=1$$

和

$$\boldsymbol{u}\cdot\boldsymbol{v}=0$$

27

在三维空间中，三个向量$\boldsymbol{u}$、$\boldsymbol{v}$、$\boldsymbol{w}$如果满足下列条件，它们就构成一组标准正交基：

$$\|\boldsymbol{u}\|=\|\boldsymbol{v}\|=\|\boldsymbol{w}\|=1$$

和

$$\boldsymbol{u}\cdot\boldsymbol{v}=\boldsymbol{v}\cdot\boldsymbol{w}=\boldsymbol{w}\cdot\boldsymbol{u}=0$$

如果$\boldsymbol{w}=\boldsymbol{u}\times\boldsymbol{v}$，那么这个标准正交基满足右手法则；否则就满足左手法则。

笛卡儿标准正交基只是无穷多个标准正交基中的一种。特别地，程序中进行低级表示时，要用到笛卡儿标准正交基及其隐含的原点位置。向量$\boldsymbol{x}$、$\boldsymbol{y}$、$\boldsymbol{z}$从不进行显式存储，对标准原点$\boldsymbol{o}$也是这样。全局模型存储在标准坐标系中，因此常称该坐标系为全局坐标系。另一方面，如果想使用另一个原点为$\boldsymbol{p}$、标准正交基向量为$\boldsymbol{u}$、$\boldsymbol{v}$、$\boldsymbol{w}$的坐标系，我们就要显式存储这些向量。这样的坐标系称为参考坐标系或坐标系。例如，在飞行模拟器中，我们可能需要建立原点位于飞机前端、标准正交基与飞机布局一致的坐标系。同时还要有主标准坐标系（见图2-21）。常称与特定目标（如飞机）相关的坐标系为局部坐标系。

28

在低层，本地坐标系存于标准坐标系中。例如：

$$\boldsymbol{u}=(x_u,y_u,z_u)\equiv x_u\boldsymbol{x}+y_u\boldsymbol{y}+z_u\boldsymbol{z}$$

一个位置隐含一个到标准原点的位移：

$$\boldsymbol{p}=(x_p,y_p,z_p)\equiv\boldsymbol{o}+x_p\boldsymbol{x}+y_p\boldsymbol{y}+z_p\boldsymbol{z}$$

注意，如果我们要存储uvw系下的向量$\boldsymbol{a}$，就存储一个三元组(u_a,v_a,w_a)，其几何意义为

$$(u_a,v_a,w_a)\equiv u_a\boldsymbol{u}+v_a\boldsymbol{v}+w_a\boldsymbol{w}$$

为了获得存储在uvw系中的向量$\boldsymbol{a}$的标准坐标，只要知道$\boldsymbol{u}$、$\boldsymbol{v}$、$\boldsymbol{w}$本身是按笛卡儿坐标存储的，就可以显式求出笛卡儿坐标表达式$u_a\boldsymbol{u}+v_a\boldsymbol{v}+w_a\boldsymbol{w}$。6.2.1节和6.5节将讨论坐标系变换的矩阵运算。

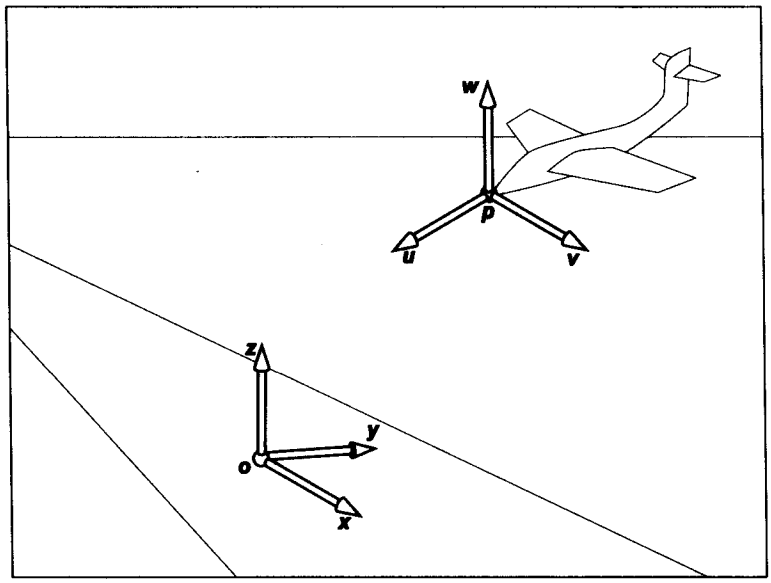


图2-21　主坐标系或者标准坐标系的原点为o，标准正交基为x、y、z。该坐标系一般与全局模型布局一致，因此常称为全局坐标系或世界坐标系。其坐标原点和基向量不以显式存储，而所有其他向量和位置都以相对于世界坐标系的坐标存储。飞机坐标系是相对全局坐标系显式存储的

2.4.6　从单向量建立正交基

经常需要从单个向量构造一标准正交基，也就是说，已知一个向量a，需要找出一组标准正交向量u、v、w，使得w的指向与向量a的方向相同（Hughes & Möller,1999）。结果不是唯一的，但可以通过适当求解过程，找到符合要求的一组基。具体如下：

$$w=\frac{a}{\|a\|}$$

为了求u和v，需要找到一个与w不共线的向量t。为此，只要让t等于w，再将其中绝对值最小的元素变成1即可。例如，如果$w=(1/\sqrt{2},-1/\sqrt{2},0)$，那么$t=(1/\sqrt{2},-1/\sqrt{2},1)$。于是$u$和$v$很容易求出：

$$u=\frac{t\times w}{\|t\times w\|}$$

$$v=w\times u$$

2.5　二维隐式曲线

直观上，一条曲线就是可以在纸上一笔画出的点的集合。表示曲线的一般方法是使用隐式方程。隐式方程具有如下的形式：

$$f(x,y)=0$$

函数$f(x,y)$返回实数值。函数值为零的点(x,y)位于该曲线上，函数值不为零的点不位于该曲线上。例如，假设$f(x,y)$为

$$f(x,y)=(x-x_c)^2+(y-y_c)^2-r^2 \tag{2-9}$$

其中(x_c,y_c)是一个二维的点，r是一个非零实数。如果令$f(x,y)=0$，那么使上述方程成立的点在圆心为(x_c,y_c)、半径为r的圆周上。该方程称为隐式方程的原因是，曲线上的点(x,y)不能根据方程

很快计算出来，而需要将(x, y)代入f中验证函数值是否为零，或者通过解方程来得到结果。因此，曲线上的点并不是由方程显式产生，而是隐含在方程中。

有意思的是，f并不是对所有的(x, y)都有值。可以把f想象成一种地形，海平面位于$f=0$处（见图2-22）。海岸线就是隐式曲线，f的值就是海拔高度。另外，曲线将空间分成$f>0, f<0$和$f=0$三个区域。因此可以通过判断f的值，来估计某点是否在曲线上。注意$f(x,y)=c$是取任意常量c时的曲线，只是通常取$c=0$。例如，对于$f(x,y)=x^2+y^2-1$，不同c值对应一组以原点为圆心的圆（见图2-23）。

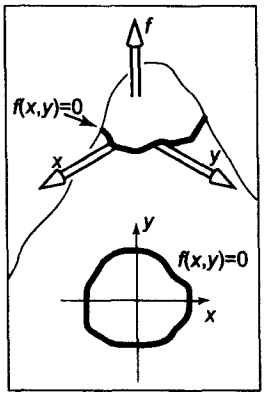


图2-22 可将隐式函数$f(x, y)=0$想象成某个高度图，f就是高度（上图）；高度为零处的轨迹就是隐式曲线（下图）

可以采用向量以简化表示方法。设$\boldsymbol{c}=(x_c,y_c)$，$\boldsymbol{p}=(x,y)$，则以$\boldsymbol{c}$为圆心、r为半径的圆就可以通过位置向量表示为

$$(\boldsymbol{p}-\boldsymbol{c})\cdot(\boldsymbol{p}-\boldsymbol{c})-r^2=0$$

将这个方程进行代数展开，就能得到式（2-9），但从几何意义上更容易看出这是一个圆的方程。上式可以解释为，“在圆周上的点$\boldsymbol{p}$具有以下性质：从点$\boldsymbol{c}$到点$\boldsymbol{p}$的向量，与该向量自身的点积结果为r^2”。如果编程实现一个向量类型，那么实现一个向量方程要比实现一个完全展开的方程更容易，将不会出现关于x、y、z的裁剪、粘贴错误。由于向量与其本身的点积就是其长度的平方，我们也可以把方程解释为：“在圆周上的点$\boldsymbol{p}$具有如下的性质：由$\boldsymbol{c}$到$\boldsymbol{p}$的向量长度的平方为r^2”。

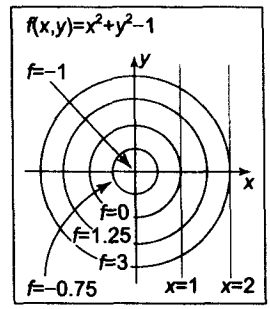


图2-23 不同c值对应一组以原点为圆心的圆

30

更好的解释是：r^2就是从$\boldsymbol{c}$到$\boldsymbol{p}$的距离的平方，这意味着如下方程成立：

$$\|\boldsymbol{p}-\boldsymbol{c}\|^2-r^2=0$$

当然也就意味着

$$\|\boldsymbol{p}-\boldsymbol{c}\|-r=0$$

此式可以解释为：“圆周上的点$\boldsymbol{p}$，就是那些到圆心$\boldsymbol{c}$的距离为r的点”。这种定义圆的方式非常容易理解。以上表明，向量方程要比与其等价的含x和y的完全笛卡儿形式更有优势，其几何意义更丰富，直观上也更容易理解。因此，如果可能我们就使用向量方程的形式。此外，在代码中可以采用向量类，采用向量形式可使代码显得更清晰。要习惯这些方程中的向量，需要花费一点时间。但如果你拒绝使用向量，代价将更大。

2.5.1 二维梯度

如果我们把函数$f(x,y)$看成高度为$f(x,y)$的高度图，那么梯度向量就指向坡度最大的方向，也就是直接上坡。梯度向量$\nabla f(x,y)$由下式给出：

$$\nabla f(x,y)=\left(\frac{\partial f}{\partial x},\frac{\partial f}{\partial y}\right)$$

隐式曲线$f(x, y)=0$上，某点处的梯度向量与曲线在该点处的切向量正交。该梯度向量一般称为曲线的法向量。此外，因为梯度指向上坡的方向，它就指明了$f(x,y)>0$的区域的方向。

在高度图情况下，偏导数和梯度的几何意义就更直观了。假设在点(a, b)附近$f(x,y)$是一个平面（见图2-24）。有明确的上坡和下坡方向，与该方向垂直的方向就是关于这个平面的水平方向。

该平面与平面$f(x, y) = 0$的任意交线都将在这个水平方向上。因此，上坡/下坡方向正交于平面与$f(x, y)$的交线。为讲清楚为什么偏导数与该问题有关，我们需要理解偏导数的几何意义。一维函数$y = g(x)$的导数为：

$$\frac{dy}{dx} \equiv \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{g(x+\Delta x) - g(x)}{\Delta x} \tag{2-10}$$

31 它表示函数g的切线的斜率（见图2-25）。

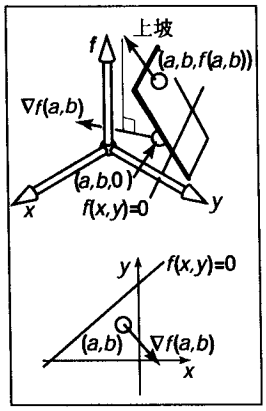


图2-24　高度为$f(x, y)$的曲面，在$(x, y) = (a, b)$附近是局部平坦的。梯度就是上坡方向在高度为零的平面上的投影

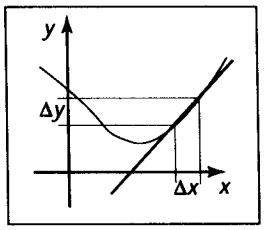


图2-25　一维函数的导数表示曲线切线的斜率

偏导数是一维导数的推广。对于二维函数$f(x, y)$，我们不能像式（2-10）那样对x求极限，因为当x做某种变化时，f对应多种变化情况。然而，如果我们把y看作常量，则可以仿照导数的定义，给出偏导数的定义（见图2-26）：

$$\frac{\partial f}{\partial x} \equiv \lim_{\Delta x \to 0} \frac{f(x+\Delta x, y) - f(x, y)}{\Delta x}$$

为什么梯度向量的两个成分分别是x、y的偏导数呢？从几何上比从代数上更容易看明白。在图2-27中，向量$\boldsymbol{a}$沿着f不改变的一条路径。注意此时曲面在(x, y)处的高度为$f(x, y)$，是指在(x, y)附近的足够小区域内，认为曲面是局部平坦的。从图中可以看出，向量$\boldsymbol{a} = (\Delta x, \Delta y)$。

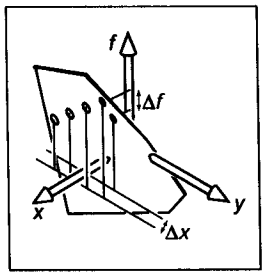


图2-26　二维函数f的偏导数，必须将y看作是具有唯一值的常量，如图中的黑点所示。中空的点表示y不是常量时的其他f值

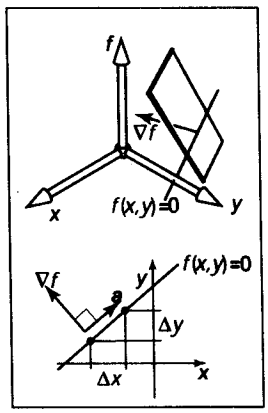


图2-27　向量$\boldsymbol{a}$指向f值不发生变化的方向，因此它与梯度向量∇f垂直

由于上坡方向与$\boldsymbol{a}$是垂直的，所以它们的点积为零：

$$(\nabla f)\cdot \boldsymbol{a} = (x_\nabla, y_\nabla)\cdot(x_a, y_a) = x_\nabla \Delta x + y_\nabla \Delta y = 0 \tag{2-11}$$

我们还知道，f在(x_a, y_a)方向上的变化为零：

$$\Delta f = \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y \equiv \frac{\partial f}{\partial x}x_a + \frac{\partial f}{\partial y}y_a = 0 \tag{2-12}$$

对于任意两个互相垂直的向量(x, y)和(x', y')，由于它们夹角为90°，所以它们的点积为零(两个向量的点积与它们夹角的余弦成正比)，从而有$xx' + yy' = 0$。对于给定的向量(x, y)，构造出一个与之点积为零的向量很容易，最显而易见的两个就是$(y, -x)$和$(-y, x)$，可以验证它们与(x, y)的点积为零。更一般的情况是(x, y)与$k(y, -x)$垂直，其中k是一个非零常量。于是得到

$$(x_a, y_a) = k\left(\frac{\partial f}{\partial y}, -\frac{\partial f}{\partial x}\right) \tag{2-13}$$

从式（2-11）和式（2-13）可以得到：

$$(x_\nabla, y_\nabla) = k'\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

其中k'是任意的非零常量。根据定义，“上坡”隐含f有正的改变，因此我们希望$k' > 0$，一般规定$k' = 1$。

32

下面举个梯度实例。已知圆的隐式方程为$x^2 + y^2 - 1 = 0$，其梯度向量为$(2x, 2y)$，表示圆外的区域是使函数$f(x, y) = x^2 + y^2 - 1$值为正的区域。注意梯度向量的长度取决于隐式方程里的系数。例如单位圆可以用$Ax^2 + Ay^2 - A = 0$来描述，其中A为任意的非零值。该曲线的梯度就是$(2Ax, 2Ay)$，方向与圆周垂直，但其长度由A决定。$A>0$时，法向量就指向圆外；而$A<0$时，法向量就指向圆内。因为到了圆的内部，函数值为正的区域发生了变化，所以法向量在圆的内外指向不同。对于高度图$h = Ax^2 + Ay^2 - A$，该圆就是高度为零的点。$A>0$时，这个圆就围着一个洼地；$A<0$时，它就围着一个凸起。当A在负向变得越来越小时，凸起的高度越来越大，但是$h=0$处的圆不发生变化。最大上坡处的方向不发生变化，但是斜率却增大了。因此，可以直观地认为梯度的方向指向上坡的方向，而它的幅值就等于斜率。

2.5.2 隐式二维直线

我们熟悉的直线“斜截式”为

$$y = mx + b \tag{2-14}$$

可以很容易地把它转化为隐含式（见图2-28）

$$y - mx - b = 0 \tag{2-15}$$

这里m是斜率，b是直线穿过y轴时的y值，通常称为y截距。这条直线把二维平面划分为上下两部分。这里称“上、下”比“内、外”更直观一些。

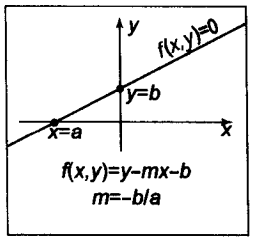


图2-28 二维直线可以用方程$y-mx-b=0$来描述

在不改变方程的零点情况下，可以对隐式方程两边同乘以一个常量。对任意非零的k，$kf(x,y)=0$与$f(x,y)=0$是相同的曲线。所以，同一条直线可以有多种不同的表示方式，例如：

$$2y-2mx-2b=0$$

在有些情况下斜截式不是最好的表示方式，例如直线$x=0$没法用斜截式表示，因为这条直线的m是无穷大。因此更通用的形式是

33

$$Ax+By+C=0 \tag{2-16}$$

其中A、B、C为实数。如果知道两个点(x_0,y_0)和(x_1,y_1)位于直线上，则它们满足式（2-16）的直线方程：

$$Ax_0+By_0+C=0$$
$$Ax_1+By_1+C=0$$

在这两个方程中存在三个未知数A、B、C。这是由于隐式方程系数的任意性而带来的问题。为方便起见，我们可以令C=1，于是有

$$Ax+By+1=0$$

但这时又会出现与斜截式类似的无穷斜率问题：如果直线过原点，则有$A(0)+B(0)+1=0$。此时A或B中就得有一个是无穷大了。举例来说，一个过原点的45°直线可以写为$x-y=0$或者$y-x=0$，甚至写为$17y-17x=0$的形式，但是不能写为$Ax+By+1=0$形式。

遇到这一代数难题，需要用几何方法来解决。工具之一就是2.5.1节中讨论的梯度。对于直线$Ax+By+C=0$，其梯度向量是(A,B)。梯度向量与直线垂直，并指向$Ax+By+C$值为正的区域（见图2-29）。对于直线上的两个点(x_0,y_0)和(x_1,y_1)，我们知道这两点之间的向量方向与直线的方向相同。两点之间的向量表示为(x_1-x_0,y_1-y_0)，由于它与直线平行，必定与梯度向量(A,B)垂直。由于直线的隐式表示可乘以任意系数，则存在无穷多组(A,B,C)满足要求，我们只需要其中的一组。

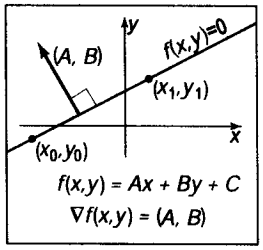


图2-29　梯度向量(A,B)垂直于隐式直线$Ax+By+C=0$

可以先找任意与(x_1-x_0,y_1-y_0)垂直的(A,B)。向量$(A,B)=(y_0-y_1,x_1-x_0)$就满足要求，原因见2.5.1节的内容。这意味着过点(x_0,y_0)和(x_1,y_1)的直线方程是

$$(y_0-y_1)x+(x_1-x_0)y+C=0 \tag{2-17}$$

现在只要求出C就可以了。因为点(x_0,y_0)和(x_1,y_1)位于直线上，所以它们都满足式（2-17）。可以将两点中的任意一个代入方程并求出C。将(x_0,y_0)代入，得到$C=x_0y_1-x_1y_0$。于是，该直线的完整表达形式是

$$(y_0-y_1)x+(x_1-x_0)y+x_0y_1-x_1y_0=0 \tag{2-18}$$

通过两点的直线隐式方程有多种表达方式，这只是其中的一种。这种形式没有除法运算，因此避免了有限笛卡儿坐标点的数值退化问题。式（2-18）表达方式的一个优点是，可以通过将不含y

34

的项移到方程右边，然后两边同除以y项的系数将其转化为斜截式：

$$y=\frac{y_1-y_0}{x_1-x_0}x+\frac{x_1y_0-x_0y_1}{x_1-x_0}$$

隐式直线方程存在一个有趣的特性，那就是：可以用它来计算点到直线的有向距离。$Ax+By+C$的值与到该直线的距离成正比（见图2-30）。如图2-31所示，从一个点到直线的距离就是向量$k(A, B)$的长度，也就是：

$$\text{距离}=k\sqrt{A^2+B^2} \tag{2-19}$$

对于点$(x,y)+k(A,B)$，$f(x,y)=Ax+By+C$的值为

$$\begin{aligned} f(x+kA,y+kB) &= Ax+kA^2+By+kB^2+C \\ &= k(A^2+B^2) \end{aligned} \tag{2-20}$$

对上述方程化简时用到了$Ax+By+C=0$，因为点(x,y)位于直线上。从式（2-19）和式（2-20）可以看出，从点(a,b)到直线$Ax+By+C=0$的有向距离为

$$\text{距离}=\frac{f(a,b)}{\sqrt{A^2+B^2}}$$

这里的“有向距离”是指，其幅值（绝对值）等于距离值，但可正可负。在直线的一边距离是正的，而在另一边距离就是负的。注意如果(A,B)是单位向量，那么$f(a,b)$就表示有向距离。可以对式（2-18）乘以一个常数，以保证(A,B)是单位向量，于是有

$$\begin{aligned} f(x,y)=&\frac{y_0-y_1}{\sqrt{(x_1-x_0)^2+(y_0-y_1)^2}}x+\frac{x_1-x_0}{\sqrt{(x_1-x_0)^2+(y_0-y_1)^2}}y \\ &+\frac{x_0y_1-x_1y_0}{\sqrt{(x_1-x_0)^2+(y_0-y_1)^2}} \end{aligned} \tag{2-21}$$

通过式（2-21）计算$f(x,y)$的值，就直接得出有向距离，但是需要增加一项平方根运算。另外还应注意到，尽管在直线的一边距离为正、在直线的另一边距离为负，也可以通过选择使另一边的距离为正。方法是在$f(x,y)=0$和$-f(x,y)=0$中选择使用，使得在直线规定的一边距离为正。这一点在三角形光栅化技术中非常有用(见3.6节)。二维直线的其他表示形式在第14章进行讨论。

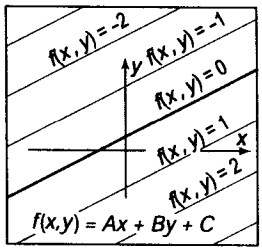


图2-30 隐式函数$f(x,y)=Ax+By+C$的值，等于到$Ax+By+C=0$的有向距离的常数倍

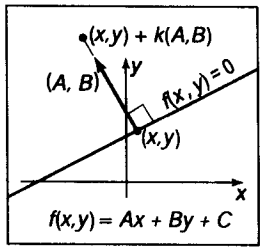


图2-31 向量$k(A, B)$把直线外的一个点与直线上与该点最近的点(x, y)连起来，两点之间的距离与k成正比

35

2.5.3 隐式二次曲线

对于二维二次曲线，也就是椭圆和抛物线，以及特殊的双曲线、圆和直线，都可以用一个通用的隐形式来表示：

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

对于圆心为(x_c, y_c)、半径为r的圆来说，上式变为

$$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$$

如下是抛物线方程

$$y - k(x - x_c)^2 = 0$$

其中k是一个非零常数，x_c是抛物线的对称轴。另外有一类抛物线的对称轴是水平的。对称轴与坐标轴平行的椭圆的方程为

$$\frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} - 1 = 0$$

其中(x_c, y_c)是椭圆的中心，a和b分别是椭圆的长、短半轴的长度（见图2-32）。

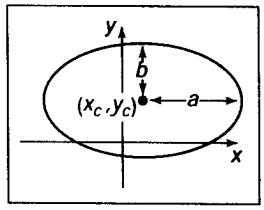


图2-32　以(x_c, y_c)为中心、半轴长分别为a和b的椭圆

2.6　二维参数曲线

参数曲线通过一个参数进行控制，该参数可以看作沿曲线连续运动的指针。这样的曲线具有以下形式：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} g(t) \\ h(t) \end{bmatrix}$$

其中(x,y)是曲线上的一点，t就是影响曲线的参数。对于给定的t，函数g和h确定某一点。如果g和h是连续的，那么当t发生微小变化时，x和y相应会产生微小变化。随着t的不断变化，就产生了曲线上的一系列点。这个特点非常有用，因为我们可以通过改变参数t显式绘制曲线上的点。我们经常把参数曲线写成向量的形式：

$$\boldsymbol{p} = f(t)$$

36 其中f是向量函数$f: \mathbb{R} \to \mathbb{R}^2$。从向量函数可以编写出非常清晰的代码，所以如果可能要尽量使用向量函数。我们可以认为曲线上的位置是时间的函数，该曲线可以延伸到任何地方，可以环绕、交叉。还可以认为曲线的任何一点都存在一个速度。例如，点$\boldsymbol{p}(t)$在$t=-2$附近缓慢运动，而在$t=2$和$t=3$附近运动迅速。在讨论参数曲线时，经常要用到“运动点”这个词，即使当曲线并不真的反映运动点时也这样。

2.6.1　二维参数直线

在二维空间中，过两点$\boldsymbol{p}_0 = (x_0, y_0)$和$\boldsymbol{p}_1 = (x_1, y_1)$的直线参数方程可表示为

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + t(x_1 - x_0) \\ y_0 + t(y_1 - y_0) \end{bmatrix}$$

因为公式中的x和y具有类似的结构，我们可以采用向量形式$\boldsymbol{p}=(x,y)$表示（见图2-33）：

$$\boldsymbol{p}(t) = \boldsymbol{p}_0 + t(\boldsymbol{p}_1 - \boldsymbol{p}_0)$$

几何意义为：在参数t作用下，从$\boldsymbol{p}_0$点出发，经过一段距离走向$\boldsymbol{p}_1$。这种形式的优点是，$\boldsymbol{p}(0)=\boldsymbol{p}_0$且$\boldsymbol{p}(1)=\boldsymbol{p}_1$。因为点随着$t$做线性

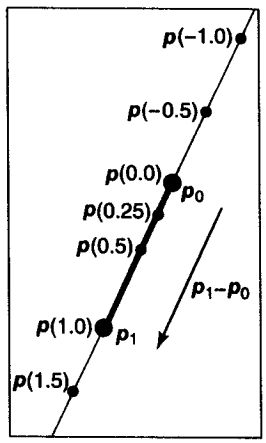


图2-33　通过$\boldsymbol{p}_0$和$\boldsymbol{p}_1$的二维参数直线。粗线段对应$t \in [0,1]$

变化，$\boldsymbol{p}_0$和$\boldsymbol{p}_1$之间的t值就是两点之间相对距离的测度，$t<0$的点离$\boldsymbol{p}_0$较远，$t>1$的点离$\boldsymbol{p}_1$较远。

参数直线也可以只用一个点$\boldsymbol{o}$和一个向量$\boldsymbol{d}$来表示：

$$\boldsymbol{p}(t)=\boldsymbol{o}+t(\boldsymbol{d})$$

当向量$\boldsymbol{d}$具有单位长度时，直线就是弧长参数化的。这意味着t就是直线上距离的确切测度了。任意参数曲线都可以是弧长参数化的，这是一种非常方便的表达形式，但并不是所有曲线都能够转化为解析形式。

2.6.2 二维参数圆

对于圆心为(x_c, y_c)、半径为r的圆，其参数形式为

$$\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} x_c+r\cos\phi \\ y_c+r\sin\phi \end{bmatrix}$$

37

为了保证曲线上每点只对应唯一的参数ϕ，可以将ϕ的范围限制在$[0,2\pi)$、$(-\pi,\pi]$或者其他长度为2π的半开区间内。

调整x、y表达式中的系数，就可以写出对称轴与坐标轴平行的椭圆参数方程：

$$\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} x_c+a\cos\phi \\ y_c+b\sin\phi \end{bmatrix}$$

2.7 三维隐式曲面

隐式方程隐式地确定了曲面上的点的集合：

$$f(x,y,z)=0$$

曲面上的任意一点(x, y, z)将使f的值为零，不在曲面上的点将返回非零值。由于是通过计算f的值来验证某点是否位于曲面上，而不是显式构造曲面上点的集合，因此称这种方式为隐式而不是显式。为了便于书写，关于$\boldsymbol{p}=(x, y, z)$的这类函数将采用下列方式表达：

$$f(\boldsymbol{p})=0$$

2.7.1 隐式曲面的法向量

在光学计算中会用到曲面的法向量，也就是垂直于曲面的向量。曲面上的每个点都对应独自的法向量。交叉点$\boldsymbol{p}$处的曲面法向量由隐式函数的梯度给出：

$$\boldsymbol{n}=\nabla f(\boldsymbol{p})=\left(\frac{\partial f(\boldsymbol{p})}{\partial x},\frac{\partial f(\boldsymbol{p})}{\partial y},\frac{\partial f(\boldsymbol{p})}{\partial z}\right)$$

梯度向量可以指向曲面内侧，也可以指向曲面外侧。如果某个f的梯度指向曲面内侧，而我们希望梯度指向曲面外侧，那么就采用$-f(\boldsymbol{p})=0$的形式。曲面$-f(\boldsymbol{p})=0$与曲面$f(\boldsymbol{p})=0$相同，只是二者的梯度方向相反，也就是$\nabla f(\boldsymbol{p})=-\nabla(-f(\boldsymbol{p}))$。

2.7.2 隐式平面

作为一个例子，考虑过点$\boldsymbol{a}$、法向量为$\boldsymbol{n}$的无限平面。该平面的隐式方程表示为

$$(\boldsymbol{p}-\boldsymbol{a})\cdot\boldsymbol{n}=0 \qquad (2\text{-}22)$$

38

注意$\boldsymbol{a}$和$\boldsymbol{n}$都是已知的量。点$\boldsymbol{p}$是满足方程的任意未知点。该方程的几何意义是：从$\boldsymbol{a}$到$\boldsymbol{p}$的向量与

平面的法向量垂直。如果p不在平面内，那么$(p-a)$就不与n成直角（见图2-34）。

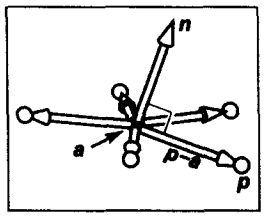


图2-34　如果式（2-22）成立，则图中所示p点都在平面内，平面的法向量n经过a点

有时需要建立过三点a、b、c的平面方程。可以通过求平面上任意两向量的叉积来求平面的法向量。例如：

$$n=(b-a)\times(c-a)$$

那么平面的隐式方程可写为

$$(p-a)\cdot((b-a)\times(c-a))=0$$

该方程的几何意义是：由$p-a$、$b-a$、$c-a$定义的平行六面体的体积为零，也就是说这三个向量共面。这一点只有当p与a、b、c在同一个平面上时才成立。完全笛卡儿行列式表示方式为（详细讨论见5.2.3节）：

$$\begin{vmatrix} x-x_a & y-y_a & z-z_a \\ x_b-x_a & y_b-y_a & z_b-z_a \\ x_c-x_a & y_c-y_a & z_c-z_a \end{vmatrix}=0 \qquad (2\text{-}23)$$

该行列式可以扩展到多项（见5.2.3节行列式扩展机制）。

式（2-22）与（2-23）是等价的，下面对它们进行一下比较。式（2-22）的几何意义容易解释，并能生成高效的代码。另外，利用式（2-22）可以避免打字错误，如果利用调试过的叉积和点积代码，打字错误将编译出不正确的代码。式（2-23）的几何意义也容易进行解释；如果实现了3 × 3的行列式函数，那么根据式（2-23）也能编写出高效代码。如果有现成函数*determinant*(a, b, c)可以调用，那么实现式(2-23)也很容易，并且不会产生打字错误。如果将函数名*determinant*改为*volume*，则其他人就更容易读懂代码。所以，对式（2-22）与（2-23）都很容易编写代码。任一方程的完全展开式都容易产生打字错误。这种错误有可能会通过编译，非常麻烦。这个例子充分说明，简洁的数学表示能使代码简洁，而复杂的数学表示容易使代码臃肿。

2.7.3　隐式曲面上的三维曲线

有人可能想用$f(p)=0$的形式表示一条隐式三维曲线。但这种形式只是退化的曲面，在实际中很少使用。三维曲线可以由两个隐式方程的交线确定：

39

$$f(p)=0$$
$$g(p)=0$$

例如，两隐式平面相交形成一条三维直线。一般用三维参数曲线更方便，三维参数曲线由二维参数曲线直接扩展而来。

2.8　三维参数曲线

三维参数曲线的表示形式与二维参数曲线类似：

$$x=f(t)$$
$$y=g(t)$$
$$z=h(t)$$

例如，绕z轴的螺旋线方程是

$$x=\cos t$$
$$y=\sin t$$
$$z=t$$

本章只详细讨论三维参数直线。一般的三维参数曲线将在第15章进行更深入的讨论。

三维参数直线

对二维参数直线进行扩展，可以得到三维参数直线：

$$x = 2+7t$$
$$y = 1+2t$$
$$z = 3-5t$$

这种形式显得不简洁，而且不容易编写代码。因此我们用向量的形式来表示：

$$\boldsymbol{p} = \boldsymbol{o}+t\boldsymbol{d}$$

40

其中，$\boldsymbol{o}$和$\boldsymbol{d}$是向量，在本例中它们可以这样给出：

$$\boldsymbol{o} = (2,1,3)$$
$$\boldsymbol{d} = (7,2,-5)$$

可以看出，这与二维的情况非常相似。想象这条直线过点$\boldsymbol{o}$，并且与$\boldsymbol{d}$平行。任给一个t值，就可以得到直线的点$\boldsymbol{p}(t)$。例如，在t=2处，$p(t) = (2, 1, 3) + 2(7, 2, -5) = (16, 5, -7)$。该定义与二维情况相同（见图2-30）。

与二维情况类似，线段可以用三维参数直线及某个区间$t \in [t_a, t_b]$来描述。$\boldsymbol{a}$、$\boldsymbol{b}$之间的线段表示为：$\boldsymbol{p}(t) = \boldsymbol{a} + t(\boldsymbol{b} - \boldsymbol{a})$，$t \in [0, 1]$。其中$\boldsymbol{p}(0) = \boldsymbol{a}$，$\boldsymbol{p}(1) = \boldsymbol{b}$，$\boldsymbol{p}(0.5) = (\boldsymbol{a} + \boldsymbol{b})/2$表示$\boldsymbol{a}$和$\boldsymbol{b}$的中点。

射线或者半直线，是半开区间（一般为$[0,\infty)$）上的三维参数直线。从现在起，我们将把所有的直线、线段、射线都称为光线（ray）。这样做有点不合适，但符合一般使用习惯，讨论起来也更方便。

2.9 三维参数曲面

另一种方法是通过二维参数确定三维曲面（三维空间中的曲面）。这时曲面具有下列形式：

$$x = f(u, v)$$
$$y = g(u, v)$$
$$z = h(u, v)$$

参数球面

地球表面上的一点由经度和纬度两个参数确定。如果把极坐标系置于半径为r的球体上，球心在原点（见图2-35），就得到球面的参数方程：

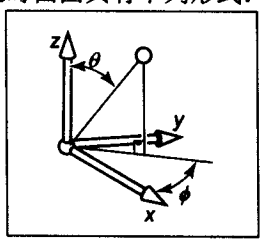


图2-35 球面坐标系的几何示意图

$$x = r\cos\phi\sin\theta$$
$$y = r\sin\phi\sin\theta \qquad (2\text{-}24)$$
$$z = r\cos\theta$$

我们希望用向量来表示。但是对于这样一种特殊的参数形式，向量表示是行不通的。采用怎样的θ、ϕ顺序，取决于大家的习惯经验。对于各角应采用什么样的表示符号，各学科并没有统一的标准。通常我们用$\boldsymbol{Ax}=\boldsymbol{b}$的方程形式，你也可以采用$\boldsymbol{bA}=\boldsymbol{x}$，其中$\boldsymbol{b}$是一个方阵，其他两个是列向量。对线性代数越熟悉，符号表示就变得越混乱。在图形学中，我们总采用式（2-24）中的θ和ϕ的含义。当绘制带纹理的球体时，我们还会用到这个公式。

41

已知某组(x, y, z)，我们还想求出对应的(θ,ϕ)。假设$\phi \in (-\pi, \pi]$，利用式（2-2）中的atan2函数很容易求出

$$\begin{aligned}\theta &= \operatorname{acos}(z/\sqrt{x^2+y^2+z^2}) \\ \phi &= \operatorname{atan2}(y,x)\end{aligned} \tag{2-25}$$

2.10　线性插值

图形学中最常用到的数学运算可能就是线性插值了。我们已经见过线性插值的例子：在二维或者三维情况，插入一个位置形成线段。由两点$\boldsymbol{a}$、$\boldsymbol{b}$构成的以t为参数的直线形式为$\boldsymbol{p}=(1-t)\boldsymbol{a}+t\boldsymbol{b}$，这就是插值，因为当$t=0$和$t=1$时，$\boldsymbol{p}$正好通过$\boldsymbol{a}$和$\boldsymbol{b}$。因为系数$t$和$1-t$是$t$的线性多项式，所以称为线性插值。

另一种常用的线性插值情况。对于x轴上的一组点：$x_1, x_2, \cdots, x_n$，每个x_i都有相应的高度y_i，我们想建立一个连续函数$y=f(x)$来插入这些点，使f能通过每一个数据点，即$f(x_i)=y_i$。在线性插值中，各点(x_i, y_i)由直线段相连。自然地，可以用参数直线方程来表示这些线段。参数t就是x_i和x_{i+1}之间的相对距离：

$$f(x) = y_i + \frac{x-x_i}{x_{i+1}-x_i}(y_{i+1}-y_i) \tag{2-26}$$

由于该加权函数是关于x的线性多项式，因此是线性插值。

以上两个例子都具有线性插值的一般形式。构造一个变量t，随着数据项A变化到数据项B，t从0变化到1。中间值就是函数$(1-t)A+tB$。可以看出式（2-26）就具有这种形式，其中

42

$$t=\frac{x-x_i}{x_{i+1}-x_i}$$

2.11　三角形

不管是二维情况还是三维情况，三角形都是图形学程序中的基本图形。一般像颜色这样的信息标记在三角形的顶点上，并且被插值到三角形当中。实现这种插值的坐标系称为重心坐标系，我们将进行较详细的讨论。我们也讨论二维三角形，在二维屏幕上绘制之前应该对二维三角形有所理解。

2.11.1　二维三角形

由二维点$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$确定的二维三角形，其面积为

$$\begin{aligned}\text{面积} &= \frac{1}{2}\begin{vmatrix} x_b-x_a & x_c-x_a \\ y_b-y_a & y_c-y_a \end{vmatrix} \\ &= \frac{1}{2}(x_a y_b + x_b y_c + x_c y_a - x_a y_c - x_b y_a - x_c y_b)\end{aligned} \tag{2-27}$$

该公式出自5.2.3节的推导过程。如果点$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$是逆时针顺序，得到的面积为正，反之为负。

在图形学中，经常希望为三角形的每个顶点赋以性质，如颜色，并将该性质的值插值到三角形中。实现这种要求有多种方法，其中最简单的方法是利用重心坐标系。一种方式是像2.4.2节中讨论的那样，把重心坐标系看作非直角坐标系。如图2-36所示的坐标系，其坐标原点为$\boldsymbol{a}$，从$\boldsymbol{a}$到

$\boldsymbol{b}$和$\boldsymbol{c}$的向量为基向量。于是点$\boldsymbol{p}$就可以表示为

$$\boldsymbol{p}=\boldsymbol{a}+\beta(\boldsymbol{b}-\boldsymbol{a})+\gamma(\boldsymbol{c}-\boldsymbol{a}) \tag{2-28}$$

我们可以重新安排式（2-28）中的各项顺序，得到

$$\boldsymbol{p}=(1-\beta-\gamma)\boldsymbol{a}+\beta\boldsymbol{b}+\gamma\boldsymbol{c}$$

经常定义一个新的变量α，以改善方程的对称性：

$$\alpha \equiv 1-\beta-\gamma$$

于是得到方程

$$\boldsymbol{p}(\alpha,\beta,\gamma)=\alpha\boldsymbol{a}+\beta\boldsymbol{b}+\gamma\boldsymbol{c} \tag{2-29}$$

43

其中

$$\alpha+\beta+\gamma=1 \tag{2-30}$$

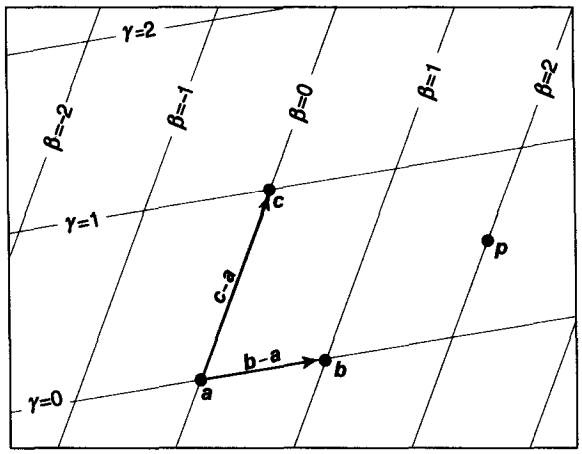


图2-36 可以借助顶点为$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$的二维三角形，建立非直角坐标系，其原点为$\boldsymbol{a}$，基向量为向量$(\boldsymbol{b}-\boldsymbol{a})$、$(\boldsymbol{c}-\boldsymbol{a})$。于是就可以用有序对$(\beta,\gamma)$来表示点。例如，点$\boldsymbol{p}$=(2.0,0.5)，也就是$\boldsymbol{p}=\boldsymbol{a}+2.0(\boldsymbol{b}-\boldsymbol{a})+0.5(\boldsymbol{c}-\boldsymbol{a})$

初看起来，重心坐标系好像抽象、不直观，但实际上它功能强大而且方便使用。考虑城市街道分布情况，有两组街道，每组内的道路是互相平行的，但是两组街道之间交叉而不成直角，这时用重心坐标系就比较方便。自然系统就是重心坐标系，大家很快就会习惯使用。重心坐标系由平面上的所有点确定。重心坐标系的一大特点是，对于由$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$形成的三角形，当且仅当下列条件满足时，点$\boldsymbol{p}$位于三角形的内部：

$$0<\alpha<1$$
$$0<\beta<1$$
$$0<\gamma<1$$

如果其中一个坐标为零，而其他两个坐标位于0和1之间，那么该点位于三角形边上。如果有两个坐标为0，剩下的一个为1，则该点位于顶点。另一个优点是，式（2-29）实际上采用一种平滑的方式，使三个顶点的坐标相加。可以用相同的系数(α,β,γ)计算其他性质，如颜色，在下一章我们会看到这些内容。

已知点$\boldsymbol{p}$，如何计算它的重心坐标？一种方法是将式（2-28）写成关于未知量β、γ的线性方程，然后求解，并令$\alpha=1-\beta-\gamma$。线性方程为

44

$$\begin{bmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x_p - x_a \\ y_p - y_a \end{bmatrix} \tag{2-31}$$

尽管可以用代数的方法直接解式（2-31），但用几何方法求解常常更方便。

重心坐标的一个几何特性是，它们表示到三角形边所在直线的有向距离，且距离的大小具有一定的比例。关于重心坐标β，如图2-37所示。在2.5.2节讲过，相对于直线$f(x,y)=0$，函数$f(x,y)$的返回值表示从点(x,y)到该直线的不同比例的有向距离。还讲过，如果$f(x,y)=0$是某条直线的方程，那么对于非零实数k，$kf(x,y)=0$也是这条直线的方程。改变k的值可以改变距离的大小，也可以控制直线的某一侧距离为正，而另一侧距离为负。例如，我们可以选择满足$kf(x,y)=\beta$的k值。因为k是唯一的未知量，我们可以对它施加约束，即在b点令$\beta=1$。这样，如果直线$f_{ac}(x,y)=0$穿过a、c两点，那么就可以计算点(x,y)的β值，具体如下：

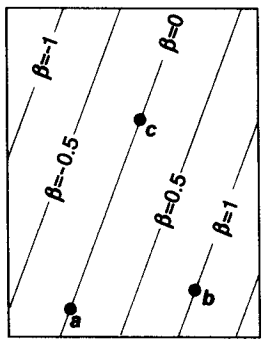


图2-37　重心坐标β，表示到直线ac的不同比例的有向距离

$$\beta = \frac{f_{ac}(x,y)}{f_{ac}(x_b,y_b)} \tag{2-32}$$

可以用类似方法计算α、γ。一般先只计算重心坐标中的两个，然后通过式（2-30）求第三个，这样可以提高计算效率。

为了求通过两点$\boldsymbol{p}_0$和$\boldsymbol{p}_1$的直线的理想形式，可以首先采用2.5.2节介绍的方法，求出通过顶点的有效隐式直线。根据式（2-18），可以得出

$$f_{ab}(x,y) \equiv (y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a = 0$$

注意$f_{ab}(x_c,y_c)$很可能不等于1，因此这很可能不是我们想要的理想形式。如果除以$f_{ab}(x_c,y_c)$，我们得到

$$\gamma = \frac{(y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a}{(y_a - y_b)x_c + (x_b - x_a)y_c + x_a y_b - x_b y_a}$$

由于引入了除法，我们会担心出现零除数的情况。但对于面积不接近零的三角形，零除数的情况不会出现。类似可得出关于α、β的表达式，但一般只需要再算出一个即可：

$$\beta = \frac{(y_a - y_c)x + (x_c - x_a)y + x_a y_c - x_c y_a}{(y_a - y_c)x_b + (x_c - x_a)y_b + x_a y_c - x_c y_a}$$

$$\alpha = 1 - \beta - \gamma$$

45

另一种计算重心坐标的方法是，计算图2-38中所示的子三角形A_a、A_b、A_c的面积。重心坐标服从以下法则：

$$\begin{aligned} \alpha &= A_a / A \\ \beta &= A_b / A \\ \gamma &= A_c / A \end{aligned} \tag{2-33}$$

其中A表示三角形的面积。因为有$A = A_a + A_b + A_c$，因此可以通过两次加法运算算出总面积，而不使用求总面积的公式。如果面积可以有正负，那么对于三角形之外的点，上述法则同样适用。具体原因见图2-39。注意这些面积都是带符号的，只要三角形面积A和子三角形面积A_a、A_b、A_c都是有向面积，计算结果将会是正确的。

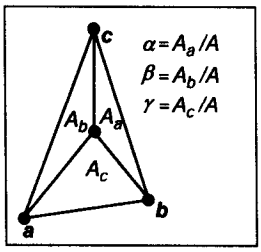


图2-38 重心坐标与图中所示的子三角形面积成正比

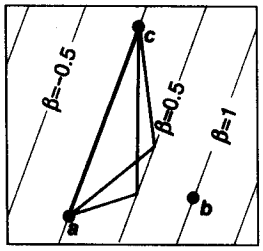


图2-39 图中所示的两个三角形，由于同底等高，因此具有相等的面积。对于顶点位于直线$\beta = 0.5$上的所有三角形，面积都相等。因为高度与β成正比，所以三角形面积也与β成正比

2.11.2 三维三角形

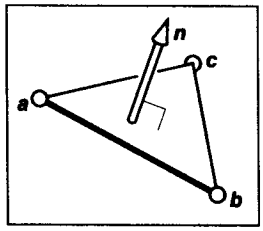


图2-40 三角形的法向量，垂直于三角形所在平面上的所有向量，因此它垂直于三角形的各边

三角形的重心坐标可以扩展到三维空间。假设点$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$是三维点，我们仍然可以使用下式：

$$\boldsymbol{p} = (1-\beta-\gamma)\boldsymbol{a} + \beta\boldsymbol{b} + \gamma\boldsymbol{c}$$

现在，通过改变β和γ，形成一个平面。

计算三角形所在平面上的任意两个向量的叉积，就可以得到三角形的法向量（见图2-40）。用三角形的两边构造这样的向量最方便，例如：

$$\boldsymbol{n} = (\boldsymbol{b}-\boldsymbol{a}) \times (\boldsymbol{c}-\boldsymbol{a}) \tag{2-34}$$

注意法向量不一定具有单位长度，它服从叉积的右手法则。

可以通过叉积长度来表示三角形的面积：

$$\text{面积} = \frac{1}{2}\left\|(\boldsymbol{b}-\boldsymbol{a}) \times (\boldsymbol{c}-\boldsymbol{a})\right\| \tag{2-35}$$

注意这不是带符号的面积，所以不能直接用来计算重心坐标。然而，通过观察我们发现，取顺时针顶点顺序时的三角形法向量，与取逆时针顶点顺序时的法向量方向相反。我们知道

$$\boldsymbol{a} \cdot \boldsymbol{b} = \|\boldsymbol{a}\|\|\boldsymbol{b}\|\cos\phi$$

46

其中ϕ是向量之间的夹角。如果$\boldsymbol{a}$和$\boldsymbol{b}$平行，则$\cos\phi = \pm 1$。我们可以据此判断两向量的方向是一致还是相反。联合式（2-33）、（2-34）、（2-35），可以得到

$$\alpha = \frac{\boldsymbol{n} \cdot \boldsymbol{n}_a}{\|\boldsymbol{n}\|^2}$$

$$\beta = \frac{\boldsymbol{n} \cdot \boldsymbol{n}_b}{\|\boldsymbol{n}\|^2}$$

$$\gamma = \frac{\boldsymbol{n} \cdot \boldsymbol{n}_c}{\|\boldsymbol{n}\|^2}$$

其中$\boldsymbol{n}$通过顶点$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$利用式（2-34）求出，$\boldsymbol{n}_a$通过顶点$\boldsymbol{b}$、$\boldsymbol{c}$、$\boldsymbol{p}$求出，等等，也就是：

$$\begin{aligned} \boldsymbol{n}_a &= (\boldsymbol{c}-\boldsymbol{b}) \times (\boldsymbol{p}-\boldsymbol{b}) \\ \boldsymbol{n}_b &= (\boldsymbol{a}-\boldsymbol{c}) \times (\boldsymbol{p}-\boldsymbol{c}) \\ \boldsymbol{n}_c &= (\boldsymbol{b}-\boldsymbol{a}) \times (\boldsymbol{p}-\boldsymbol{a}) \end{aligned} \tag{2-36}$$

常见问题

- **为什么没有向量除法？**

向量除法尚不存在理想的类比意义。但是详细研究这个问题，可能会促使四元数的提出（见本章提到的Hoffmann著作）。

- **三角形的重心坐标表达很清楚，对边数大于3的多边形有没有类似的表示方法？**

很不幸，没有。即使是凸四边形，也比三角形复杂得多。这也是把三角形作为图形学中基本几何图形的原因之一。

- **有没有三维直线的隐式形式？**

没有。但是两个三维平面的交线确定一条三维直线，因此一条三维直线可以同时由两个隐式
47 三维方程来描述。

文献注记

向量分析的发展历史非常有趣。在很大程度上，向量分析是由Grassman在19世纪中期提出的，但是没有受到重视，后来又被重新提出来（Crowe，1994）。在图形学领域，Grassman的追随者正在基于他的一些思想，推动着几何代数学的发展（Doran & Lasenby，2003）。让读者感兴趣的是，为什么这种特殊的标量和向量积在某种意义上是正确的，以及为什么没有一种通用的向量除法，关于这些问题读者可以在*About Vectors*（Hoffmann，1975）一书中找到启示。另一种重要的几何工具是*四元数*，是由Hamilton在19世纪中叶提出的。四元数在很多情况下都非常有用，尤其当涉及方位问题时（Hanson，2005）。

习题

1. 集合的*基数*指集合包含的元素个数。在IEEE浮点表示（见1.6节）中，浮点数集合的基数是多少？
2. 要实现这样一个函数：把32位整数映射到64位整数，并且存在反函数，可能吗？所有映射32位整数到64位整数的函数都存在反函数吗？
3. 用三个区间的笛卡儿积，表示单位立方体（x、y、z坐标都在0到1之间）。
4. 如果你有自然对数函数$\ln(x)$，那么如何用它表示以b为底的对数函数$\log(b,x)$？如果b为负数怎么办？实现时要用IEEE浮点表示。
5. 解二次方程$4x^2-6x+9=0$。
6. 实现由二次方程$Ax^2+By+C=0$确定的函数，输入系数A、B、C，并求这个方程的两个根。要求函数返回有效根（不是NaN）的个数，并将根返回，数值小的根排在前面。
48 7. 举反例证明，对于三维向量$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$，等式$\boldsymbol{a}\times(\boldsymbol{b}\times\boldsymbol{c})=(\boldsymbol{a}\times\boldsymbol{b})\times\boldsymbol{c}$不总是成立。
8. 已知两个不平行的向量$\boldsymbol{a}$和$\boldsymbol{b}$，计算满足右手法则的标准正交基，使得$\boldsymbol{u}$与$\boldsymbol{a}$平行，且$\boldsymbol{v}$位于$\boldsymbol{a}$和$\boldsymbol{b}$所在的平面。
9. $f(x,y,z)=x^2+y-3z^3$的梯度是什么？
10. 对称轴与坐标轴平行的二维椭圆的参数形式是什么？
11. 过三点(1,0,0)、(0,1,0)、(0,0,1)的平面的隐式方程是什么？参数方程是什么？平面的法向量是什么？
12. 已知四个二维点$\boldsymbol{a}_0$、$\boldsymbol{a}_1$、$\boldsymbol{b}_0$、$\boldsymbol{b}_1$，编写程序判断线段$\boldsymbol{a}_0\boldsymbol{a}_1$和$\boldsymbol{b}_0\boldsymbol{b}_1$是否相交。
49 13. 编程计算三个不在同一直线上的二维点的重心坐标。

第3章

光栅算法

多数计算机图形图像，是通过光栅显像显示给用户的。这种系统将图像作为像素阵列进行显示，像素（pixel）即图像元素（picture element）的简称。这些像素采用RGB颜色空间。本章讨论光栅显像的基本原理，着重讨论RGB颜色系统和标准图像显示器存在的非线性。

3.1 光栅显像

台式机和投影显示器的显示技术有多种。这些显示器的分辨率（像素数量）和物理尺寸各不相同。编程人员通常假设像素排成矩形阵列，又称为光栅（raster）。

像素

光栅显示器上的可显示元素称为像素。在显示器上，通常用有序对(i, j)来表示像素的索引，即表示该像素所在的行与列。如果一台显示器具有n_x行、n_y列像素，则左下角的元素是像素$(0, 0)$，右上角的元素是像素(n_x-1, n_y-1)。[1]

我们需要用实际的二维屏幕坐标来表示像素的位置。随着API的不同，这些系统在细节上会有所不同，但是最常使用的是用整数点阵作为像素的中心，如图3-1中的4×3屏幕所示。由于像素占据一定的空间区域，所以距离像素中心具有0.5个单位的过冲（overshoot）。

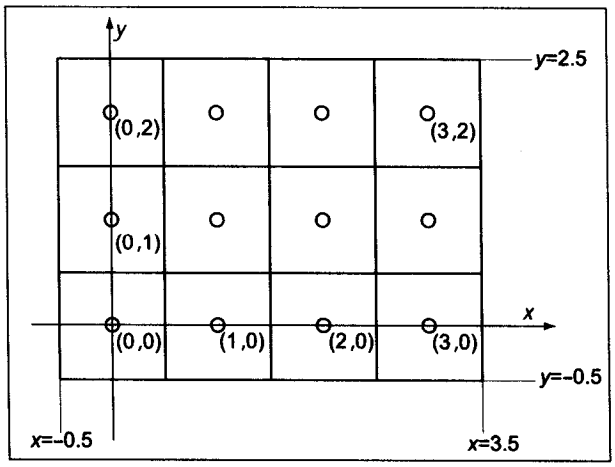


图3-1 4像素×3像素的屏幕的坐标系。注意，在某些API中，y轴向下

1. 在许多API中，图像的行用不太直观的“自上到下”的方式来标识，因此左上角的像素坐标为(0,0)。这种习惯有其历史原因，这种行扫描顺序符合电视传输标准。

物理像素，就是硬件上实际可显像的元素。不同的显示系统采用不同的物理像素。在CRT（阴极射线管）中，像素与CRT中的一小片荧光物质对应。这种荧光物质的发光强度与电子束的激发程度有关。电子束如何扫过该像素，以及荧光点在屏幕上如何分布，这些因素都会影响像素的形状。首先，可以认为荧光点在屏幕上呈水滴状，即在像素中间密度最大，向边缘方向密度逐渐减小。在LCD（液晶显示器）中，像素近似为方形滤波器，通过调整其不透明性从而挡住后面的光线。这些像素几乎都是正方形，并且各正方形之间有一段小间隙，这样可使控制电路获得像素信息。多数显示系统都和CRT或LCD类似，像点呈水滴状或者呈方形。

3.2　显示器亮度和γ值

现在所有的显示器都采用数字信号输入方式，接收的是用数字信号表示的像素“值”，然
52 后将其转化为亮度值。断电后显示器的亮度实际上不是零，因为屏幕能够反射环境光线。但我们可以认为此时显示器呈“黑色”，显示器完全打开时呈“白色”。对像素颜色用0到1的数值来表示。黑色为0，白色为1，介于黑白中间的中间灰色为0.5。注意这里“中间”指的是从像素发出光线的强度，而不是指观察到的亮度。人类对光强的感知呈非线性，这不是我们现在讨论的内容。

为了在显示器上产生图像，要先明白两个关键问题。第一，显示器对输入信号的处理是非线性的。例如，如果输入三个像素值0、0.5、1.0，显示的亮度则可能是0、0.25、1.0（也就是0、打开1/4、完全打开）。对于多数显示器，一般利用γ值来近似表示其非线性，具体数值就是下面公式中的指数：

$$\text{显示亮度}=(\text{最大亮度})\,a^{\gamma} \tag{3-1}$$

其中a是介于0到1之间的输入亮度值。例如，如果显示器的γ值为2.0，输入值为$a=0.5$，则显示亮度是最大亮度的1/4，因为$0.5^2=0.25$。不管γ值为多少，$a=0$对应显示亮度为0，$a=1$对应最大显示亮度。用γ值表示显示器的非线性只是一阶近似，实际中在估计设备的γ值时不需要很高的精度。一种度量非线性的直观方法是，找到能产生黑白之间的中间亮度的a值，即下式中的a值：

$$0.5=a^{\gamma}$$

如果能找到满足上式的a值，则通过两边取对数就可以求出γ值：

$$\gamma=\frac{\ln 0.5}{\ln a}$$

标准求a值的方法是，显示黑白相间的像素棋盘模式，旁边是具有a值的灰色像素方阵（见图3-2）。当从远处观察这幅图像时（或者当你眼睛近视但没带眼镜时），如果a值正好是黑白的中间值，那么这幅图像的左右两边看起来会大致一样。这是因为，虽然存在很多的黑、白像素，但经棋盘模式混合，整体效果就显示一致的中间色了。为了做好这个实验，必须尝试很多a值，直到
53 找到需要的数值为止。可以为用户提供一个滑动条来控制a值，或者将多个不同灰度的方块同时与一个大的棋盘模式比较。值得注意的是，CRT沿水平方向快速改变亮度是有难度的，因此水平的黑白条纹要比棋盘模式效果好。利用人脸识别技术可使亮度匹配结果更加精确（Kindlmann，Reinhard, & Creem，2002）。

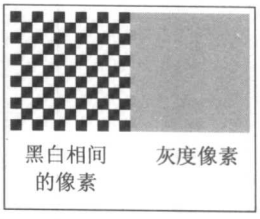


图3-2　黑白相间的棋盘模式，从远处看表现为介于黑白之间的中间色。通过寻找看起来与黑白模式有相同效果的灰度值，来推出显示器的γ值

一旦知道了γ值，就可以对输入进行伽马校正，使得输入值$a=0.5$可以在屏幕上显示出介于

黑白中间的灰度效果。变换方法为

$$a = a^{\frac{1}{\gamma}}$$

把该式代入式（3-1），得

$$\text{显示亮度} = \left(a^{\frac{1}{\gamma}}\right)^{\gamma} \text{（最大亮度）} = a \text{（最大亮度）}$$

实际显示器的另一个重要特征是，输入值通常经过了量化处理。因此，我们可以在浮点范围[0,1]内处理亮度值。输入到显示器的信息一般是大小固定的非负整数，取值范围一般是0~255，可采用8位二进制数存储。所以a的取值不是[0,1]中的数字，而是

$$a\text{的取值} = \left\{\frac{0}{255}, \frac{1}{255}, \frac{2}{255}, \cdots, \frac{254}{255}, \frac{255}{255}\right\}$$

则显示的亮度值近似为

$$\left\{M\left(\frac{0}{255}\right)^{\gamma}, M\left(\frac{1}{255}\right)^{\gamma}, M\left(\frac{2}{255}\right)^{\gamma}, \cdots, M\left(\frac{254}{255}\right)^{\gamma}, M\left(\frac{255}{255}\right)^{\gamma}\right\}$$

M表示最大亮度。实际应用需要精确控制亮度值，我们要分出256个可能的亮度等级，而且这些亮度在屏幕的不同位置其值也有所不同，尤其是CRT显示器更需要这样做。视角不同也会影响亮度值。幸运的是，实际系统一般不需要做精确标定。

3.3 RGB 颜色

计算机图形学中，多数显示效果都由红绿蓝（RGB）颜色空间确定。RGB颜色空间比较简单，经转换能够直接控制多数计算机屏幕的显示。本节内容从用户观察的角度讨论RGB颜色空间，目的是为了方便使用。第20章会对颜色做更详细的讨论。了解了RGB颜色空间的机制，我们就可以编写图形学中的大部分程序。RGB颜色空间的基本思想是，将红、绿、蓝三种基色光混合产生新的颜色。三种光以相加方式混合，加性颜色混合与常见的减性颜色混合不同，后者控制着颜料和画笔。在我们常见的媒体中，红、黄、蓝是基色，可以采用我们熟悉的方式对它们进行混合。例如，黄色和蓝色相混合得到绿色。对于RGB加性颜色空间，我们有下列混合效果（见图3-3）：

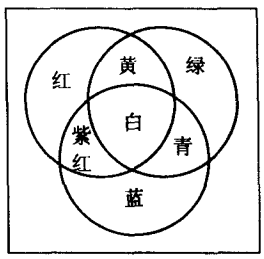


图3-3 红绿蓝三基色的加性混合法则

54

红色 + 绿色 = 黄色
绿色 + 蓝色 = 青色
蓝色 + 红色 = 紫红色
红色 + 绿色 + 蓝色 = 白色

“青色”是一种蓝绿色，“紫红色”是一种紫色。

如果让基色光的强度由最小到最大变化，就能在RGB显示器上显示所有的颜色。根据惯例，把单色的最亮状态表示为1，则颜色值为0~1之间的分数。这样就产生出一种三维RGB颜色立方体，红、绿、蓝为坐标轴，坐标值从0到1变化。颜色立方体如图3-4所示。

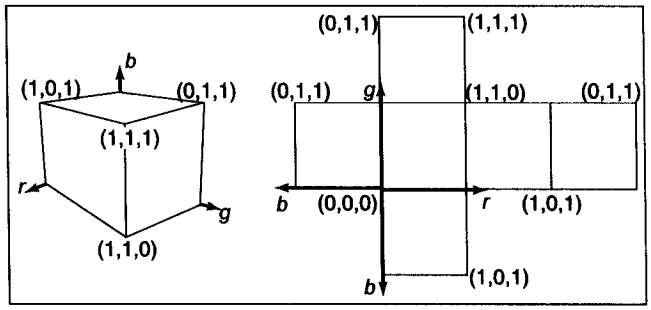


图3-4　三维RGB颜色立方体的三维形式及其展开面。任何RGB颜色都是立方体中的一个点（另见彩图I）

常见的RGB颜色坐标为：

黑色=(0,0,0)
红色=(1,0,0)
绿色=(0,1,0)
蓝色=(0,0,1)
黄色=(1,1,0)
紫红=(1,0,1)
青色=(0,1,1)
白色=(1,1,1)

实际RGB数值以量化形式给出，类似3.2节讨论的灰度级，各成分都用整数表示。常用的整数大小占一个字节，所以RGB的各成分是0~255之间的一个整数。三个整数共占用三个字节，也就是24位二进制数。因此，具有“24位颜色”的系统，三基色中的每一种都有256个等级。在3.2
55 节讨论的伽马校正问题，也分别适用于RGB的三个成分。

3.4　α通道

我们经常想只覆盖像素的部分内容。常见的例子是，在合成图像时，我们想在背景上面插入一幅前景图片。对于前景中的不透明像素，我们只要用它们替换相应位置的背景像素即可。对于完全透明的前景像素，背景像素不再改变。而对于半透明像素，必须认真对待。当前景物体中有
56 像玻璃这样的半透明的区域时，或者在前景物体中有子像素孔洞（如远处的树叶）时，就会出现半透明的像素。在有像素孔洞的情况下，为了将前景与背景混合，就要权衡像素作为前景的分数（fraction）。该分数用α表示。如果想把前景色c_f与背景色c_b混合，并且被前景覆盖的像素的分数为α，那么就可以采用下面的公式计算：

$$c = \alpha c_f + (1-\alpha)c_b \qquad (3\text{-}2)$$

式（3-2）的使用效果见图3-5。需要注意的是，图中的α图像可以存储为RGB图像，也可以存储为单通道的灰度图像。

尽管式（3-2）的使用率很高，但在很多情况下α以其他方式得到使用（Porter & Duff, 1984）。

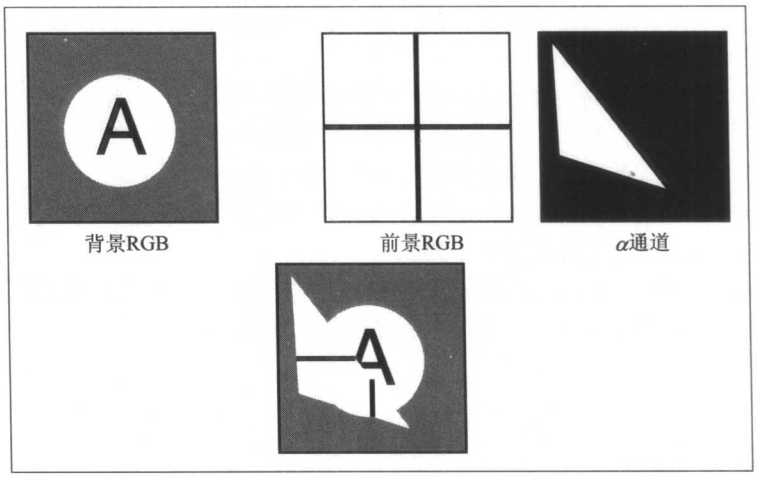


图3-5 利用式（3-2）合成图像的示例。实际上，在前景图像与背景图像混合之前，先经α通道进行了修剪处理。最下面是组合效果图

3.5 直线绘制

多数图形软件都包含直线绘制命令，即捕捉屏幕上两端点的坐标（见图3-1），并在它们之间画一条直线。例如，如果发现两个端点是(1, 1)和(3, 2)，那么就在屏幕上显示这两点，并在它们之间填上一个像素。对一般的屏幕坐标端点(x_0，y_0)和(x_1，y_1)，常规作法是在它们之间画出合适的像素点集，这些点近似一条直线。为了简化处理，x_0、x_1、y_0、y_1的值经常规定为整数（像素中心），因为直线本身就比较粗略，要求子像素精度不合适。在实现API时需要的是实数端点坐标，一般方法是将它们就近取整，这样做比较合理，一般应用编程人员注意不到有什么不同。因为端点坐标是整数，在整数与浮点变量共同参与运算时，就要注意隐式类型转换问题。我们基于直线方程进行直线绘制。有两种类型的方程可供选择：隐式方程和参数方程。下面讲解基于这两种方程的两种算法。

3.5.1 基于隐式方程绘制直线

基于隐式方程绘制直线的最常用算法是*中点算法*（Pitteway(1967)；Van Aken & Novak (1985)）。中点算法结束时绘制的直线与Bresenham*算法*（Bresenham，1965）的结果一样，只是中点算法更加直接。

57

首先要找到如2.5.2节介绍的直线隐式方程：

$$f(x, y) \equiv (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0 \tag{3-3}$$

假设$x_0 \leqslant x_1$。如果不满足该条件，就交换点的顺序使之成立。直线的斜率为

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

在下面的讨论中都假设$m \in (0,1]$。类似地，可以推出$m \in (-\infty,-1]$、$m \in (-1,0]$和$m \in (1,\infty)$时的结果。这4种情况覆盖了所有的可能情况。

当$m \in (0,1]$时，直线在x轴上的变化速度大于在y轴上的变化速度。对于y轴方向向下的API，我们可能会想到情况是否变得复杂。实际上我们可以忽略这个细节。可以不考虑几何上的“升”和“降”，因为两种情况下的代数表示是一样的。认真的读者可以证明，得到的算法同样适用于y

轴向下的情况。中点算法的重要假设是，我们能绘出没有间隔的最细的直线。两对角像素之间的连接被认为不产生间隔。

绘制直线时，从左端点向右端点进行。只有两种可能：绘制一像素时，和左边所绘像素高度一样，或者高出一个像素。在两端点之间每一列，总是只有一个像素。没有像素就意味着有间隔，有两个像素则直线太粗。就我们正讨论的情况来说，同一行上可以有两个像素；因为直线更接近于水平，因此其走势是向右或者向上。这种认识如图3-6所示，图中显示3条这样的直线，每一条在水平方向的前进速度要比在垂直方向上快。

针对$m\in(0,1)$情况的中点算法，首先建立最左边的像素，以及最右边像素的列号（x值），然后沿水平循环建立每个像素的行号（y值）。该算法的基本形式如下：

```
y = y0
for x = x0 to x1 do
  draw(x,y)
  if(some condition)then
    y=y+1
```

其中的x和y都是整数。简单地说就是“从左至右不断绘制像素，在该过程中有时需要向y轴正向
58 有所偏移”。关键是如何确定if语句中的条件。

一种有效方法是，参考两候选像点之间的中点位置。更具体地说，对于刚绘制的像素点(x, y)，它在屏幕上的坐标为(x, y)，下面要绘制的候选像素为$(x + 1, y)$和$(x + 1, y + 1)$。两候选像点之间的中点为$(x + 1, y + 0.5)$。如果直线通过中点的下方，就绘制下面的像素；如果直线通过中点的上方，就绘制上面的像素（见图3-7）。

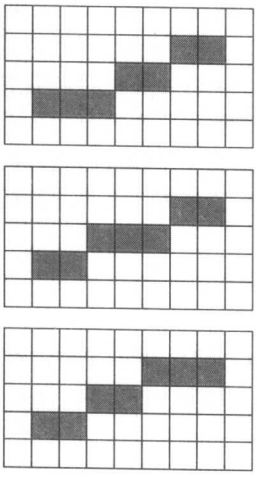


图3-6　3条合理直线，水平跨过7个像素，垂直跨过3个像素

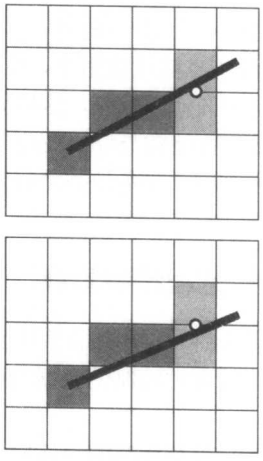


图3-7　上部：直线通过中点的上方，所以绘制上面的像素。下部：直线通过中点的下方，所以绘制下面的像素

判断直线是通过点$(x + 1, y + 0.5)$的上方还是下方，可以通过式（3-3）中$f(x, y + 0.5)$的值来判断。在2.5节中我们讨论过，位于直线上的点(x, y)，满足直线方程$f(x, y)=0$；位于直线一边的点，满足 $f(x, y)>0$；位于直线另一边的点，满足 $f(x, y)<0$。由于$-f(x, y)=0$和 $f(x, y)=0$都可以作为直线的方程，所以很难利用$f(x, y)$的正负来快速判断(x, y)究竟位于直线的上、下哪个方向。但是，可以通过分析算出来。式（3-3）中的关键项是关于y的项$(x_1 - x_0)y$。要注意$(x_1 - x_0)$肯定为正数，因为$x_1 > x_0$。这就意味着随着y的增大，$(x_1 - x_0)y$项的值将变大。因此$f(x, +\infty)$肯定为正，并且位于直线的上方，也就是说直线上方的点都是正的。另一种判断方法是，由于梯度向量中的y分量

是正的，因此在直线的上方y可以任意增加，$f(x, y)$也就肯定是正的。于是，我们可以把if语句中的条件写具体，代码就更明确了：

```
if f(x+1, y+0.5)<0 then
  y=y+1
```

对于有适当斜率（介于0和1之间）的直线，上面的代码非常有效。读者可以对另外三种情况进行推导，它们只在一些小细节上与第一种情况不同。

如果希望有更高的效率，就使用增量方法。增量方法利用上一步的计算结果，使循环更加高效。在上述中点算法中，主要步骤是计算$f(x+1, y+0.5)$的值。可以看出，在循环内部，经过第一次迭代，我们已经算出$f(x-1, y+0.5)$或者$f(x-1, y-0.5)$的值（见图3-8）。另外，存在如下关系：

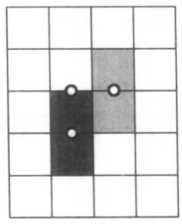


图3-8 在使用两个浅灰像素之间显示的决策点时，我们恰好画两个深灰像素中的一个，所以我们在左边显示的两个点中的一个点上计算f

$$f(x+1, y) = f(x, y) + (y_0 - y_1)$$

$$f(x+1, y+1) = f(x, y) + (y_0 - y_1) + (x_1 - x_0)$$

于是增量算法的代码如下：

```
y = y0
d = f(x0+1, y0+0.5)
for x = x0 to x1 do
  draw(x,y)
  if d < 0 then
    y=y+1
    d = d+(x1–x0)+(y0–y1)
  else
    d = d+(y0–y1)
```

与非增量算法相比，该增量算法需要的额外调整很少，因此具有更高的运行效率(不是所有的增量算法都这样)，但是对于较长的直线，在计算 $f(x, y + 0.5)$的值时包括很多加法运算，结果使数值误差得以积累。不过由于直线很少会长于几千像素，这种误差一般关系不大。如果把$(x_1 - x_0)+(y_0 - y_1)$和$(y_0 - y_1)$作为变量存储，就会增加额外调整次数，但循环代码会执行得更快。一个好的编译器也许可以帮助我们做到这一点，但是如果代码存在风险，最好还是认真检查编译结果，以免出错。

有时，如果只有整数参加运算，算法执行起来会更快。由于已经规定 x_0、x_1、y_0、y_1为整数，上面的算法也就是整数运算了。但是，初始化时还是要用到$d=f(x_0+1, y_0+0.5)$。将该表达式进行扩展，得到

$$f(x_0 + 1, y_0 + 0.5) = (y_0 - y_1)(x_0 + 1) + (x_1 - x_0)(y_0 + 0.5) + x_0 y_1 - x_1 y_0$$

其中$y + 0.5$项不是整数运算，结果是非整数之间做乘法运算。我们已经知道：如果 $f(x, y)=0$是直线方程，那么$2f(x, y)=0$也是同一直线的方程。因此，我们可以用$2f(x, y)$代替$f(x, y)$，于是d的表达式就变成

$$2f(x_0 + 1, y_0 + 0.5) = 2(y_0 - y_1)(x_0 + 1) + (x_1 - x_0)(2y_0 + 1) + 2x_0 y_1 - 2x_1 y_0$$

上面的所有项都成了整数项，算法代码变为

```
y = y0
d=2(y0–y1)(x0+1)+(x1–x0)(2y0+1)+2x0y1–2x1y0
for x = x0 to x1 do
  draw(x,y)
  if d < 0 then
```

```
    y=y+1
    d = d+2(x1-x0)+2(y0-y1)
  else
60  d = d+2(y0-y1)
```

细心的读者会注意到，在执行上面的全整数运算代码之前，必须检查$m \in [0,1)$是否满足，并且在计算m时需要进行除法运算。可以采用下面的代码来判断斜率的取值范围，从而避免除法运算：

$$((y_1 \geqslant y_0)\text{and}(x_1 - x_0 > y_1 - y_0)) \equiv (m \in [0,1))$$

3.5.2 基于参数方程绘制直线

我们在2.6.1节讨论过，过两点$\boldsymbol{p}_0=(x_0, y_0)$和$\boldsymbol{p}_1=(x_1, y_1)$的直线参数方程为

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + t(x_1 - x_0) \\ y_0 + t(y_1 - y_0) \end{bmatrix}$$

等价的向量形式为

$$\boldsymbol{p}(t) = \boldsymbol{p}_0 + t(\boldsymbol{p}_1 - \boldsymbol{p}_0)$$

如果直线斜率$m \in [-1,1]$，那么for循环就可以沿x轴继续执行，而且代码很简洁。对于$[-1,1]$之外的m，可得到类似的算法。因此只需要考虑两种情况。因为t沿直线推进固定的长度，直线的x和y分量也一样，这样就可以把t看作x的函数：

$$t = \frac{x - x_0}{x_1 - x_0}$$

得到的代码为

```
for x = x0 to x1 do
  t = (x-x0)/(x1-x0)
  y = y0+t(y1-y0)
  draw(x,round(y))
```

该算法的一个优点是，不论$x_1 > x_0$是否成立，算法都能正常运行。可以看出，基于直线参数方程的代码非常简单。然而，我们将会看到，不能将代码设计为只处理整数。在很多计算机语言中，从浮点数到整数的转换都采用截断的方式实现，因此，round(y)项可作为(y+0.5)的取整结果。

61　因为在每次迭代中t和y的变化都是一个常量，因此可采用增量算法：

```
Δy = (y1 - y0)/(x1 - x0)
y = y0
for x = x0 to x1 do
  draw(x,round(y))
  y = y + Δy
```

有时用两端点的RGB颜色$\boldsymbol{c}_0$和$\boldsymbol{c}_1$来确定直线的颜色。我们希望沿着直线颜色能够均匀地改变。如果想根据$t \in [0,1]$对线段进行参数化，则可用下面的公式：

$$\boldsymbol{c} = (1-t)\,\boldsymbol{c}_0 + t\boldsymbol{c}_1$$

这样就能算出每个像素的颜色。如图3-9所示，彩色直线从红色逐渐变到绿色。如果愿意，计算$\boldsymbol{c}$和t的表达式也可以是增量方式。代码如下：

```
Δy = (y1 - y0)/(x1 - x0)
Δr = (r1 - r0)/(x1 - x0)
Δg = (g1 - g0)/(x1 - x0)
```

$\Delta b = (b_1 - b_0)/(x_1 - x_0)$
$y = y_0,\ r = r_0,\ g = g_0,\ b = b_0$
for $x = x_0$ to x_1 **do**
  draw(x, round(y), r, g, b)
  $y = y + \Delta y$
  $r = r + \Delta r$
  $g = g + \Delta g$
  $b = b + \Delta b$

62

对中点（隐式）算法也可做类似处理，但如果想保留纯整数运算则很难做到。其中的浮点除法非常耗时，上面的4个除法可以用一个除法和4个乘法代替。

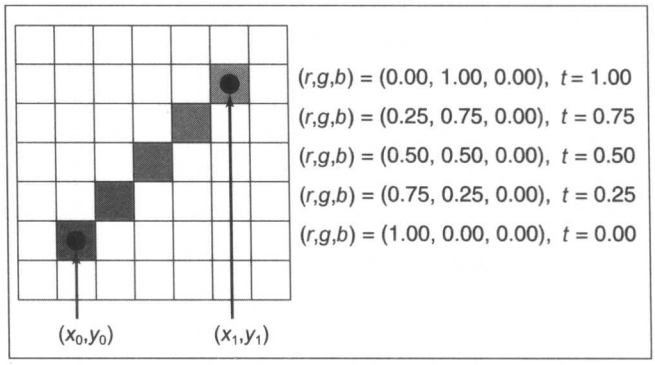


图3-9 一条从红变到绿的彩色直线。中点像素的颜色是半红半绿，即“深黄色”（另见彩图II）

3.6 三角形光栅化

在屏幕坐标系中，我们经常想绘制一个过二维点$\boldsymbol{p}_0=(x_0, y_0)$、$\boldsymbol{p}_1=(x_1, y_1)$、$\boldsymbol{p}_2=(x_2, y_2)$的二维三角形。这与直线绘制问题类似，但是它也有自己的特别之处。已经证明端点采用整数坐标没有什么益处，因此允许(x_i, y_i)具有浮点值。与直线绘制一样，希望能根据顶点数值插入颜色或其他性质。如果存在重心坐标系（见2.11节），就可以直接计算插入值。例如，假设顶点的颜色为$\boldsymbol{c}_0$、$\boldsymbol{c}_1$、$\boldsymbol{c}_2$，在重心坐标为(α, β, γ)的三角形内部，某点的颜色值为

$$\boldsymbol{c} = \alpha\boldsymbol{c}_0 + \beta\boldsymbol{c}_1 + \gamma\boldsymbol{c}_2$$

这种颜色插值方法在图形学中称为Gouraud插值法，来源于提出该算法的作者（Gouraud，1971）的名字。

光栅化三角形的另一个特别之处是，我们通常对具有公共顶点和公共边的那些三角形进行光栅化。这就意味着要光栅化的是相邻三角形，因此没有空隙。可以使用中点算法画出每个三角形的轮廓，然后填入内部像素，从而实现三角形光栅化。因此，相邻三角形在邻边处都绘制了相同的像素。如果两相邻三角形的颜色不同，则图像将由两个三角形被绘制的顺序来确定。为了避免顺序问题并且消除空隙，最常用的三角形光栅化方法是利用下列约定：当且仅当一个像素的中心位于三角形内部时，才绘制该像素。也就是说，像素中心的重心坐标都在区间(0,1)内。于是又产生了这样的问题：如果像素的中心正好位于三角形的边上，这时怎么办？在后面将讨论到，有好几种方法来处理这种情况。关键结论是：当我们根据顶点插入颜色时，利用重心坐标可以确定是否该绘制一个像素，以及该像素应是什么颜色。于是问题变为如何能快速找到像素中心的重心坐标（Pineda，1988）。蛮力光栅化算法如下：

```
for all x do
  for all y do
63  compute (α, β, γ) for (x, y)
    if (α ∈ [0,1]) and β ∈ [0,1]) and γ ∈ [0,1]) then
      c = αc0 + βc1 + γc2
      drawpixel (x, y) with color c
```

算法的其余部分把外部循环限制到更小的候选像素集，使重心计算快速有效。

寻找三顶点的包围矩形，只对该矩形内的候选像素执行循环，这样就能提高算法的效率。利用式（2-32）来计算重心坐标。于是算法变为：

```
x_min = floor (x_i)
x_max = ceiling (x_i)
y_min = floor (y_i)
y_max = ceiling (y_i)
for y = y_min to y_max do
  for x = x_min to x_max do
    α=f12(x,y)/f12(x0, y0)
    β=f20(x,y)/f20(x1, y1)
    γ=f01(x,y)/f01(x2, y2)
    if (α > 0 and β>0 and γ > 0) then
      c = αc0 + βc1 + γc2
      drawpixel (x, y) with color c
```

其中 f_{ij} 是根据适当顶点，通过式（3-3）确定的直线：

$$f_{01}(x,y)=(y_0-y_1)x+(x_1-x_0)y+x_0y_1-x_1y_0$$
$$f_{12}(x,y)=(y_1-y_2)x+(x_2-x_1)y+x_1y_2-x_2y_1$$
$$f_{20}(x,y)=(y_2-y_0)x+(x_0-x_2)y+x_2y_0-x_0y_2$$

注意我们用判断条件 $\alpha>0$代替了 $\alpha\in(0,1)$。原因是，考虑到 $\alpha+\beta+\gamma=1$，那么如果 α、β、γ 都是正的，它们必然都小于1。同样，我们可以只算出三个重心坐标中的两个，然后利用它们的这种关系计算另一个。和绘制直线的算法类似，这时的算法也可以是增量算法，但不清楚能否降低计算量。每次计算 α、β、γ，也就是计算 $f(x,y)=Ax+By+C$ 的值。在内部循环中，只有 x 变化，每次变
64 化1。要注意 $f(x+1,y)=f(x,y)+A$，这是增量算法的基础。在外部循环中，由计算 $f(x,y)$变成计算 $f(x,y+1)$，因此计算效率类似。由于在循环中 α、β、γ 的变化增量是固定的，颜色 **c** 的变化也一样，因此该算法也可以变为增量算法。例如，像素(x,y)到像素$(x+1,y)$的红色值改变量，可以设为预先计算出的一个常量。三角形颜色插值的实例见图3-10。

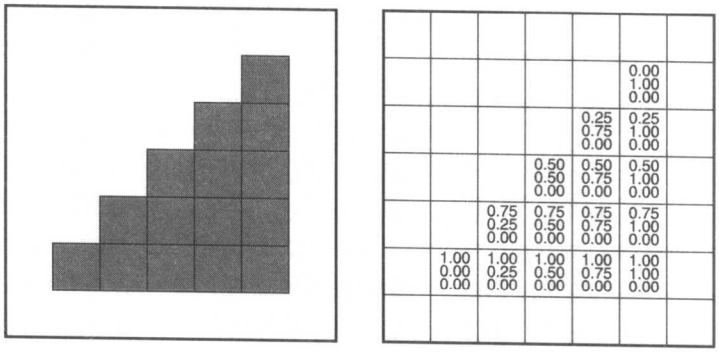


图3-10　用重心插值的彩色三角形。注意在每行、每列及沿着每条边的颜色分量的变化是线性的。实际上，沿着每条线的变化都是线性的，如对角线（另见彩图III）

处理三角形边上的像素

我们还没有讨论如何处理那些中心位于三角形边上的像素。如果一个像素正好落在三角形的边上，那么它同时还位于相邻三角形的边上。没有明确的方式来确定该像素到底属于哪个三角形。最坏的决定就是不绘制该像素，这时两个三角形之间就会出现一个空隙。如果在两个三角形上都绘制该像素，情况相对会好一点，但仍然不是个好办法。如果三角形是透明的，会造成重复绘制。我们确实想把这样的像素点归于其中的一个三角形，并希望这是个简单过程。只要做好选择，无论选定哪个三角形都没有关系。

应该注意到，屏幕外的任何点肯定位于要绘制的公共边的一侧。对两个互不重叠的三角形来说，不在公共边上的顶点位于公共边的相对两侧。这些顶点中正好有一个与屏幕外的点处在同一侧（见图3-11）。这个常识是做判断的基础。判断p与q是否同号，可以通过判断$pq>0$是否成立来实现，这种方法在多数情况下都是行之有效的方法。

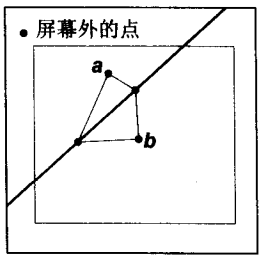


图3-11 屏幕外的点位于三角形边的某一侧。非公共顶点***a***和***b***中有一个正好与该屏幕外的点位于同一侧

注意这种判断方法不是完美的，因为通过三角形边的直线也许会正好通过屏幕外的那一点， 65
但至少我们缩小了考虑范围。选用屏幕外的哪一点是任意的，较好的选择是$(x,y)=(-1,-1)$。需要对那些正好位于边上的点进行核查，但对于完全在内部或者完全在外部的一般情况不应核查。所以有：

x_{min} = floor (x_i)
x_{max} = ceiling (x_i)
y_{min} = floor (y_i)
y_{max} = ceiling (y_i)
$f_\alpha = f_{12}(x_0, y_0)$
$f_\beta = f_{20}(x_1, y_1)$
$f_\gamma = f_{01}(x_2, y_2)$
for $y = y_{min}$ to y_{max} **do**
  for $x = x_{min}$ to x_{max} **do**
    $\alpha = f_{12}(x,y)/f_\alpha$
    $\beta = f_{20}(x,y)/f_\beta$
    $\gamma = f_{01}(x,y)/f_\gamma$
    if ($\alpha \geq 0$ and $\beta \geq 0$ and $\gamma \geq 0$) **then**
      if ($\alpha>0$ or $f_\alpha f_{12}(-1,-1)>0$) and ($\beta>0$ or $f_\beta f_{20}(-1,-1)>0$)
      and ($\gamma>0$ or $f_\gamma f_{01}(-1,-1)>0$) **then**
        $\mathbf{c} = \alpha\mathbf{c}_0 + \beta\mathbf{c}_1 + \gamma\mathbf{c}_2$
        drawpixel (x,y) with color **c**

当对两个三角形使用同样的直线方程时，我们会希望上面的代码能够消除空隙和重复绘制。实际上，在调用三角形绘制函数时，只有当两个公共顶点的顺序是一样时，直线方程才是一样的。否则直线方程的符号就不同。这个问题取决于编译器是否改变操作的顺序。因此，如果想使算法稳

健，应明白编译器以及算术单元的细节。上面伪码的前4行需要认真考虑，用以处理三角形的边正好过像素中心的情况。

除了能把代码改为增量形式外，还存在一些提前退出的可能。例如，如果α是负的，那么就不必计算β或者γ了。显然这样做能够提高运算速度，对程序进行改进总是值得提倡的。额外的分支对流水操作和并行处理不利，而且会降低代码的运行速度。所以，对于关键部分的代码，应该尝试各种优化方法。

以上代码的另一个细节是，对于退化三角形，即$f_\gamma = 0$情况，上面的代码可能会出现除数为
66 0的现象。为了解决这个问题，应采用浮点误差条件，或者增加其他判断方式。

3.7　简单反走样技术

前面讲过的直线绘制和三角形绘制算法，普遍存在锯齿现象。如果允许像素值变低或变高一些（Crow，1978），则可以减轻这种视觉上的不足。例如，对于白色背景上的一个黑色三角形，如果一个像素的中心几乎位于该三角形的内部，则可以将该像素设为黑白之间的颜色。图3-12中上面的那条直线就是按这种方法绘出的。在实际应用中，这种模糊化处理可以改进视觉效果，尤其是在动画制作中。

构造“无锯齿”图像的最直接方法就是采用*盒式滤波器*，将像素设置为盒式区域的平均颜色值。这就意味着，要把所有可绘制的实体都看作已经明确定义的区域。例如，图3-13中的直线段就是一个矩形。第4章中讨论更复杂的模糊化技术，这些技术可以改善外观质量。像中点算法产生的锯齿状直线这样的现象，是*走样*（aliasing）造成的结果，该术语来自信号处理。因此，通过认真选择像素值来避免锯齿现象的技术，称为*反走样*（antialiasing）技术。盒式滤波器可以满足多数情况的需要，适用于对视觉质量要求不是特别高的场合。

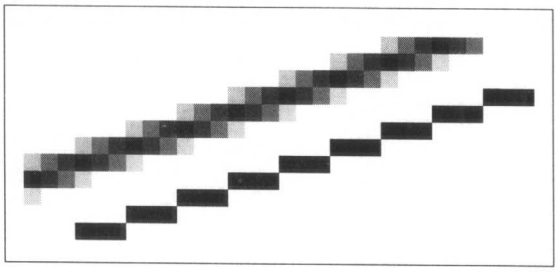


图3-12　近距离观察经反走样处理过的和锯齿状直线，可以观察到每个像素

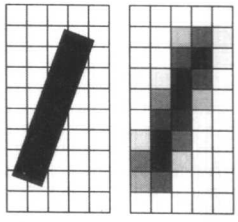


图3-13　把直线段看作矩形，经盒式滤波得到无锯齿直线

实现盒式滤波器反走样的最容易办法是，先建立高分辨率图像，然后进行下采样。例如，我们要处理一幅256×256的图像，其中的直线宽度为1.2个像素。可以在1024×1024的屏幕上，对宽为4.8像素的直线段构成的矩形区域进行光栅化处理，然后对像素进行4×4平均，得到256×256的压缩图像。这是对实际图像的近似盒式滤波，如果目标相对于像素间距不是特别小的话，这种
67 方法的效果还是非常好的。

3.8　图像捕捉与存储

几乎所有的图形软件都处理实际图像，这些图像由数码摄像机或者平台扫描仪获取。本节讨论对这些图像的获取、存储及处理。

3.8.1 扫描仪和数码摄像机

扫描仪和数码摄像机使用某些类型的光感芯片来记忆光能。其核心技术是CCD和CMOS阵列。这些阵列可以感测所有波长的光强。要获得彩色图像，一种方法是将光线分成三路，然后经红、绿、蓝三片滤光器进行滤光（需要三个硅片），经同一硅片的三个滤光器之后就生成了红、绿、蓝三色光。另一种方法是，在硅片的各传感器上分别覆盖不同的彩色滤光层。

现在的多数数码摄像机，都是单光感CCD与Bayer马赛克（Bayer，1976）彩色滤光器一起使用（见图3-14）。这种模式将传感器的一半空间用作绿色通道，而红色和蓝色通道各占四分之一。绿色通道的传感器布局与芯片点阵成45°角。对于自然景物，这种模式的使用效果很好。但是对于一些人造景观会产生颜色混乱现象。采用比单通道线性插值更复杂的Bayer马赛克技术，将捕捉的图像变成一幅RGB彩色图像，不同的摄像机制造商都有各自不同的一套算法。

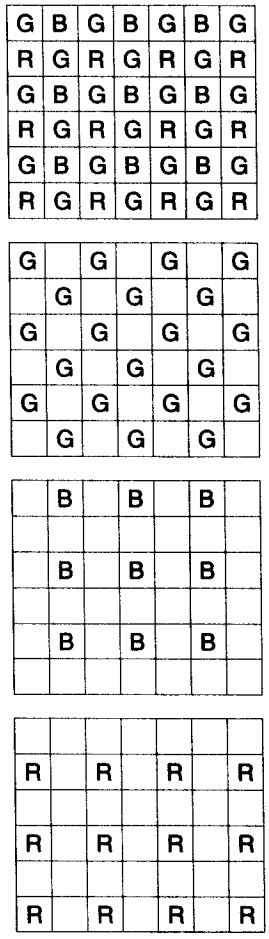


图3-14 最上面的图是一般数码摄像机的RGB传感器马赛克布局图。下面的三幅图显示与上面相同模式的单色布局

3.8.2 图像存储

在多数RGB图像格式中，红、绿、蓝通道各占8位二进制数，因此存储一幅百万像素的原始图像大约需要3MB。为减少所需的存储空间，多数图像格式都进行某种形式的压缩。这种压缩分为无损压缩和有损压缩。在无损压缩中信息不会丢失；在有损压缩中会丢失部分信息，而且是不可恢复的丢失。常见的图像存储格式包括：

- **gif**：有损压缩格式，只能表示出256种可能的颜色。图像质量取决于这256种颜色的选择。这种格式适用于自然图像。
- **jpeg**：有损压缩格式，以人类视觉系统的阈值为基础。适用于自然图像。
- **tiff**：无损压缩格式，每个像素通常占24位的常用压缩图像格式，虽然存在很多其他的可选格式。
- **ppm**：无损压缩格式，每个像素通常占24位的常用未压缩图像格式，虽然存在很多其他的可选格式。
- **png**：无损格式的集合，具有很好的开放源码管理工具。

由于压缩和变形，可能还会涉及为图像写输入/输出程序。对非商业应用，如果没有读/写库可用的话最好只使用原始的ppm格式。

常见问题

- **人们为什么不把显示器做成线性的，从而避免伽马校正？**

理想情况下，显示器的256级亮度看起来应该是均匀分布的，与能量的线性分布对应。人类的光感知系统本身是非线性的，通过1.5~3之间的伽马校正（大小取决于观察条件），在主观上就会感到亮度分布近似是均匀的。从这个角度来看，伽马代表一种视觉特征。要不然，制造商就会把显示器设计成线性的了。

- **如何将非三角形的多边形光栅化？**

可以直接采用逐行扫描，或者把多边形分割成三角形，然后再按照三角形的光栅化方法来处理。

- **反走样技术总是有利的吗？**

不是。有的图像如果不进行反走样处理，看起来更清晰。很多程序使用锯齿状的“屏幕字体”，因为这样更容易阅读。

习题

69

1. 推导直线中点绘制算法的增量形式，端点的颜色值$0 < m \leqslant 1$。
2. 修改三角形的绘制算法，使得对于三角形某边上的点，正好只绘制一个像素。其中三角形的这条边通过点$(x,y)=(-1,-1)$。
3. 获取一幅自然图像（最好是扫描图像，而不是经Bayer马赛克处理过的数字图像），并根据交错布局的红/绿/蓝数据合成一幅灰度图像，通过这种方法来模拟经Bayer马赛克处理的图像。本题模拟数码摄像机的原始图像。然后从该原始图像建立RGB图像，并将它与原来的自然图像做比较。

70

第4章

信号处理

Stephen R. Marschner

在图形学中，我们经常遇到连续变量函数：图像就是我们见过的第一个实例。随着对图形学研究的深入，我们会遇到更多类似的情况。但是连续函数不能直接用于数字计算机，必须设法对它们进行数字化处理。处理连续函数的最有效方法之一，是利用函数的采样值，将函数在多个不同点处的值存起来，需要时就重构其他函数值。

我们已经熟悉了用二维像素栅格表示一幅图像的思想，也就是说我们已经见识了采样表示法。下面考虑用数码摄像机获取图像的情况，通过镜头得到的实际场景图像是关于图像平面上的位置的连续函数，摄像机再将该函数转化成离散的二维栅格。如用数学方法表示，就是摄像机把一个 $\mathbb{R}^2 \to C$（C是颜色的集合）类型的函数转化成颜色样本的二维阵列，也就是 $\mathbb{Z}^2 \to C$类型的函数。

另一种样本表示的实例是二维数字化的书写板，如平板计算机或者PDA的屏幕。在这些实例中，初始函数是触笔的运动，它是时变的二维位置，或者说是 $\mathbb{R} \to \mathbb{R}^2$ 类型的函数。数字转换器随时测量触笔在很多点的位置，得出二维坐标的序列，也就是 $\mathbb{Z} \to \mathbb{R}^2$ 类型的函数。对于附着在行动者身上的特殊标志，运动捕捉系统起着与前面数字转换器类似的作用，它随时得到标志的三维位置（$\mathbb{R} \to \mathbb{R}^3$），并将其转化为一系列的瞬时位置值（$\mathbb{Z} \to \mathbb{R}^3$）。

维数更高的实例是医疗CT扫描仪，用来非接触地检查人体内部，将密度作为体内位置的函数进行测量。扫描仪的输出是关于密度值的三维栅格，它把人体密度（$\mathbb{R}^3 \to \mathbb{R}$）转换为实数三维阵列（$\mathbb{Z}^3 \to \mathbb{R}$）。

这些实例似乎难以理解，但实际上都基于相同的数学原理。在所有的实例中，都是在一维或者高维栅格点对函数进行采样，并且都需要能根据样本阵列重构原来的连续函数。

就二维图像的实例来说，像素可能已经足够多了，自从摄像机把图像离散化后我们就再也不考虑连续函数了。但是，如果想在屏幕上把图像变大或者变小该怎么办，尤其是当缩放非整数倍数时？实践证明用最简单算法的效果非常差，出现了明显的附加噪声，称为走样。理解这种现象出现的原因并加以避免，需要用到关于采样理论的数学知识。结果得到的算法相当简单，但其后面隐藏的推理过程，以及如何正确运用这些算法，都不是件容易的事情。

就图形学来说，在计算机中表示连续函数的方法不是唯一的。采样与重构的方法也不是唯一的。从数字音频到计算物理学都用到了采样表示，图形学只是相关算法及数学知识的应用领域之一，但决不是第一个。20世纪20年代，通信领域已经用到了采样与重构技术。到20世纪40年代，关于采样与重构理论的论述就和现在所用的形式一样了（Shannon & Weaver，1964）。

本章第一节，用具体的一维数字音频实例简单介绍采样与重构技术。接着通过一维和二维实例介绍采样与重构的基本数学知识和算法。最后，详细介绍频域知识，从而能够更深入研究这些算法的原理。

4.1　数字音频：一维采样

虽然采样表示在电信领域已经使用了很多年，但是，随着在过去十年里音频数字记录得到越
72 来越多的应用，于是在1982年出现了压缩唱片，这才是采样理论第一次在消费市场得到应用。

在音频记录中，麦克风把以压力波形式在空气中存在的声音转化成时变的电压信号，该电压信号成为麦克风附近气压变化的度量值。要设法把这种电信号存储起来，以便稍后能通过扬声器播放出来。扬声器与电压信号同步使膜片振动，就把电压信号转变成了压力波。

记录音频信号的数字方法是采样（见图4-1）：模数转换器（ADC）每秒钟测量电压数千次，产生整数流。这些整数可以很容易地存储在媒体中，比如录音棚里电脑的磁盘；或者传送到另外的位置，如可移动音频播放器的内存里。在重放录音时，按照适当的速度读出数据，然后送入数模转换器（DAC）。DAC根据所得数值的大小产生一个电压。从实际应用的角度来看，如果我们得到了足够多的样本，能够完全表示电压的变化，那么经DAC得到的电信号与输入信号将是相同的。

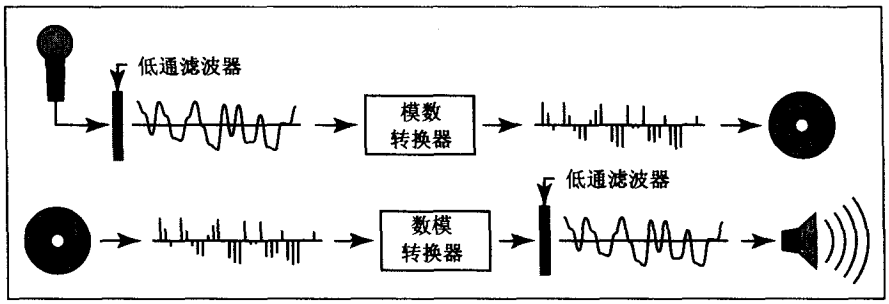


图4-1　数字音频的采样与重构

已经证明，要想使重构出的音频具有较高的质量，每秒的采样次数取决于所记录声音的音调高低。假设某个采样速度能很好再现弦低音或者击鼓的声音，但如将该采样速度用于恢复短笛或者铙钹声音，则会产生稀奇古怪的效果。如果采用更高的采样速度，就能很好地再现短笛或者铙钹的声音效果。为了避免这种欠采样干扰（undersampling artifact），数字音频记录仪将输入进行滤波后再送给ADC，以去掉其中可能产生问题的高频信号。

另一种问题出现在输出端。DAC产生一个电压，每当新的样本到来时该电压就会随之改变，但是它在下一个样本到来之前一直是恒定的，因此产生阶梯状的波形。这些阶梯状波形就像是噪声，附加了一个高频、依赖信号的嗡嗡声。为了消除这种重构干扰（reconstruction artifact），数
73 字音频播放器将对DAC的输出进行滤波，从而对波形进行平滑。

采样干扰与走样

数字音频记录链可作为图形学采样与重构过程的具体模型。在图形学的图像或者其他采样信号中，也会出现同样的欠采样干扰和重构干扰现象，解决办法也一样：采样前滤波，重构过程中再次滤波。

出现这种干扰的具体实例见图4-2，主要是由于采样频率太低造成的。例如对简单的正弦波进行采样，使用两种不同的采样频率：上面图中每个周期采样10.8次，下面图中每个周期采样1.2次。显而易见，高频率采样得到的样本集能较好地代表原始信号；而低频率采样的样本集，其表示结果很难与更低频的正弦波区别，实际上这样的样本集合，似乎更适合代表图中虚线表示的低频正弦曲线。

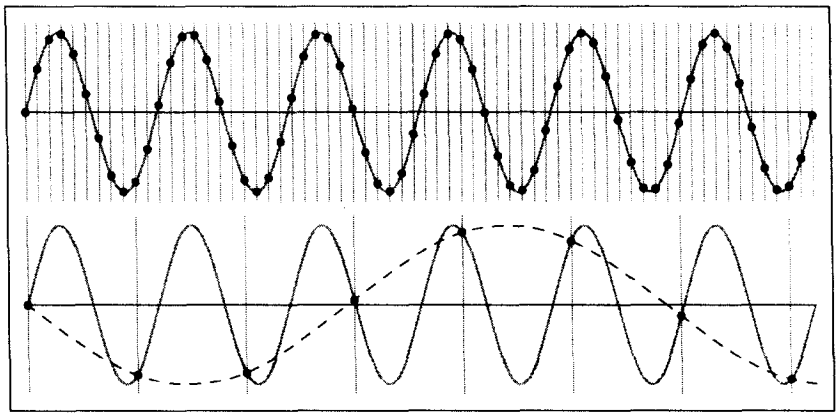


图4-2 用两种不同的采样频率对同一正弦波（灰色曲线）进行采样。上图：采用高频采样率，得到的样本（黑点）能够很好地表示原始信号。下图：采用低频采样，产生的结果具有歧义性：得到的样本集与图中低频正弦波（虚线）的样本集相同

采样之后我们得到了样本集，但不知道结果与哪个原始信号对应，是高频信号还是低频信号？这时，找不出能够适当重构原始信号的唯一方法。这种高频信号“假扮”低频信号的现象，称为走样。

当采样与重构中因频率问题带来人为干扰时，就产生了走样现象。对于声音情况，走样现象的结果是产生听起来很奇怪的音调，例如10 kHz的门铃声音，以8 kHz采样后，就变成了6 kHz的音调。对于图像情况，走样经常表现为莫尔图案（moire pattern），它是由样本栅格与图像规则特征相互作用形成的结果，例如图4-34中的百叶窗。 74

在合成图像中，另一个走样实例是我们常见的呈阶梯状的直线，其中直线只有黑、白两色像素组成（见图4-34）。这是小尺度特征（直线的尖锐边缘）产生不同尺度（浅斜线的阶梯较长）干扰现象的实例。

采样与重构的基本问题，可以简单地借助特征太小或者太大来理解，但是更多定量问题难以回答：

- 采用多高的采样频率能保证得到好的结果？
- 什么样的滤波器适合于采样与重构？
- 为了避免走样，需要平滑到什么程度？

对这些问题的回答要等到4.5节讲完采样定理之后。

4.2 卷积

在讨论采样与重构算法之前，首先研究其依赖的数学概念，即卷积。卷积是一个简单的数学概念，它是采样、滤波、重构等算法的基础。同样它还是本章后面分析这些算法的基础。

卷积是对函数的运算，由两个函数产生一个新的函数。在本书中，卷积运算符用星号“$\star$”表示。函数f与g的卷积表示为$f \star g$，读作f与g进行卷积，或者$f \star g$是f与g的卷积。

卷积可用于连续函数（由任意实数自变量x的函数$f(x)$），也可用于离散序列（整数自变量i的函数$a[i]$）。卷积可用于一维、二维或高维定义域的函数（也就是有一个、两个或者更多个自变量的函数）。我们先从离散一维情况开始，然后过渡到连续函数情况以及二维和三维函数情况。

为了方便，在定义中一般假设函数的定义域是无限的。实际定义域是有限的，并且要对端点进行特殊的处理。 75

4.2.1　滑动平均

为了大致了解卷积，考虑采用滑动平均方法对一维函数进行平滑处理的实例（见图4-3）。为了获得任意点的平滑值，计算该点r邻域的函数平均值。距离r称为平滑运算的半径，这是一个控制平滑程度的参数。

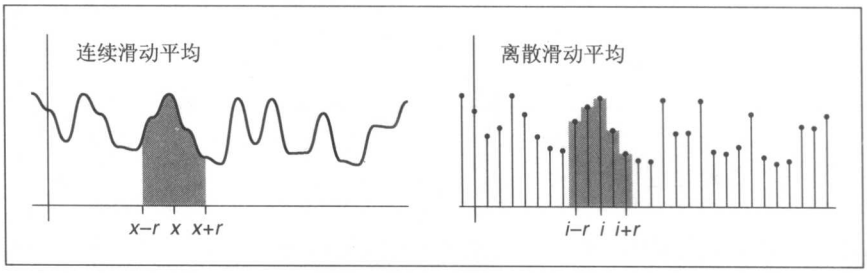


图4-3　用滑动平均方法进行平滑

可以在数学上对离散或连续函数的滑动平均思想进行描述。假设我们对连续函数$g(x)$进行平滑处理。平均就是先在某个区间上对g积分，然后除以区间的长度：

$$h(x)=\frac{1}{2r}\int_{x-r}^{x+r} g(t)\mathrm{d}t$$

另一方面，如果平滑对象是离散函数$b[i]$，平均就是将某个索引范围内的b加起来，然后除以值的个数：

$$c[i]=\frac{1}{2r+1}\sum_{j=i-r}^{i+r} b[j] \tag{4-1}$$

以上两种情况下，都要选择一个规范化常量，使得对常量函数平滑时能得到相同的函数结果。

卷积的本质是滑动平均思想，唯一的不同是卷积中的滑动平均是加权平均。

4.2.2　离散卷积

首先介绍离散卷积计算，即将离散序列$a[i]$与离散序列$b[i]$进行卷积。结果为离散序列$(a\star b)[i]$。
76 该过程就像是对b进行滑动平均，但不是在r邻域内对所有的样本取一样的权重，而是用第二个序列a给每个样本施加权值（见图4-4）。计算第i个卷积时，索引为i–j的样本$b[i-j]$的权重为$a[j]$。这时关于（$a\star b$）的计算公式表示为

$$(a\star b)[i]=\sum_j a[j]\,b[i-j] \tag{4-2}$$

忽略j的边界条件，上式是对所有整数的求和运算（也就是从－∞到＋∞）。图4-4显示输出样本是如何计算出来的。举例来说，$a=\frac{1}{16}[\cdots, 0, 1, 4, 6, 4, 1, 0, \cdots]$，也就是a[0]=6/16，a[±1]=4/16，等等。

在图形学中，一般两个函数中有一个具有有限支撑集（如图4-4中的实例），这意味着只在自
77 变量的有限区域内取非零值。如果假设a的支撑集有限，那么就存在某个半径r，使得当$|j|>r$时，都有$a[j]=0$。这时，上面的求和公式可写为

$$(a\star b)[i]=\sum_{j=-r}^{r} a[j]b[i-j]$$

将上面的定义用代码实现为

```
function convolve(sequence a,sequence b, int r, int i)
  s=0
  for j=-r to r
    s=s+a[j]b[i-j]
return s
```

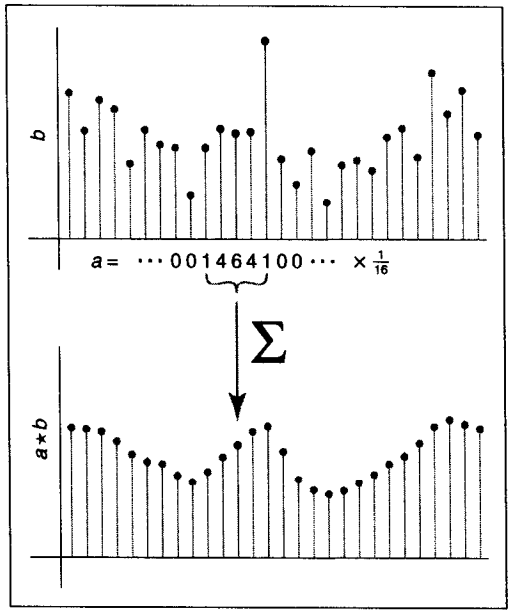


图4-4 计算序列b与滤波器a的离散卷积，支撑集为五个样本宽度。$a\star b$中的每个样本都是b的邻域样本的加权平均，其中权重就是a的值

1. 卷积滤波器

卷积很重要，因为我们可以用它来进行滤波。回到关于滤波的第一个实例，即滑动平均滤波。现在可以将滑动平均操作解释为是与一个特殊序列的卷积。当计算有限索引范围内的平均值时，就相当于对该范围内的点取相同的权值，而对此范围之外的点权值取0。这种滤波器，在非零区间上存在一个常数，称这种滤波器为盒式滤波器（因为画出来看着像矩形，见图4-5）。一个半径为r的盒式滤波器，其权值为$1/(2r+1)$：

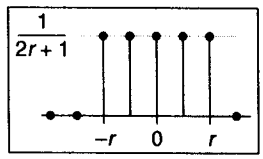


图4-5 离散盒式滤波器

$$a[j]=\begin{cases}\dfrac{1}{2r+1} & -r\leqslant j\leqslant r\\ 0 & \text{其他}\end{cases}$$

如果将该滤波器代入式（4-2）中，就会发现变成了滑动平均公式，即式（4-1）。

如下面的例子所示，设计卷积滤波器时，一般保证它们的和为1。这样就不会影响信号的总体水平。

例题：求盒式滤波器与阶跃函数的卷积。设阶跃函数信号为

$$b[i]=\begin{cases}1 & i\geqslant 0\\ 0 & i<0\end{cases}$$

滤波器是五点的盒式滤波器，以0为中心：

78

$$a[j]=\frac{1}{5}\begin{cases}1 & -2\leqslant j\leqslant 2\\ 0 & \text{其他}\end{cases}$$

a和b的卷积结果是什么？如图4-6所示，在某个索引i上，结果是阶跃函数在i-2到i+2范围内的平均值。如果i<-2，就是对所有的0求平均，结果为0。如果$i\geqslant 2$，就是对所有的1求平均，结果为1。在这之间有i+3个1，结果为$\frac{i+3}{5}$。输出是0到1的线性斜坡函数，斜坡上有五个样本点：$\frac{1}{5}[\cdots, 0, 0, 1, 2, 3, 4, 5, 5, \cdots]$。

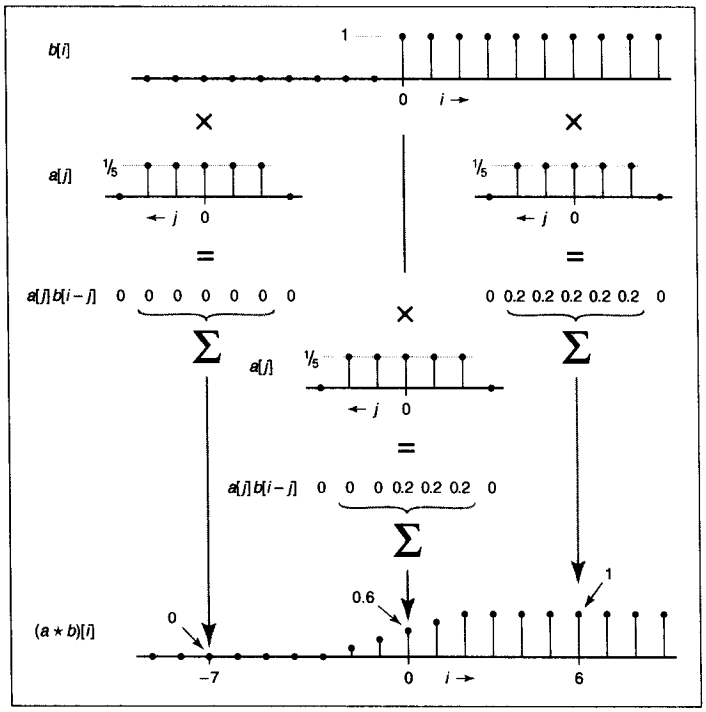


图4-6　盒式滤波器和阶跃函数的离散卷积

2. 卷积性质

到目前为止，卷积似乎是一种不对称的运算：b是我们要平滑的序列，而a提供权值。卷积的一大优点是：滤波器和信号可以互换，二者实际上没有什么不同。为了弄明白这个问题，回想式

79

（4-2）的求和公式，其中的索引来自序列b，而不是a。下面我们根据序列a来得到索引值。在式（4-2）中，用i–k代替j，结果得到

$$\begin{aligned}(a\star b)[i]&=\sum_k a[i-k]b[i-(i-k)]\\&=\sum_k b[k]a[i-k]\end{aligned}$$

上式与式（4-2）一样，只是用b作为滤波器而用a作为信号。因此，对于任意序列a和b，$(a\star b)=(b\star a)$，卷积运行满足交换律。[1]

1. 我们可能会注意到，卷积求和中的一个函数似乎存在跳变，也就是说，$a[j]$在序列中赋给样本的权值提前了j个单位，而$a[-j]$赋给样本的权值延迟了j个单位。这样做是为了保证结合律，见习题4。由于多数滤波器是对称的，因此不需要担心这个问题。

更一般地说，卷积运算类似乘法运算。和数值或函数的加法及乘法运算一样，自变量的顺序和括号的位置都不会影响计算结果。同样，卷积与加法的关系和乘法与加法的关系一样。更准确地说，卷积运算满足交换律、结合律，对加法满足分配律：

$$\text{交换律：} \quad (a \star b)[i] = (b \star a)[i]$$
$$\text{结合律：} \quad (a \star (b \star c)[i] = ((a \star b) \star c)[i]$$
$$\text{分配律：} \quad (a \star (b+c)[i] = (a \star b + a \star c)[i]$$

如果把卷积看成与乘法一样，上面的性质就容易理解了。在计算卷积之前，利用上述性质对卷积进行化简，就可以节省工作量。例如，求序列b与三个滤波器a_1、a_2、a_3的卷积，也就是求$a_3 \star (a_2 \star (a_1 \star b))$。如果序列很长而滤波器很短（也就是它们的半径很短），根据交换律和结合律，先计算三个滤波器的卷积（$a_1 \star a_2 \star a_3$），最后计算与信号的卷积$(a_1 \star a_2 \star a_3) \star b$，这样计算速度更快，而结果不变。

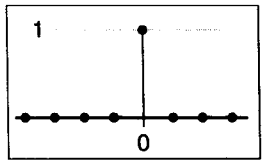


图4-7 单位离散滤波器

下面介绍一种简单的滤波器，可作为离散卷积的单位函数：它是半径为零的离散滤波器，也就是序列$d[i]=\cdots, 0, 0, 1, 0, 0, \cdots$（见图4-7）。如果将$d$与一个信号$b$进行卷积，那么在求和式中只有一项的值不为零：

$$(d \star b)[i] = \sum_{j=0}^{j=0} d[j]b[i-j]$$
$$= b[i]$$

80

很显然，b与d求卷积结果还是b。序列d称为**离散脉冲**。在表示滤波器时有时要用到离散脉冲。例如，用滤波器a平滑信号b，然后将结果从初始信号b中减去的过程，可以表示为信号单独与滤波器$d-a$求卷积：

$$c = b - a \star b = d \star b - a \star b = (d-a) \star b$$

4.2.3 把卷积看作移位滤波器之和

对式（4-2）存在另一种完全等价的解释方法。我们已经知道，每次考虑$a \star b$的一个样本，产生了加权平均的解释。如果省去$[i]$，可以认为是对整个序列求和。需要先定义一种表示符号。如果b是一个序列，那么将整个序列向右移动j个位置，得到的新序列称为$b_{\rightarrow j}$（见图4-8）：

$$b_{\rightarrow j}[i] = b[i-j]$$

那么，就可以把式（4-2）表示为关于整个$(a \star b)$的形式，而不是关于单个元素的形式：

$$(a \star b) = \sum_j a[j]b_{\rightarrow j}$$

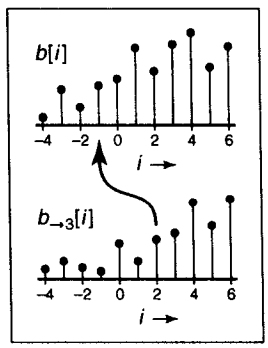


图4-8 通过移动序列b，得到$b_{\rightarrow j}$

用这种方式来考虑，卷积就是b的移位序列之和，权值由a的项来确定（见图4-9）。由于服从交换律，因此a或b都可以作为滤波器。如果选择b作为滤波器，那么就是针对输入的每个样本将移位滤波器相加。

81

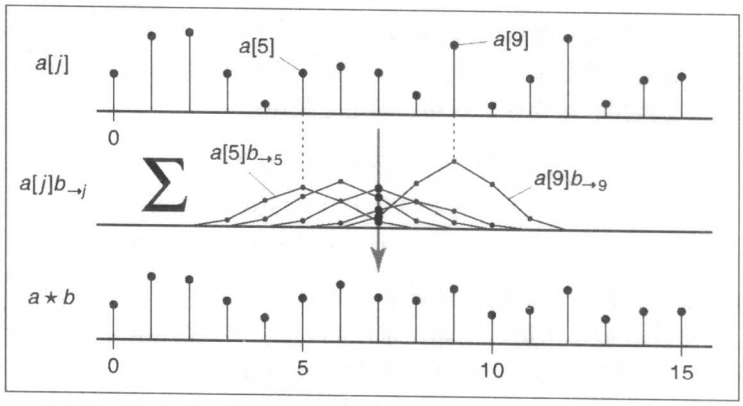


图4-9　把离散卷积看作移位序列之和

4.2.4　与连续函数的卷积

在计算机编程中虽然我们实际处理的是离散序列，但是我们认为这些采样序列代表的是连续函数，并且经常需要对连续函数进行数学推导，以决定下一步应该做什么。因此，对连续函数之间以及连续函数和离散函数之间的卷积进行定义是非常有用的。

两连续函数之间的卷积，显然是式（4-2）的推广，即用积分代替求和运算：

$$(f\star \mathrm{g})(x)=\int_{-\infty}^{+\infty} f(t)g(x-t)\mathrm{d}t \tag{4-3}$$

82 对该定义的一种解释是，x处 $f\star$ g的值，就是移动f使得$f(0)$与$g(x)$对应之后，两函数之积所形成的曲线下面的面积。和离散情况一样，求卷积就是进行滑动平均，由滤波器来提供权值（见图4-10）。

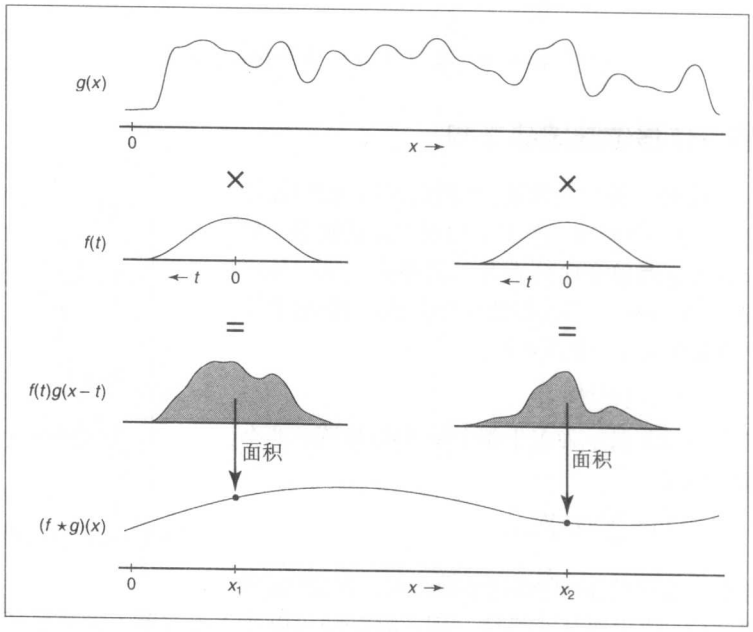


图4-10　连续卷积

和离散卷积一样，连续函数的卷积也满足交换律、结合律以及对加法的分配律。同样，连续卷积也可以看作是对移位滤波器求和，而不是计算加权平均。不同之处在于，在连续情况下，有

无限多的移位滤波器拷贝：

$$(f \star g) = \int_{-\infty}^{+\infty} f(t) g_{\to t} \mathrm{d}t$$

例题：求两盒式函数的卷积。设f是一个盒式函数：

$$f(x) = \begin{cases} 1 & -\frac{1}{2} \leqslant x \leqslant \frac{1}{2} \\ 0 & \text{其他} \end{cases}$$

那么，$f \star f$是什么？根据式（4-3）的定义：

$$(f \star f)(x) = \int_{-\infty}^{\infty} f(t) f(x-t) \, \mathrm{d}t$$

图4-11显示该积分的两种情况。两盒式函数可能有零重叠，即重叠发生在$x \leqslant -1$或者$x \geqslant 1$时。在这种情况下，结果为0。当$-1 < x < 1$时，重叠取决于两盒式函数之间的间隔$|x|$，结果为$1-|x|$。因此有 83

$$(f \star f)(x) = \begin{cases} 1-|x| & -1 < x < 1 \\ 0 & \text{其他} \end{cases}$$

该函数称为帐篷式函数（tent function），是另一种常用的滤波器（见4.3.1节）。

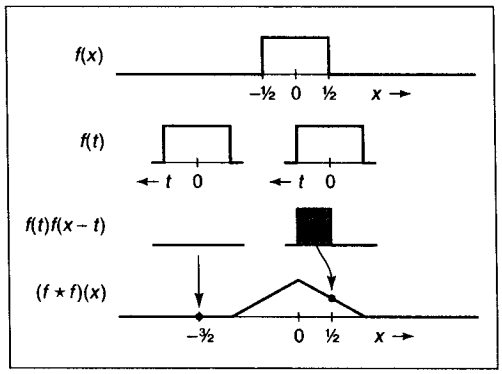


图4-11 两盒式函数的卷积结果是帐篷式函数

狄拉克δ函数

在离散卷积中，我们知道了将离散脉冲d作为单位滤波器，即有$d \star a = a$。在连续情况下，也同样存在一个单位函数，称为狄拉克脉冲函数或者狄拉克δ函数，表示为$\delta(x)$。

直观地说，δ函数是尖而高的脉冲，其宽度无限小但下面的面积仍然等于1（见图4-12）。δ函数存在一个重要的性质，它与某函数的相乘结果等于该函数在0处的值：

$$\int_{-\infty}^{\infty} \delta(x) f(x) \, \mathrm{d}x = f(0)$$

δ函数在0处没有确定值（可以认为是$+\infty$），但是对所有的$x \neq 0$都有$\delta(x) = 0$。

由于δ函数的上述性质，即可以得到函数在某点处的函数值，因此可以将δ函数看成是连续卷积的单位函数（见图4-13）。δ与函数f的卷积为 84

$$(\delta \star f)(x) = \int_{-\infty}^{\infty} \delta(t) f(x-t) \, \mathrm{d}t = f(x)$$

所以有$\delta \star f = f$。

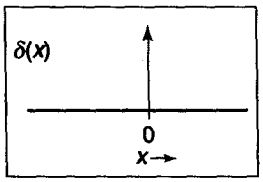


图4-12　狄拉克δ函数

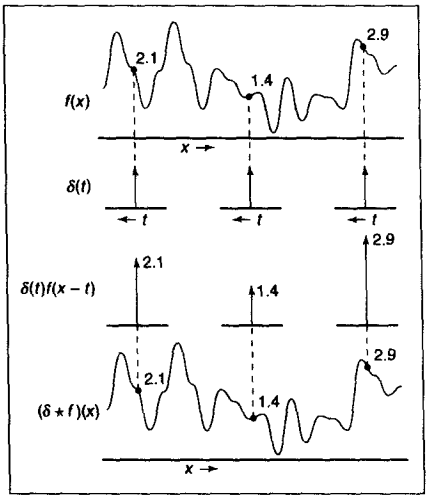


图4-13　函数与δ(x)的卷积仍然是该函数

4.2.5　离散-连续卷积

离散与连续的相互转换存在两种方式。首先是采样，通过记录连续函数在所有的整数自变量处的值，而忽略其他位置的值，这样就将连续函数转化成了离散函数。对于连续函数$f(x)$，通过对它进行采样将其转化为离散序列$a[i]$：

$$a[i]=f(i)$$

其次是从离散函数或者序列到连续函数的转化，称为重构。实现重构要用到另一种卷积形式，即离散-连续卷积形式。在这种情况下，用连续滤波器$f(x)$对离散序列$a[i]$进行滤波：

$$(a\star f)(x)=\sum_i a[i]f(x-i)$$

重构函数$a\star f$在x处的值，就是在x附近关于i值的样本$a[i]$的加权和（见图4-14）。权值来自滤波器f，
85 即间距为一个单位的各点处的函数值。例如，如果x=5.3，并且f的半径为2，那么在点1.3、0.3、-0.7和-1.7处计算 f 的值。注意一般我们把序列写在前面，而把滤波器写在后面，以便对整数求和。

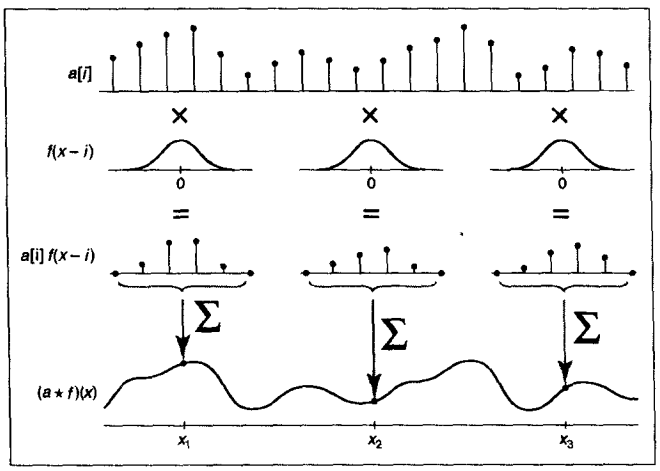


图4-14　离散-连续卷积

和离散卷积一样，如果我们知道滤波器的半径r，就可以对求和进行范围限制，忽略那些x与i至少相差r的所有点：

$$(a \star f)(x) = \sum_{i=\lceil x-r \rceil}^{\lfloor x+r \rfloor} a[i] f(x-i)$$

需要注意，如果某点与x正好相距r（也就是$x-r$的结果是一个整数），求和时它就会被忽略。这与离散情况相反，离散情况下我们包含点$i-r$。

用代码表示，就是

```
function reconstruct(sequence a,filter f, real x)
  s=0
  r=f.radius
  for i=⌈x−r⌉ to ⌊x+r⌋ do
    s = s + a[i] f(x−i)
return s
```

与其他形式的卷积一样，离散-连续卷积看起来是对移位滤波器进行求和（见图4-15）：

$$(a \star f) = \sum_i a[i] f_{\to i}$$

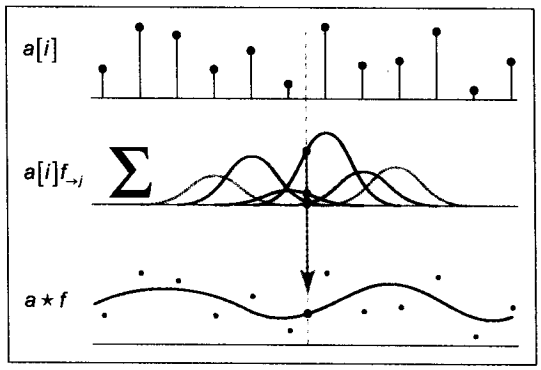


图4-15 重构（离散-连续卷积）也就是对移位滤波器进行求和

离散-连续卷积与样条函数联系密切。对于单值样条（例如均匀B样条），样条的参数化曲线正好是样条的基函数与控制点序列的卷积（见15.6.2节）。 86

4.2.6 多维卷积

到目前为止，我们所讨论的采样与重构问题都是一维的，即只有一个自变量x，或者只有一个索引i。图形学中用到的重要采样与重构，都是针对二维函数，尤其是二维图像。幸运的是，对采样算法及理论从一维到二维、三维或者更高维的推广，在概念上是非常简单的。

我们从离散卷积的定义开始，将其推广到二维情况，将和的形式变成双重求和：

$$(a \star b)[i,j] = \sum_{i'} \sum_{j'} a[i',j'] b[i-i', j-j']$$

如果a是半径为r的有限支撑的滤波器（也就是说，它有$(2r+1)^2$个值），那么我们可以对求和范围进行限制（见图4-16）：

$$(a \star b)[i,j] = \sum_{i'=-r}^{i'=r} \sum_{j'=-r}^{j'=r} a[i',j'] b[i-i', j-j'] \quad 87$$

用代码表示为

```
function convolve2d(filter2d a, filter2d b, int i,int j)
  s=0
  r=a.radius
  for i'=-r to r do
    for j'=-r to r do
      s=s+a[i'][j']b[i-i'][j-j']
return s
```

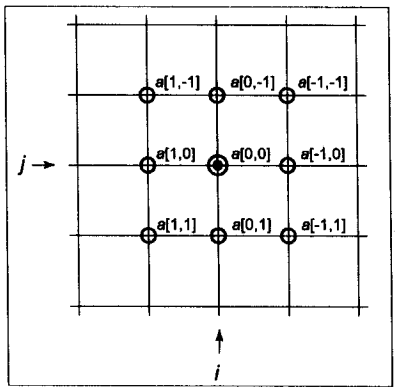


图4-16　9个输入样本的权值，这些样本参与在点（i,j）处的离散卷积运算，采用的是半径为1的滤波器a

类似一维情况，可以对该定义进行解释。输出样本等于输入面积的加权平均值，其中将二维滤波器作为“掩模”以确定每个样本的权值。

继续推广，可以将二维空间中的连续-连续卷积（见图4-17）和离散-连续卷积（见图4-18）表示出来：

$$(f\star g)(x,y)=\int\int f(x',y')g(x-x',y-y')\mathrm{d}x'\mathrm{d}y'$$

$$(a\star f)(x,y)=\sum_i\sum_j a[i,j]f(x-i,y-j)$$

两种情况下，在某点的卷积结果都是在该点附近对输入的加权平均。对于连续-连续情况，是在以该点为中心的区域上进行加权积分；对于离散-连续情况，是对落在该点附近的所有样本的加权平均。

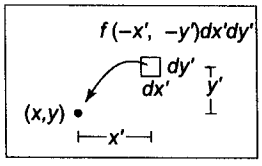


图4-17　点（x,y）处的连续卷积输入信号在无穷小区域的权值

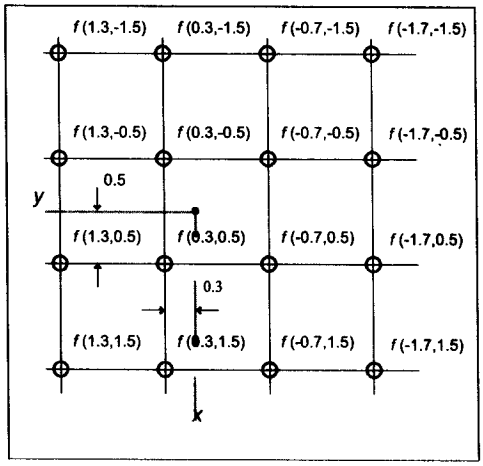


图4-18　16个输入样本的权值，这些样本参与计算在点(x,y)处的离散-连续卷积运算，采用的是半径为2的重构滤波器

卷积运算已经从一维推广到了二维，再推广到三维或者更高维也很容易做到。

4.3 卷积滤波器

既然有了卷积工具，下面来分析图形学中经常用到的特殊滤波器。

4.3.1 各种卷积滤波器

1. 盒式滤波器

盒式滤波器（见图4-19）是一个分段常值函数，它的积分结果为1。作为一种离散滤波器，其数学形式为

$$a_{\text{box},r}[i]=\begin{cases}1/(2r+1) & |i|\leqslant r\\ 0 & \text{其他}\end{cases}$$

注意为了保持对称性，函数定义包含了两个端点。

连续滤波器的数学形式为

$$f_{\text{box},r}(x)=\begin{cases}1/(2r) & -r\leqslant x<r\\ 0 & \text{其他}\end{cases}$$

这时只包含一个端点，使得半径为0.5的盒子能用作重构滤波器。由于盒式滤波器不是连续的，这些边界情况就变得非常重要，而且对于这种特殊的滤波器，我们对边界情况要引起注意。对于常见的半径为r=1/2情况，滤波器函数用f_{box}表示。

2. 帐篷式滤波器

帐篷式滤波器，或称为线性滤波器（见图4-20），是分段连续的线性函数：

$$f_{\text{tent}}(x)=\begin{cases}1-|x| & |x|<1\\ 0 & \text{其他}\end{cases}$$

$$f_{\text{tent},r}(x)=\frac{f_{\text{tent}}(x/r)}{r}$$

对于至少C^0的滤波器（也就是说，其值没有像盒式函数那样的突然跳变），不再需要将离散和连续情况分开。离散滤波器就是连续滤波器在整数处采样后的结果。也要注意，为了简单，我们通过缩放标准大小的帐篷式滤波器f_{tent}来定义$f_{\text{tent},r}$。从现在起，我们认为这种缩放表示是默认的形式。如果定义了滤波器f，那么就可以用f_r表示“滤波器f伸展r倍，或者收缩r倍”。注意f_r与f有相同的积分结果，并且总是保证这个积分结果等于1.0。

89

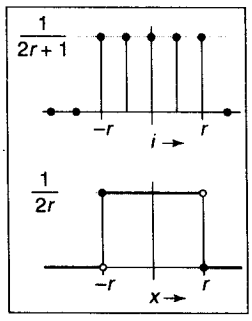


图4-19 离散及连续的盒式滤波器

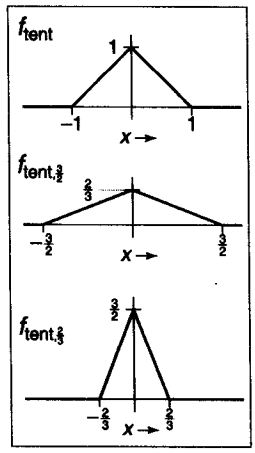


图4-20 帐篷式滤波器及其两种缩放形式

3. 高斯滤波器

高斯滤波器，又称为正态滤波器（见图4-21）。无论在理论上还是在实际中，高斯滤波器都是一类非常重要的滤波器。随着对本章的继续学习，我们将了解其更多的特殊性质：

$$f_g(x)=\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$$

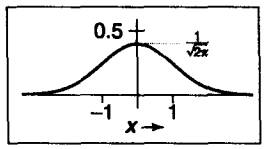


图4-21　高斯滤波器

高斯滤波器没有有限的支撑集，虽然随着指数衰减它的值会迅速地减小到可以忽略不计。必要时，可以对函数进行截尾处理，将某个半径之外的值都设为0。高斯滤波器是一种很好的采样滤波器，因为它非常平滑。在本章后面再进行更深入的讨论。

4. 三次B样条滤波器

很多滤波器都定义为分段多项式，三次滤波器有4段，经常用作重构滤波器。这种滤波器称为B样条滤波器（见图4-22），起源于作为样条曲线的混合函数（见第15章）：

$$f_B(x)=\frac{1}{6}\begin{cases}-3\left(1-|x|\right)^3+3\left(1-|x|\right)^2+3\left(1-|x|\right)+1 & -1\leqslant x\leqslant 1\\ \left(2-|x|\right)^3 & 1\leqslant|x|\leqslant 2\\ 0 & \text{其他}\end{cases}$$

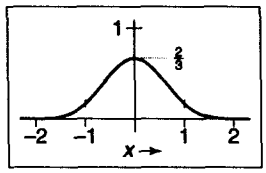


图4-22　B样条滤波器

在分段三次曲线中，B样条是特殊的，因为它有连续的一阶和二阶导数，也就是C^2。定义这种滤波器的更简练的方法是：$F_B=f_{box}\star f_{box}\star f_{box}\star f_{box}$。证明该形式与上面的形式等价，这是一道不错的卷积运算练习题（见习题3）。

5. 三次Catmull-Rom滤波器

90

另一种分段三次滤波器是Catmull-Rom滤波器（见图4-23）。在$x=-2$、-1、1、2处函数值为0，这意味着用作重构滤波器时将插值样本（见4.3.2节）：

$$f_C(x)=\frac{1}{2}\begin{cases}-3\left(1-|x|\right)^3+4\left(1-|x|\right)^2+\left(1-|x|\right) & -1\leqslant x\leqslant 1\\ \left(2-|x|\right)^3-\left(2-|x|\right)^2 & 1\leqslant|x|\leqslant 2\\ 0 & \text{其他}\end{cases}$$

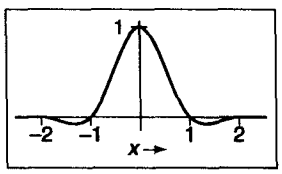


图4-23　Catmull-Rom滤波器

6. 三次Mitchell-Netravali滤波器

对重采样图像的重要应用问题，Mitchell和Netravali（Mitchell & Netravali，1988）做了关于三次滤波器的研究，并在上面两种滤波器基础上推荐折中滤波器，作为各方面最佳的可选滤波器（见图4-24）。实际上，这种滤波器只是前面两种滤波器的加权组合：

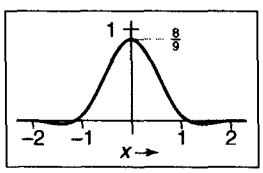


图4-24　Mitchell-Netravali滤波器

$$f_M(x)=\frac{1}{3}f_B(x)+\frac{2}{3}f_C(x)$$

$$=\frac{1}{18}\begin{cases}-21\left(1-|x|\right)^3+27\left(1-|x|\right)^2+9\left(1-|x|\right)+1 & -1\leqslant x\leqslant 1\\ 7\left(2-|x|\right)^3-6\left(2-|x|\right)^2 & 1\leqslant|x|\leqslant 2\\ 0 & \text{其他}\end{cases}$$

4.3.2 滤波器的性质

有一些专业术语伴随着滤波器，我们用这些术语来衡量滤波器的性能，并对滤波器进行相互比较。

滤波器的脉冲响应是滤波函数的另一种称呼，是指滤波器对只包含一个脉冲的信号的响应（回想滤波器与脉冲函数的卷积仍然是该滤波器）。

当用连续滤波器从离散序列重构连续函数时，结果函数只接受采样点处的样本值，也就是说，它“将那些点连起来”，而不是产生经过点附近的函数，这时我们称该连续滤波器是插值的。确切地说，插值滤波器就是那些满足 $f(0)=1$、并且在所有的非零整数i处满足$f(i)=0$的滤波器 f（见图4-25）。 91

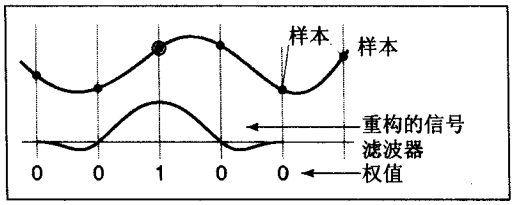


图4-25 插值滤波器准确地重构采样点，因为它在偏离中心的所有非零整数处取值都为0

具有负值的滤波器存在振铃，或者过冲，当被滤波的函数值出现急剧变化时，在这些值附近就会出现额外的振动。

例如，Catmull-Rom滤波器在两边都有负的振铃，当用它对阶跃函数进行滤波时，将使得阶跃处出现过冲，结果出现大于1小于0的函数值（见图4-26）。

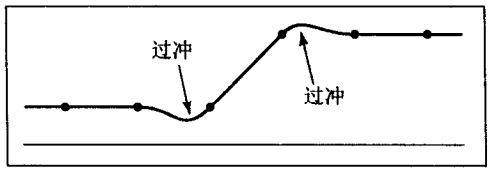


图4-26 有负振铃的滤波器，当对存在尖锐点的不连续函数进行滤波或者重构时，总会产生一些过冲

连续滤波器是无波动的，当用作重构滤波器时，它把常量序列重构为常量函数（见图4-27）。这一点，与整数栅格位置上的滤波函数值之和为1的要求是等价的。即对所有的x，都有

$$\sum_i f(x+i)=1$$

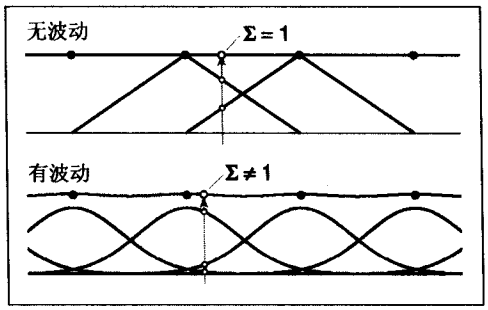


图4-27 半径为1的帐篷式滤波器是无波动的重构滤波器，而标准差为1/2的高斯滤波器则不是

连续滤波器的连续程度，指处处都有定义的最高阶导数。函数值有突变的滤波器，如盒式滤波器，是不连续的滤波器。滤波器连续但是有尖角（即一阶导数不连续），如帐篷式函数，其连续阶数为0，我们称它为C^0。如果滤波器存在连续一阶导数（没有尖角），则称它为C^1，例如前面提到的分段三次滤波器。如果滤波器的二阶导数也是连续的，则称它为C^2，如B样条滤波器。连续阶
92 数这一指标，对重构滤波器来说尤其重要，因为重构的函数继承了滤波器的连续性。

可分滤波器

到现在为止，我们只讨论了一维卷积滤波器，图像及其他多维信号也需要滤波器。一般来说，任何二维函数都可以用作二维滤波器。有时需要直接定义二维滤波器，但在多数情况下，是利用已知的一维滤波器来构建合适的二维滤波器（或者更高维滤波器）。

最有用的构建方式是采用可分滤波器。在某x、y处，可分滤波器的值$f_2(x,y)$等于一维滤波器f_1在x、y处的值之积：

$$f_2(x,y)=f_1(x)f_1(y)$$

类似地，有二维离散滤波器：

$$a_2[i,j]=a_1[i]a_1[j]$$

过f_2的任何横、纵剖面，都具有与f_1类似的形状。f_2的积分等于f_1积分的平方。特别地，如果f_1是正态的，则f_2也是正态的。

例1　可分帐篷式滤波器　假设f_1选用帐篷式函数，则得到分段双线性函数为（见图4-28）：

$$f_{2,\text{tent}}(x,y)=\begin{cases}(1-|x|)(1-|y|) & |x|<1 \text{ and } |y|<1\\ 0 & \text{其他}\end{cases}$$

93 沿坐标轴的剖面是帐篷式函数形状，但是沿对角线方向的剖面是二次函数形状（例如，在第一象限沿直线$x=y$剖开，结果就是二次函数$(1-x)^2$）。

例2　二维高斯滤波器　如果f_1选用高斯函数，则得到的二维函数是（见图4-29）

$$\begin{aligned}f_2,g(x,y)&=\frac{1}{2\pi}\left(\mathrm{e}^{-x^2/2}\mathrm{e}^{-y^2/2}\right)\\&=\frac{1}{2\pi}\left(\mathrm{e}^{-(x^2+y^2)/2}\right)\\&=\frac{1}{2\pi}\mathrm{e}^{-r^2/2}\end{aligned}$$

如果将一维高斯滤波器沿着原点旋转，就产生一个旋转对称函数，结果与上面的形式一致。这种在旋转对称的同时还可分的性质是高斯函数独有的。不仅沿坐标轴方向的剖面具有高斯形状，而且沿中心外任何方向的剖面都具有高斯形状。

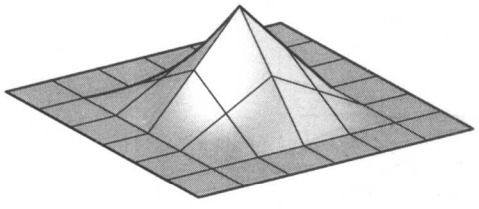


图4-28　可分二维帐篷式滤波器

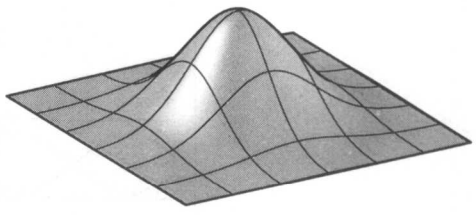


图4-29　二维高斯滤波器，具有可分性和径向对称性

与其他二维滤波器相比，可分滤波器的重要优点是实现的效率高。将a_2的定义代入离散卷积

的定义之中：

$$(a_2 \star b)[i,j]=\sum_{i'}\sum_{j'} a_1[i']a_1[j']b[i-i',j-j']$$

注意 $a_1[i']$ 与 j' 无关，因此可以从内部求和中提出去：

$$=\sum_{i'} a_1[i']\sum_{j'} a_1[j']b[i-i',j-j'] \quad \boxed{94}$$

把内部求和的表达式用$S[k]$表示为

$$S[k]=\sum_{j'} a_1[j']b[k,j-j']$$

$$(a_2 \star b)[i,j]=\sum_{i'} a_1[i']S[i-i'] \qquad (4\text{-}4)$$

利用这种形式的方程，就可以首先对每个i'值计算$S[i-i']$，并存储起来，然后再利用这些存储值计算外部和式的值。一眼看去该算法似乎不简单，因为为了算出所有的内部和，仍然要重复$(2r+1)^2$次。然而，如果想计算许多点$[i,j]$处的值，情况就不同了。

假设要计算$a_2 \star b$在[2, 2]和[3, 2]处的值，并且a_1的半径为2。根据式（4-4），我们可以发现，为了计算在[2, 2]点的值，需要知道$S[0]$, …, $S[4]$；为了计算在点[3, 2]的值，需要知道$S[1]$, …, $S[5]$。因此，在可分公式表示中，我们计算出S的所有6个值，并共享$S[1]$, …, $S[4]$（见图4-30）。

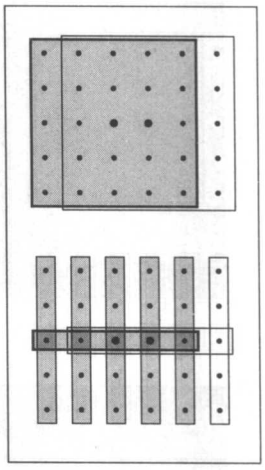


图4-30 （上图）利用25样本二维阵列计算两个输出点，（下图）沿着列方向进行一次滤波，然后用5样本一维阵列滤波

这种考虑对于大型滤波器具有重要的意义。一般情况下，用半径为r的滤波器对$m \times n$的二维图像进行滤波时，对每个像素需要计算$(2r+1)^2$次乘积。如果采用规模相同的可分滤波器对该图像进行滤波，只需要$2(2r+1)$次乘积（代价是需要中间存储），渐近复杂度从$O(r^2)$变到$O(r)$，这样就可以使用大型滤波器。

该算法是

```
function filterImage(image I,filter f)
r = f.radius
n_x = I.width
n_y = I.height
allocate storage array S[0,…,n_x − 1]
allocate image I_out[r,…,n_x − r − 1][r,…,n_y − r − 1]
initialize S and I_out to all zero
for y=r to n_y − r − 1 do
  for x=0 to n_x − 1 do
    S[x]=0
    for i=−r to r do
      S[x]=S[x]+f[i]I[x][y−i]
  for x=r to n_x − r − 1 do
    for i=−r to r do
      I_out[x][y]=I_out[x][y]+f[i]S[x−i]
return I_out
```

$\boxed{95}$

为了简便，函数将输出图像4边的r个像素进行了整理，从而避免了所有边界问题。实际应用中，存在很多处理边界的方法，见4.4.3节的内容。

4.4　图像信号处理

到目前为止我们简单讨论了采样、滤波和重构，大都是以一维信号为例。但是，从本章一开始我们就提过，信号处理在图形学中，最重要、最常见的应用对象是采样图像。下面就详细讨论图像的采样、滤波和重构。

4.4.1　离散图像滤波

也许最简单的卷积应用是用离散卷积处理图像。图像运算程序中，得到最广泛应用的是简单
96 的卷积滤波器。通过与一般低通滤波器（从盒式滤波器到高斯滤波器）进行卷积，实现对图像的模糊化处理，见图4-31。高斯滤波的结果看起来很平滑，所以经常用它进行模糊化处理。

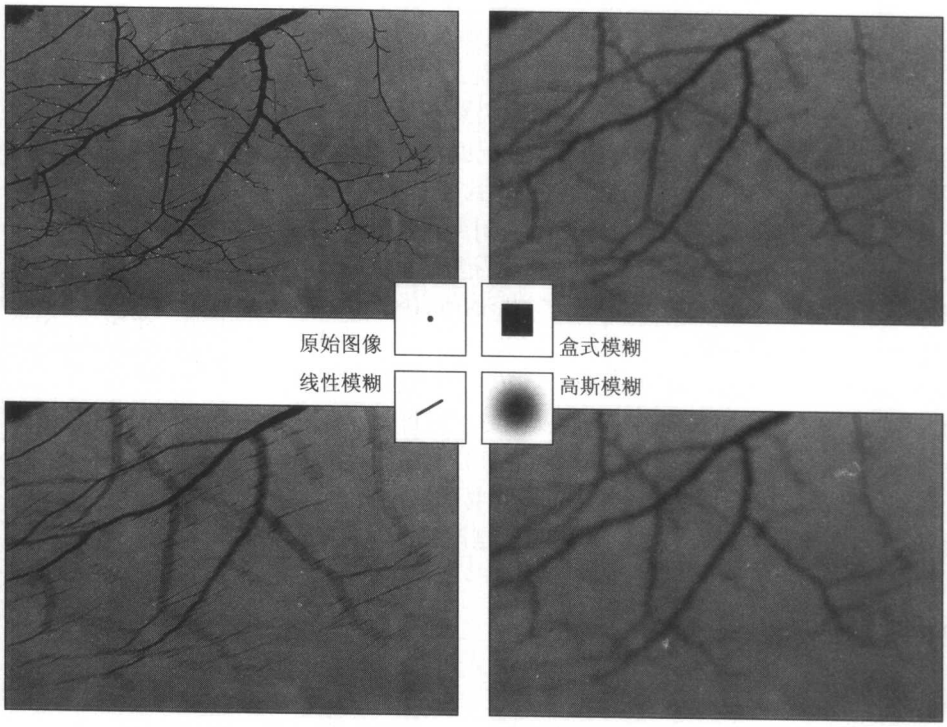


图4-31　图像与三种不同的滤波器求卷积，实现模糊化

与图像模糊化相对的运算是图像锐化。实现锐化的一种方式是采用“模糊掩模”：从原始图像中减去一幅模糊化图像的一部分α。重新调整尺度，以免图像的整体亮度发生改变。于是有：

$$\begin{aligned} I_{\text{sharp}} &= (1+\alpha)I + \alpha(f_{g,\sigma} \star I) \\ &= ((1+\alpha)d - \alpha f_{g,\sigma}) \star I \\ &= f_{\text{sharp}}(\sigma,\alpha) \star I \end{aligned}$$

其中$f_{g,\sigma}$是宽度为σ的高斯滤波器。利用离散脉冲d及卷积的分配律，可以把整个过程表示为含模糊宽度及锐化程度两个参数的单一滤波器（见图4-32）。

另一个结合使用两离散滤波器的实例是阴影。通常利用模糊的、移位的目标轮廓来构造柔和的阴影（见图4-33）。可以将移位运算表示为与偏离中心的一个脉冲的卷积：

$$\mathrm{d}_{m,n}(i,j)=\begin{cases}1 & i=m,\ j=n\\ 0 & \text{其他}\end{cases}$$

通过与两种滤波器的卷积运算，就实现了移位及模糊化：

$$\begin{aligned}I_{\text{shadow}} &= f_{g,\sigma}\star(\mathrm{d}_{m,n}\star I)\\ &= (f_{g,\sigma}\star\mathrm{d}_{m,n})\star I\\ &= f_{\text{shadow}}(m,n,\sigma)\star I\end{aligned}$$

上式中利用了结合律，将两次运算合成，变成含三个参数的一个滤波器。 97

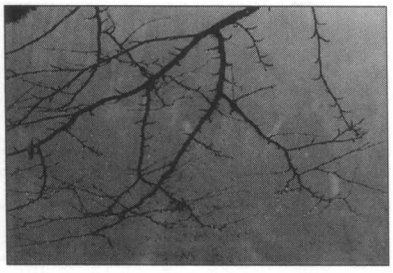


图4-32 用卷积滤波器锐化一幅图像

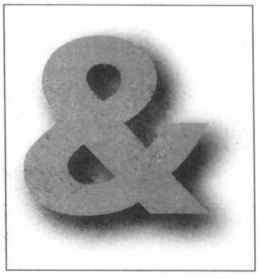


图4-33 柔和的阴影

4.4.2 图像采样中的反走样技术

在图像合成中，对用连续数学公式（至少是可计算任意点颜色、而不只是整数像点颜色的程序）表示的图像，常常需要产生一幅对应的采样图像。光线跟踪就是常见的例子。在第10章我们将学到更多关于光线跟踪的知识，以及具体的反走样技术。用信号处理语言来描述就是，对于连续的二维信号（图像），需要基于规则的二维点阵进行采样。如果不采取任何特殊措施，直接对图像进行采样，结果将显出各种各样的走样（见图4-34）。在图像的突变边缘上，会看到阶梯状伪影，称为“锯齿”。在有重复图案的区域，我们看到的带状物称为莫尔图案。

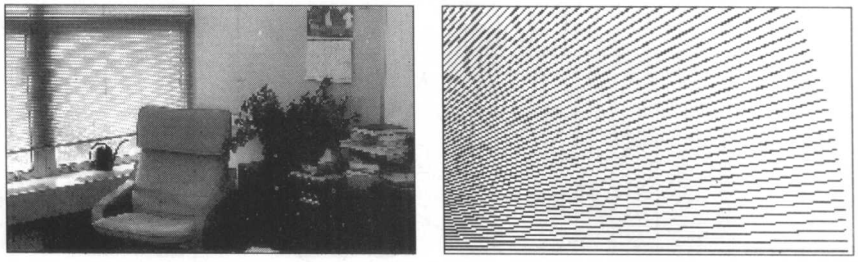


图4-34 图像中的两种伪影效果：左边显示规则纹理产生的莫尔图案，右边显示锯齿状直线

问题是图像中包含很多小尺度特征。在采样之前，要通过滤波去掉这些特征。回想式（4-3）中关于连续卷积的定义，我们需要在像素周围的区域上计算图像的平均值，而不仅仅是取个别点的值。具体方法将在第10章讨论。像盒式滤波器那样的简单滤波器，可以改善突变边缘的外观， 98
但仍然会产生一些莫尔图案（见图4-35）。高斯滤波器是非常平滑的，抑制莫尔图案的效果要好得多，但是会引起图像整体上变得模糊。这两个实例说明，在选择反走样滤波器时，要在清晰和反走样之间进行折中。

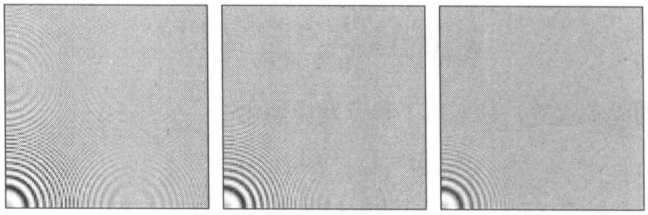


图4-35　三种不同的采样滤波器效果比较，反走样对象是一幅难处理的测试图像，该图像中包括一些圆，这些圆随半径变大而相距越来越近

4.4.3　重构与重采样

当对滤波要求比较高时，重采样是最常见的图像运算之一。重采样就是改变采样速度或者改变图像大小。

假设从数码摄像机获得大小为3000像素×2000像素的一幅图像，想在显示器上显示1280像素×1024像素。为了保持纵横比为3:2，需要对它重采样，大小变为1278像素×852像素。该如何做呢？

99

一种方法是，将这个过程看作是删除像素的过程。因为大小比例在2和3之间，所以要在留下的像素之间去掉一个或者两个像素。用这种方法来压缩一幅图像是可以的，但是结果得到的图像质量很低，图4-34中的图像就是使用像素删除方法得到的。像素删除的速度很快，但在交互操作中应该对被调整的图像生成一个预览。

调整图像大小的方法，可以看作是对图像进行重采样的过程：我们需要在特定栅格上的一组图像样本，该栅格定义在新的图像尺寸上，对根据输入样本重构的连续函数进行采样就可以得到这些样本（见图4-36）。这样看来，这个过程只是一系列标准图像运算：首先根据输入样本重构一个连续函数，然后对该函数进行采样，就像对其他连续图像采样一样。为了避免走样，在每一步都要选择适当的滤波器。

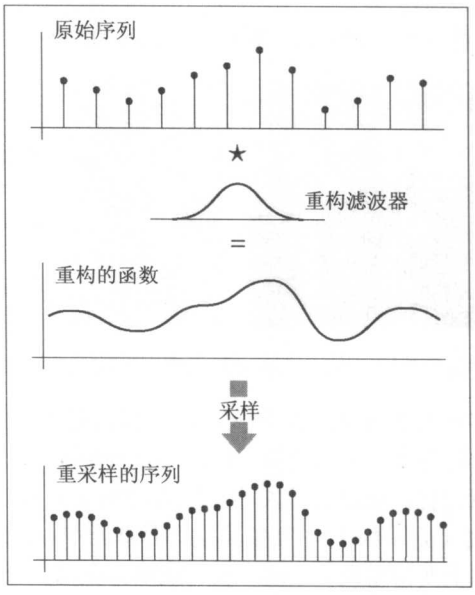


图4-36　图像重采样包括两个逻辑步骤，在代码中将这两步合二为一，成为一步运算。首先，用重构滤波器根据输入样本生成平滑的连续函数。然后，在新的栅格上对该函数进行采样，从而获得输出样本

参考如图4-37所示的小示例。如果原始图像是12×9像素，而新图像为8×6像素。在每一维，输出像素数是输入像素数的2/3，因此图像上的像素间隔是原始图像的3/2。

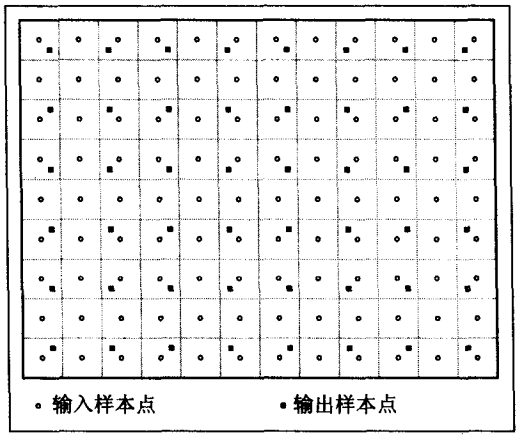


图4-37 重采样12×9的图像，得到8×6的图像。图中给出了输入、输出栅格的采样位置

为了给每个输出样本赋值，需要计算样本之间的图像值。像素删除算法就为我们提供了这样的一种方法：取输入图像中与样本最近的值作为输出值。该方法与下列方法等价：用一个像素宽的盒式滤波器重构图像，然后再点采样。

删除像素或者其他简单的滤波算法，作为一般重构-重采样过程的特例来实现时，考虑的不是它们的性能。实际上，由于盒式滤波器的不连续性，很难在一般框架下采用它。但是为了实现高质量的重采样，重构/采样框架具有一定的适应性。 100

为了明白算法的细节，最简单办法是先考虑一维情况，并讨论对序列的重采样。根据在4.2.5节定义的重构函数，写出最简单的实现方式：

```
function resample(sequence a,float x0, float Δx, int n,filter f)
create sequence b oflength n
for i=0 to n-1 do
  b[i] = reconstruct(a, f, x0 + iΔx)
return b
```

参数x_0表示根据旧序列样本得到的新序列第一个样本的位置。也就是说，如果第一个输出样本落在输入序列的样本3和样本4中间，那么x_0就等于3.5。

该程序重构一幅连续的图像，通过将输入序列与一连续的滤波器进行卷积，然后进行点采样。但不是说这两种运算按先后顺序进行，连续函数只在原理上存在，而且只在采样点上计算函数值。但是从数学上，用该函数计算函数$a \star f$的一系列样本点。

点采样似乎存在问题，因为我们刚说过一个信号应该用适当的平滑滤波器进行采样，以免出现走样。应该将重构函数与一个采样滤波器 g 求卷积，再进行点采样，即 $g \star (f \star a)$。由于此式与$(g \star f) \star a$结果一样，因此可以将采样滤波器与重构滤波器合在一起。我们所需要的就是一次卷积运算（见图4-38）。这种合成的重构和采样滤波器称为*重采样滤波器*。 101

对图像进行重采样时，一般在旧图像中指定一个*源矩形*，它确定了在新图像中要保留的部分。例如，根据第3章中的像素样本定位习惯，用来对整幅图像进行重采样的矩形为$(-0.5, n_x^{\text{old}} - 0.5) \times (-0.5, n_y^{\text{old}} - 0.5)$。给定源矩形$(x_l, x_h) \times (y_l, y_h)$，新图像在$x$方向和$y$方向的采样间隔分别为$\Delta x = (x_h - x_l)/n_x^{\text{new}}$，$\Delta y = (y_h - y_l)/n_y^{\text{new}}$。左下方的样本位于$(x_l + \Delta x/2, y_l + \Delta y/2)$。

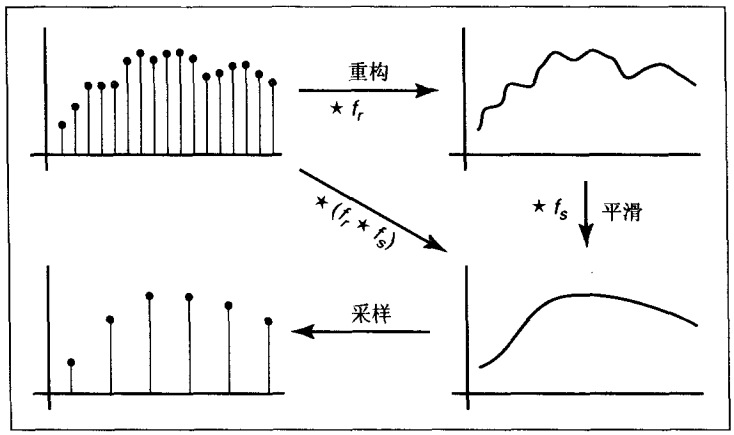


图4-38　重采样包括重构滤波和采样滤波。因为序列中的两卷积滤波器可以用单一的滤波器来代替，因此只需要一个重采样滤波器，它同时起到了重构和采样的作用

根据这一习惯，修改一维伪代码，并且将重构函数调用扩展到二重循环中，得到：

```
function resample(sequence a,float x_l, float x_h, int n,filter f)
create sequence b of length n
r = f.radius
x_0 = x_l+Δx/2
for i = 0 to n-1 do
  s = 0
  x = x_0+iΔx
  for j = ⌈x-r⌉ to ⌊x+r⌋ do
    s = s+a[i]f(x-i)
  b[i] = s
return b
```

该例程包含图像重采样的所有要素。最后还要提出一个问题，在图像的边缘处，这种简单处理会超过输入序列的范围，这时怎么办呢？这时存在以下解决办法：

- 在序列的末尾停止循环。这等价于在图像的所有边缘上补0。
- 修改阵列使序列末尾可以访问。也就是说，当考虑$a[-1]$时，返回$a[0]$。这等价于通过扩展最后一行或者最后一列来填补图像的边缘。
- 当接近边缘时修改滤波器，使得它不会超过序列的范围。

当重采样整幅图像时，第一种方法会产生模糊边缘，这一点不能令人满意。第二种方法容易实现。第三种方法可能是最好的选择。在图像边缘附近，修改滤波器的最简单方法是，对滤波器进行重新规范化，即用落在图像内部的部分滤波器之和去除该滤波器。通过这种方法，对于实际
102 图像样本，滤波器之和总为1，这样就维持图像亮度不变。从性能上考虑，希望在远离中心的边缘处，在滤波器半径之内处理这部分像素，因为这部分像素需要重新规范化；而图像中心包括更多的像素，不需要重新规范化。

重采样中选择滤波器很关键。要确定两个参数，即滤波器的形状和大小（半径）。由于滤波器既用作重构滤波器又用作采样滤波器，两个角色的任务都对滤波器的选择有影响。就重构来说，需要足够平滑的滤波器，这样在放大图像时可以避免出现走样现象，另外该滤波器还必须是没有波动的。就采样来说，滤波器必须足够大，以避免采样不足，还必须足够平滑，以避免莫尔图案生成。图4-39显示了这两种不同的需要。

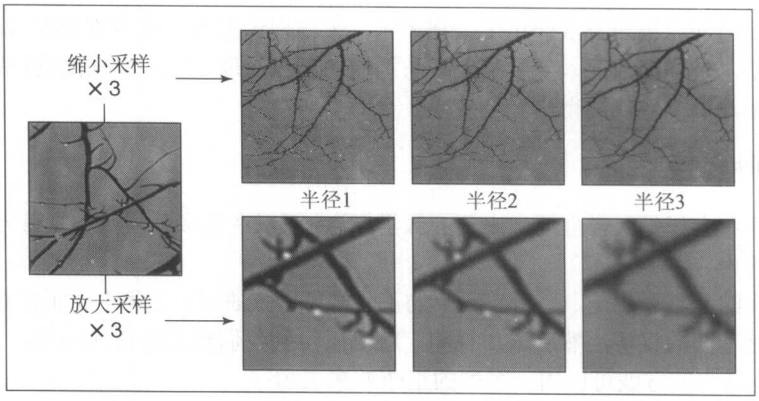


图4-39 对一幅图像进行放大和缩小采样，采样不同大小的滤波器效果

一般先选择滤波器形状，然后根据输入、输出的相对分辨率再进行缩放调整。分辨率低的那一个决定滤波器的大小。当对输出采样比对输入采样更粗略时（缩小采样，或缩小图像），对采样的平滑要求要比对重构的平滑要求高，因此根据输出样本的间距来确定滤波器的大小（图4-39中半径为3的图片）。另一方面，当对输出进行更精细采样时（放大采样，或放大图像），重构对平滑的要求更高（重构函数已经足够平滑，采样的速度可以比开始时更高），因此滤波器的大小取决于输入样本的间距（图4-39中半径为1的图像）。

选择滤波器本身是对速度和质量的折衷。一般选用盒式滤波器（当速度很重要时）、帐篷式滤波器（质量适中）或者分段三次滤波器（质量高）。在分段三次情况下，平滑度可以通过在f_B和f_C之间的插值来调整，这时可选用Mitchell-Netravali滤波器。 103

与图像滤波一样，可分滤波器能够加快速度。基本思想是，首先对所有的行重采样，产生宽度改变但是高度不变的图像，然后对图像的列重采样，得到最后的结果（见图4-40）。利用这种优化措施，可以直接修改前面给出的伪代码。

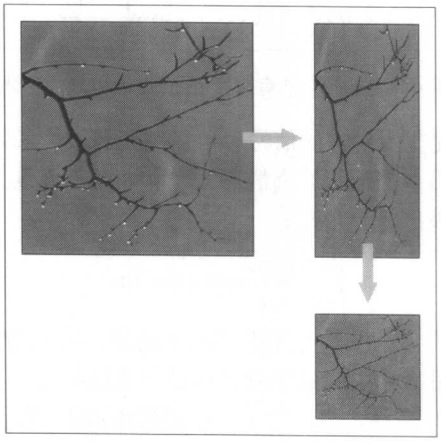


图4-40 利用可分方法重采样一幅图像

4.5 采样理论

如果只对实现感兴趣，下面的内容就可以略过不看。前面所学的算法和建议足以让我们实现采样与重构算法，并能够得到完美的结果。然而，采样问题涉及更加深入的数学理论。回顾历史

可以发现，无线电通讯领域首次使用了采样表示。采样理论回答了很多难以回答的问题。

最重要的是，采样理论深入揭示了采样与重构的工作原理。它为大家提供特别的推理工具，
104 从而知道如何编写最高效代码得出最佳效果。

4.5.1 傅里叶变换

傅里叶变换和卷积是采样理论中主要的数学概念。可以在很多分析数学及信号处理的书中读到傅里叶变换的内容。

傅里叶变换的基本思想是，用所有频率的正弦波（正弦曲线）之和来表示任意函数。不同频率的正弦波具有不同的权值，然后求加权和，就可以得到我们想要的任何函数。

例如，图4-41中的方波可以用一系列的正弦波来表示：

$$\sum_{n=1,3,5,\cdots}^{\infty} \frac{4}{\pi n}\sin 2\pi nx$$

这个傅里叶级数以频率为1.0的正弦波（$\sin 2\pi x$）开始，该频率与方波频率一样。其他各项是随频率逐渐变小的修正项，用以减小波动。无限个正弦波的加权和就重新生成了方波。注意和式中的每一项的频率都是方波频率的整数倍。这是因为其他频率项产生的结果，其周期与方波的周期不同。

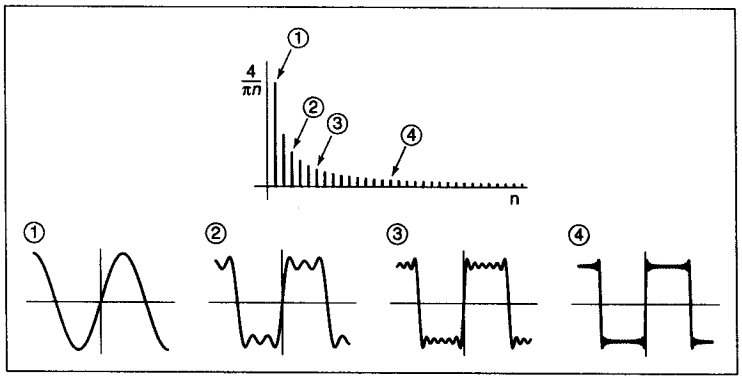


图4-41　方波通过有限个正弦波之和来近似表示

令人惊奇的事实是，即使非周期性的信号也可以用正弦波之和这种方式表示，只不过这时需
105 要更多的正弦波。下面讨论的不是正弦波离散序列之和，而是连续正弦曲线族的积分。例如，盒式函数可以表示为一族余弦波的积分形式：

$$\int_{-\infty}^{\infty} \frac{\sin \pi u}{\pi u}\cos 2\pi ux \, \mathrm{d}u \tag{4-5}$$

式（4-5）中的积分，是许多余弦波在无限空间上的和，频率u的余弦的权值为$(\sin\pi u)/\pi u$。随着被考虑的频率越来越高，最后结果收敛于盒式函数（见图4-42）。当对函数 f 用这种方式表示时，该权值就称为 f 的傅里叶变换，表示为 $\hat{f}$，它是频率u的函数。函数 $\hat{f}$ 告诉我们如何通过对一组正弦波进行积分来构建f：

$$f(x)=\int_{-\infty}^{\infty} \hat{f}(u)\mathrm{e}^{2\pi iux}\mathrm{d}u \tag{4-6}$$

式（4-6）称为傅里叶逆变换（IFT），因为它的积分函数是f的傅里叶变换，积分结果是 f。[1]

1. 注意“傅里叶变换”既指函数 $\hat{f}$，又指从 f 到 $\hat{f}$ 的运算。这种具有二义性的用法已经成为标准用法。

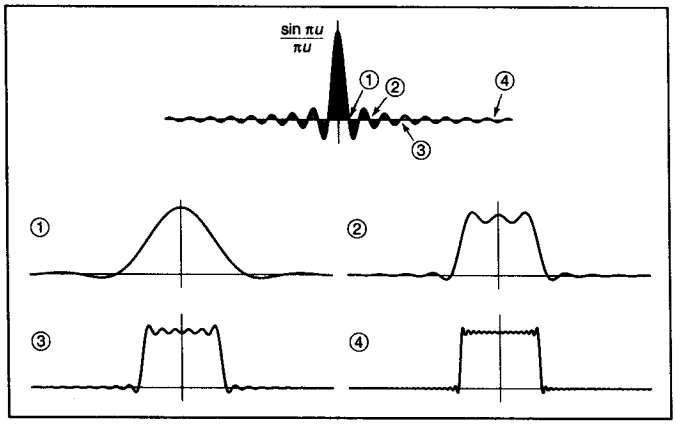


图4-42 用余弦函数的积分近似表示盒式函数。下面是四个截止频率情况

注意式（4-6）中，复数项的指数形式$e^{2\pi iux}$代替了前面公式中的余弦函数。同样，$\hat{f}$也是复数值函数。表示正弦函数的相位和频率时，要用到复数。表示任何不对称过零点的函数时，都要用到复数。$\hat{f}$的幅度称为傅里叶频谱。对我们来说，有傅里叶频谱已经足够了，我们不关心相位，也不直接用复数。 106

实践证明，由f计算$\hat{f}$，和由$\hat{f}$计算f非常类似：

$$\hat{f}(x)=\int_{-\infty}^{\infty} f(x)e^{-2\pi iux}dx \tag{4-7}$$

式（4-7）称为傅里叶变换（FT）。与前面的傅里叶逆变换相比，唯一的不同就是指数的符号不一样，实际上这只是一个技术细节。对我们来说，可以将傅里叶逆变换和傅里叶逆变换看作是相同的运算。

有时符号f–$\hat{f}$显得不方便，我们就把f的傅里叶变换表示为$\mathcal{F}\{f\}$，将$\hat{f}$的傅里叶逆变换表示为$\mathcal{F}^{-1}\{\hat{f}\}$。

经常要用到函数及其傅里叶变换之间的关系，关系的表示方式有多种。下面是本章后面要用到的几个关系（大都很容易进行验证）：

- 函数及其傅里叶变换的平方积分相等：

$$\int (f(x))^2 dx=\int \hat{f}(u)^2 du$$

物理含义是，两者有相等的能量（见图4-43）。

特别地，函数放大a倍，其傅里叶变换也放大a倍，即$\mathcal{F}\{af\}=a\mathcal{F}\{f\}$。

- 沿x轴拉伸函数b倍，对应的傅里叶变换沿u轴收缩$1/b$（见图4-44）：

$$\mathcal{F}\{f(x/b)\}=b\hat{f}(bx)$$

（需要用b进行重新规范化，以保证能量相等。）

这就意味着，如果对一族宽度和高度都不相等的函数（如中心位于零点的所有盒式函数）感兴趣，那么只要知道一个典型函数的傅里叶变换（如宽度和高度都等于1的盒式函数），就容易知道所有其他缩小或放大的函数的傅里叶变换。例如，根据式（4-5），可以立即得出高度为a宽度为b的盒式函数的傅里叶变换为： 107

$$ab\frac{\sin \pi bu}{\pi bu}$$

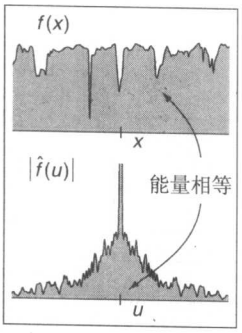


图4-43　函数及其傅里叶变换的平方积分相等

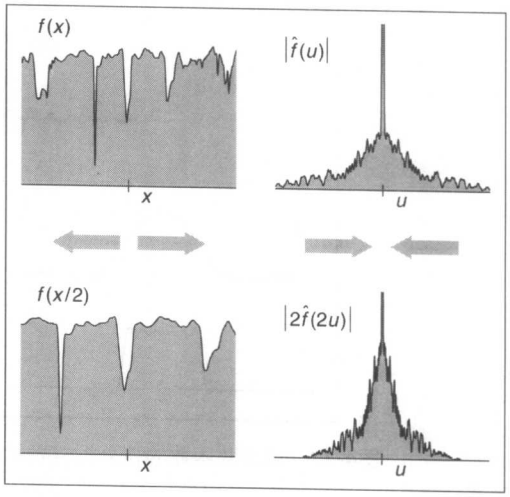


图4-44　在空域中沿x轴缩放信号，则在频域中沿u轴逆缩放

- f 的平均值等于 $\hat{f}(0)$。这一点很有意义，因为 $\hat{f}(0)$是信号的零频率成分（如果是电压，就是直流成分）。
- 如果f是实函数（对我们研究的问题来说，f总是实函数），则 $\hat{f}$ 就是偶函数。也就是说，$\hat{f}(u)=\hat{f}(-u)$。反之，如果f是偶函数，那么 $\hat{f}$ 就是实函数（对我们研究的问题来说，$\hat{f}$ 一般不是实函数。但是我们只关心 $\hat{f}$ 的幅度）。

4.5.2　卷积与傅里叶变换

特别值得一提的是傅里叶变换的最后一个性质，即它与卷积的关系（见图4-45）。用公式表示如下：

108

$$\mathcal{F}\{f\star g\}=\hat{f}\,\hat{g}$$

两函数卷积的傅里叶变换，等于它们的傅里叶变换的乘积。下面是我们熟悉的对称公式：

$$\hat{f}\star\hat{g}=\mathcal{F}\{fg\}$$

两函数的傅里叶变换的卷积，等于它们的乘积的傅里叶变换。这些结论都可以根据定义直接推出来。

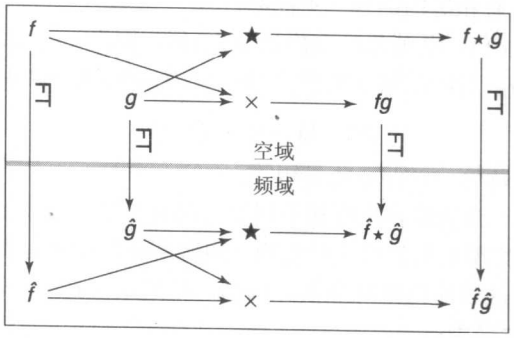


图 4-45　本图直观地显示了卷积与乘积的关系。下面的两种情况结果相同：（1）在空域中将f与g相乘，然后将结果变换到频域；（2）先分别把f和g变换到频域，然后再求卷积。同样，下面的两种情况结果相同：（1）在空域中求f与g的卷积，然后将结果变换到频域；（2）先分别将f与g变换到频域，然后再求乘积

这种关系，是傅里叶变换在研究采样与重构中得到应用的主要原因。我们已经知道，空域中的采样、滤波和重构都是卷积运算。现在，傅里叶变换带来了频域，在频域上述运算都变成积运算了。

4.5.3 傅里叶变换举例

我们已经掌握了傅里叶变换的有关知识，下面讨论个别函数的傅里叶变换。特别要研究4.3.1节中提到的滤波器，图4-46显示了这些函数及其傅里叶变换。已经知道盒式函数的傅里叶变换为

$$\mathcal{F}\{f_{\text{box}}\} = \frac{\sin \pi u}{\pi u} = \text{sinc}\pi u$$

函数[1]$\sin x/x$非常重要，它有自己的名字，即$\text{sinc}\, x$。 109

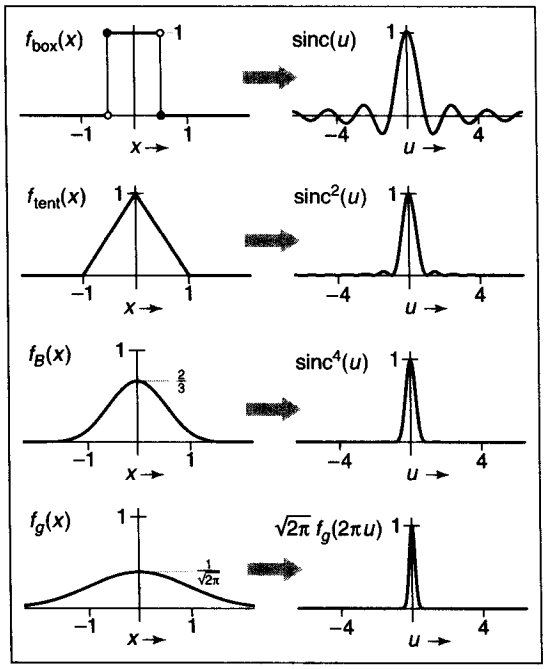


图4-46 盒式函数、帐篷式函数、B样条函数、高斯函数的傅里叶变换

帐篷式函数是盒式函数与自身的卷积结果，因此它的傅里叶变换就等于盒式函数傅里叶变换的平方：

$$\mathcal{F}\{f_{\text{tent}}\} = \frac{\sin^2 \pi u}{\pi^2 u^2} = \text{sinc}^2 \pi u$$

继续下去，就可以得到B样条滤波器的傅里叶变换（见习题3）：

$$\mathcal{F}\{f_{\text{B}}\} = \frac{\sin^4 \pi u}{\pi^4 u^4} = \text{sinc}^4 \pi u$$

高斯滤波器的傅里叶变换非常特别：

$$\mathcal{F}\{f_{\text{G}}\} = \text{e}^{-(2\pi u)^2/2}$$

1. 我们可能会注意到，u=0时$\sin \pi u/\pi u$没有定义，但函数却是连续穿过0点，我们认为u=0时该函数取有限值1。

[110] 结果是另一个高斯函数！标准差为1.0的高斯函数，其傅里叶变换是标准差为1/2π的高斯函数。

4.5.4 采样理论中的狄拉克脉冲

脉冲在采样理论中是非常有用的，因为可以用来讨论在连续函数与傅里叶变换背景下的样本。样本具有位置和值，我们用某位置的脉冲及其系数值来表示样本。一个位置在a、值为b的样本表示为$b\delta(x-a)$。用这种方式可以表示在a处对函数$f(x)$的采样运算，结果为$f(a)\delta(x-a)$。

因此，在一系列等间隔的点处，对函数进行采样，可以表示为等间隔的脉冲系列之和与该函数相乘，这些脉冲系列称为脉冲链（见图4-47）。一个周期为T的脉冲链，意味着脉冲之间的间隔为T，即

$$s_T(x)=\sum_{i=-\infty}^{\infty}\delta(x-Ti)$$

设s_1是整数频率处的脉冲序列，则s_1的傅里叶变换与s_1相同。当用正弦函数去乘脉冲链并且积分时，会验证上面的结果。所有整数频率的正弦函数值之和发散。对非整数频率，由于相互抵消总和变为0，而对于整数频率和趋向于+∞。

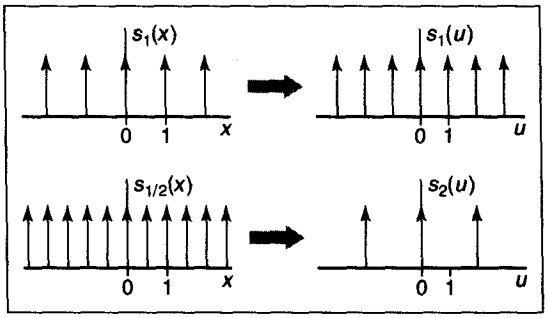


图4-47 脉冲链。脉冲链的傅里叶变换是另一个脉冲链。在空域中改变脉冲链的周期，将导致频域中周期的逆向变化

由于傅里叶变换具有膨胀性质，可以推测周期为T的脉冲链（就像是s_1经膨胀得到的），其傅
[111] 里叶变换是周期为$1/T$的脉冲链。在空域中的采样越密，在频域中脉冲之间的间隔越远。

4.5.5 采样与走样

我们从频域的角度，建立了理解采样与重构过程的数学机制。引入傅里叶变换的重要优点是，变换后对信号的卷积滤波作用看得更加清楚，同时更加明确地解释了在采样与重构中为什么要进行滤波。

对原始连续信号进行采样。一般说来，原始信号的傅里叶变换可以包含任何频率成分。不过对于多数信号，特别是图像信号，我们总是希望成分含量随频率的升高而下降。而且，图像信号中的零频率分量往往最大。要记住一点，零频率或者直流成分是对整幅图像的积分，由于图像都取正值，所以直流分量的数字往往很大。

如果在采样与重构时不进行专门滤波，那么傅里叶变换将会是怎样的？（见图4-48）。对信号进行采样时，将采样看作与脉冲链的乘法运算。设样本信号为fs_T。根据乘法-卷积的性质，样
[112] 本信号的傅里叶变换是$\hat{f}\star\hat{s}_T=\hat{f}\star s_{1/T}$。

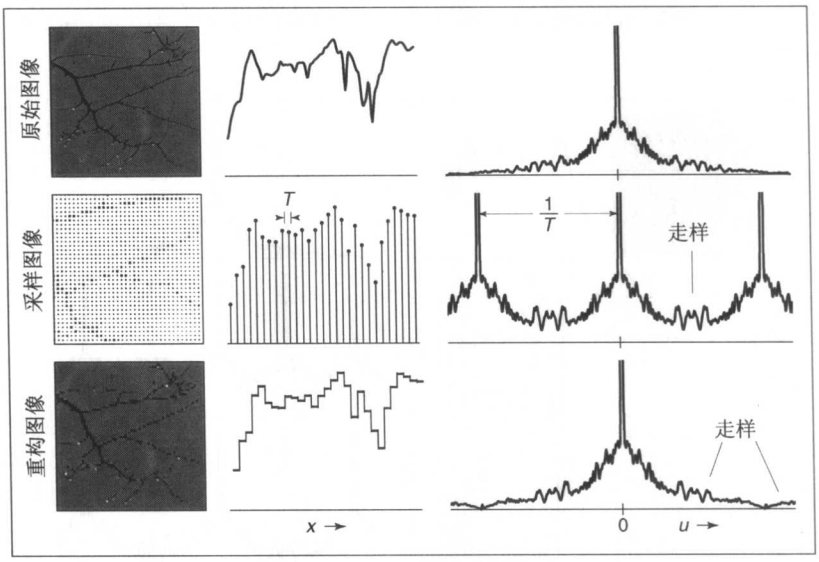


图4-48 无滤波的采样与重构。采样产生了走样谱，与基频谱重叠混合。通过盒式滤波器重构，则从走样谱中吸收了更多的信息。结果信号存在严重的走样

前面我们学过δ是卷积的单位。这就意味着：

$$\left(\hat{f} \star s_{1/T}\right)(u) = \sum_{i=-\infty}^{\infty} \hat{f}(u - i/T)$$

也就是说，与脉冲链的卷积结果，得到的是一系列等间距的f频谱的复制品。对于这种奇特结果的直观解释是，所有的复制品表明了这样的事实（见4.1.1节）：它们的频率都是采样频率的整数倍，一旦完成了采样，这些信号之间就难以区分，它们将产生完全相同的样本集。原始频谱称为*基频谱*，而那些复制品称为*走样谱*。

如果信号频谱的复制品相互重叠就会产生麻烦，当信号的主要频率超过采样频率的一半时，就会发生这种重叠。当这种情况发生时，频谱相加，不同频谱的信息不可避免地混合在一起。这是走样可能发生的第一种形式。如果走样真的在这时发生，原因是由于采样不足，即对信号的采样频率太低。

如果采用最近邻技术重构信号，这等价于与宽度为1的盒式函数进行卷积。（用来实现这一任务的离散-连续卷积，与表示样本的脉冲序列连续卷积是相同的。）根据卷积-乘积性质，重构信号的频谱将等于采样信号频谱与盒式函数频谱的乘积。结果重构的傅里叶变换，包含基频谱（在高频处有些被削弱了），加上被削弱的所有走样谱复制品。因为盒式函数具有相当宽的傅里叶变换，因此这些被削弱的走样谱变得很突出。由于重构滤波器不合适，带来了走样，这是走样产生的第二种形式。在图像上这些走样成分本身显示为方块图案，这是最近邻重构法的特点。

1. 避免采样走样

想进行高质量的采样与重构，需要适当地选择采样与重构滤波器。从频域的观点来看，采样时进行低通滤波是为了限制信号的频率范围，使得走样谱不会与基频谱重叠。图4-49显示采样速度对样本信号的傅里叶变换的影响。较高的采样速度，使走样谱离得更远，最后不论什么样的重叠都没有关系。

113

关键的指标是，频谱的宽度必须小于复制品之间的距离，也就是说，信号中出现的最高频率必须小于采样频率的一半。这个指标称为*尼奎斯特指标*（Nyquist criterion），允许的最高频率称为*尼奎斯特频率*或者*尼奎斯特极限*。*尼奎斯特-香农采样定理*（Nyquist-Shannon sampling theorem）

规定，频率不超过尼奎斯特极限的信号（或者说带宽限制为尼奎斯特频率的信号），原则上可以根据样本进行准确重构。

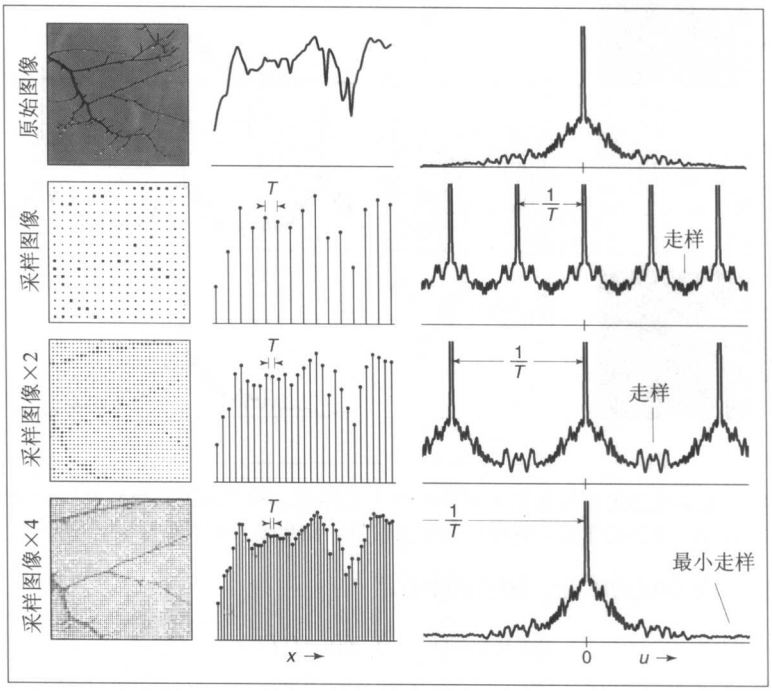


图4-49　采样速度对样本信号频谱的影响。高采样速度使走样谱离得更远，从而减少重叠带来的问题

对特殊信号采样足够高的采样速度，就不必使用采样滤波器。但对于频率范围宽的信号（例如有锐利边缘的图像），就必须使用采样滤波器，以便在采样之前对信号进行带宽限制。图4-50显示了频域中三个低通滤波器的效果。图4-51显示了采样时利用这些滤波器的效果。即使滤波中没有
114 频谱重叠，将信号与低通滤波器进行卷积，也可以缩小频谱、消除重叠，并且生成能充分表示滤波后信号的样本集。当然，我们丢失了高频率成分，但是这个效果比信号混杂、产生走样的结果要好。

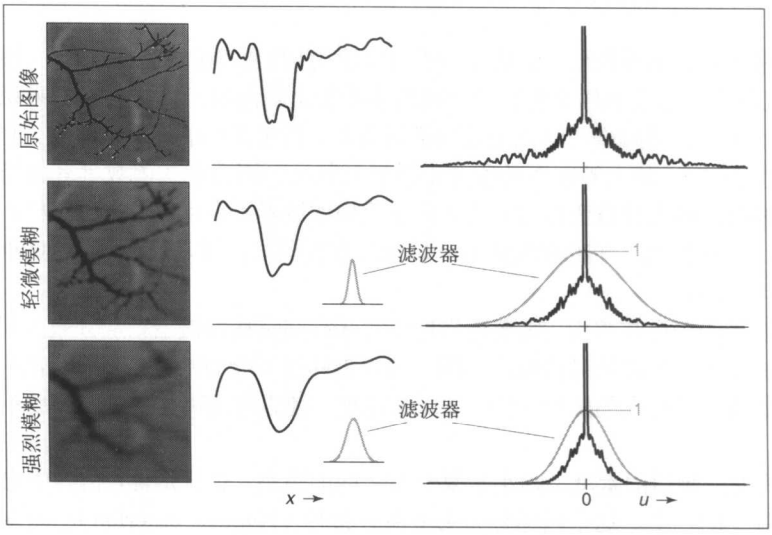


图4-50　采用低通（平滑）滤波器使信号频谱变窄

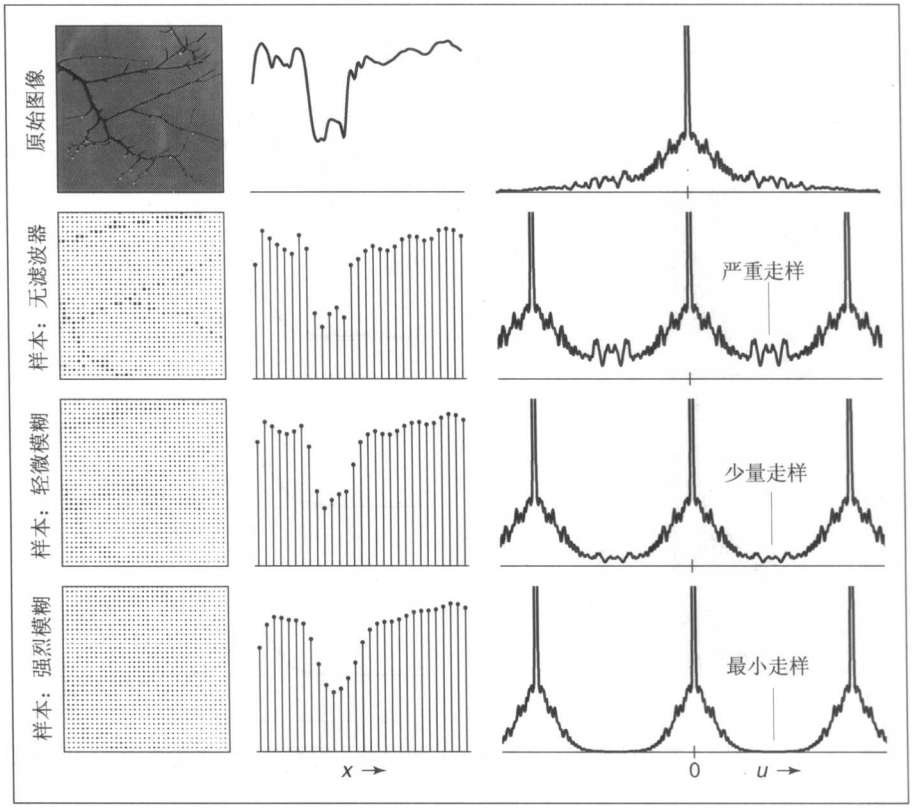


图4-51 图4-50中的低通滤波器在采样过程中如何避免产生走样。低通滤波器使信号频谱变窄，从而使复制品之间的重叠减少，走样谱中的高频成分对基频的干扰更少

2. 避免重构走样

从频域的观点看，重构滤波器的工作就是去除走样谱同时保留基频谱。在图4-48中，可以看到最简单的重构滤波器，即盒式滤波器，确实削弱了走样谱。最重要的是，滤波器完全阻止了所有走样谱的直流尖峰信号。这是所有合理的重构滤波器的特征：在频域空间，在采样频率的倍频处都有零值。实践证明这一特点与空域中的无波动性质是等价的。

因此，好的重构滤波器要成为好的低通滤波器，还要能够完全阻止所有采样频率倍频处的频谱。使用重构滤波器的目的不同于使用盒式滤波器，前者是为了更加彻底地消除走样谱，减少泄漏到重构信号中的高频干扰，同时尽可能不影响基频谱。图4-52显示了在重构过程中采用不同滤波器的结果。正如我们所看到的，盒式滤波器是非常“漏”的，会产生很多走样，即使采样速度很高也如此。帐篷式滤波器，结果是线性插值，更加削弱了高频干扰，产生较少的走样。而B样条滤波器非常平滑，能非常有效地控制走样谱。它同样还能对基频谱进行一些平滑。这就是我们前面提到的在平滑与走样之间的折衷。

3. 避免重采样走样

在重采样中，将重构与采样结合在一起运算时，采用的原理与前面的相同，只是用一个滤波器完成重构和采样两项工作。图4-53显示重采样滤波器如何去除走样谱，并使频谱变得足够窄，以便能以新的采样速度采样。

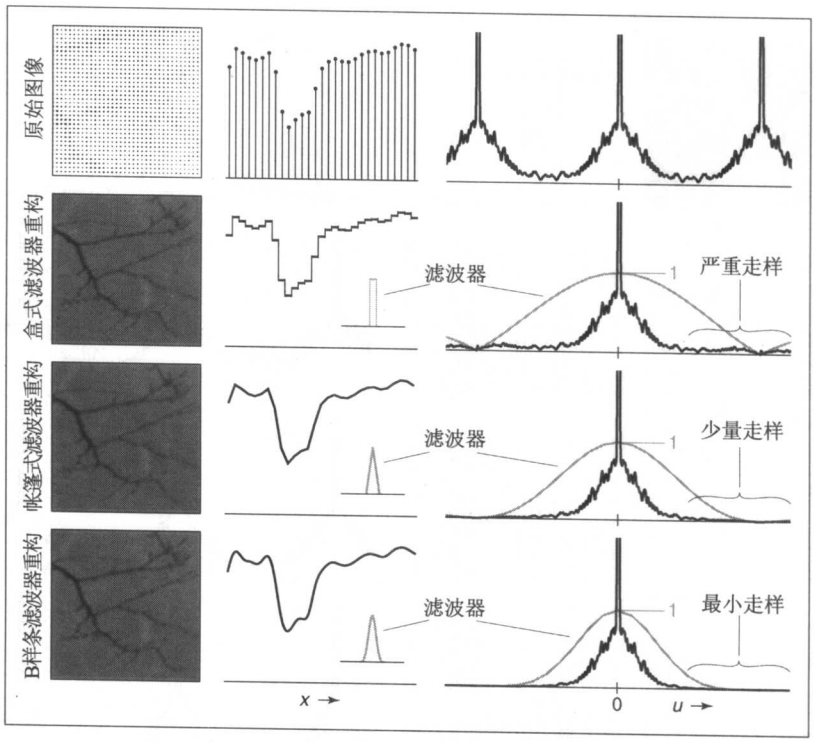


图4-52　频域中不同重构滤波器的效果。一个好的重构滤波器，能有效地削弱走样谱，同时保留基频谱

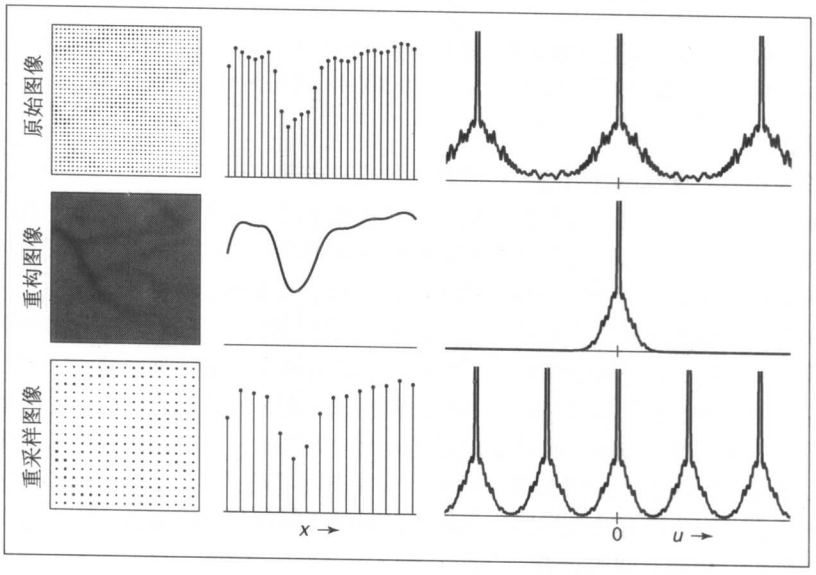


图4-53　频域中的重采样。重采样滤波器既重构信号（去除走样谱），又进行带宽限制（减少其带宽），以便以新的速度采样

4.5.6　理想滤波器与实用滤波器

115 ~ 117

根据频域分析，从逻辑上得出如下结论：频域中的盒式滤波器是采样与重构的理想滤波器。在这两种运算中该滤波器都能避免走样，而不减少尼奎斯特频率之下的频率范围。

我们已经知道，傅里叶逆变换与傅里叶变换在本质上是相同的，因此傅里叶变换为盒式滤波器的空域滤波器，就是函数$\sin\pi x/\pi x = \mathrm{sinc}\pi x$。

然而，实际在采样或者重构中很少用到sinc滤波器，因为它不实用。虽然它的频域指标最优，但在很多应用中并不能产生最佳结果。

就采样来说，sinc滤波器无限延伸，而且远离中心时函数值下降速度相对较慢，这是它的一个缺点。另外，对于某些类型的采样，sinc滤波器的负向突起也是个问题。高斯滤波器是非常好的采样滤波器，即使是在输入信号中去除高频图案的困难任务，它也能得出很好的结果，这是因为它的傅里叶变换以指数速度衰减，而没有能让走样泄漏的突起。对一些不太困难的情况，帐篷式滤波器就足够了。

就重构来说，sinc函数的大小也会产生问题。更重要的是，多波动在重构信号中引起“突起”干扰。

习题

1. 证明离散卷积满足交换律与结合律。对连续卷积进行同样地证明。
2. 离散-连续卷积不满足交换律，因为两种参数的类型不同。证明离散-连续卷积满足结合律。
3. 证明B样条函数是4个盒式函数的卷积。
4. 利用下面的（错误）定义（见4.2.2节的第2小节中的脚注），通过证明卷积满足交换律与结合律，证明卷积定义的“跳变”是必要的。

$$(a \star b)[i] = \sum_j a[j]b[i+j]$$

5. 证明$\mathcal{F}\{f \star g\} = \hat{f}\,\hat{g}$和$\hat{f} \star \hat{g} = \mathcal{F}\{fg\}$。

118

第5章 线性代数

最常用的图形编程工具，可能就是对点和向量进行变换的矩阵了。在下一章，我们会看到如何将向量表示成列矩阵，以及如何通过与方阵相乘将该向量表示在另一个基中。同样也将看到，如何通过这种乘法，实现对向量的缩放、旋转和平移。本章从几何的角度，复习一下线性代数的有关知识。熟悉线性代数的读者可以跳过这一章。但即使熟悉线性代数，浏览本章关于行列式展开、奇异值与特征值分解的内容，也会有所收获。

5.1 行列式

我们通常认为行列式是在研究线性方程组解法时被提出的。然而，从图像学的角度，认为行列式是向量相乘的另一种方法。对于二维向量$\boldsymbol{a}$和$\boldsymbol{b}$，行列式$|\boldsymbol{ab}|$表示由$\boldsymbol{a}$和$\boldsymbol{b}$形成的平行四边形的面积（见图5-1）。这个面积是有符号的，如果$\boldsymbol{a}$和$\boldsymbol{b}$符合右手法则，则面积为正；如符合左手法则，则面积为负。这就意味着$|\boldsymbol{ab}| = -|\boldsymbol{ba}|$。二维空间中的右手法则含义是，第一个向量沿逆时针方向旋转，经过最小的夹角到达第二个向量。在三维空间，行列式包含三个向量。对于三个三维向量$\boldsymbol{a}$、$\boldsymbol{b}$和$\boldsymbol{c}$，行列式$|\boldsymbol{abc}|$表示由这三个向量形成的平行六面体的体积，这个体积也是有正负的（见图5-2）。要计算二维行列式的值，首先要了解行列式的一些性质。可以注意到，将平行四边形的一边放大k倍，它的面积也同样放大k倍（见图5-3）：

$$|(k\boldsymbol{a})\boldsymbol{b}| = |\boldsymbol{a}(k\boldsymbol{b})| = k|\boldsymbol{ab}|$$

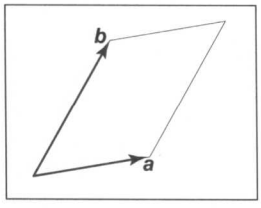


图5-1 平行四边形的有符号面积为$|\boldsymbol{ab}|$，图中的面积为正

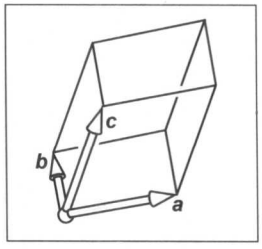


图5-2 平行六面体的有符号体积为$|\boldsymbol{abc}|$，图中的体积为正，因为向量分布服从右手法则

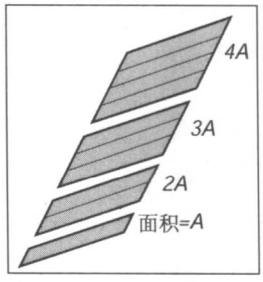


图5-3 如果平行四边形的一边进行缩放，其面积也按同样的比例缩放

另外还应注意到，在不改变底和高的情况下，任意扭动平行四边形都不会改变它的面积（见图5-4）：

$$|(\boldsymbol{a}+k\boldsymbol{b})\boldsymbol{b}| = |\boldsymbol{a}(\boldsymbol{b}+k\boldsymbol{a})| = |\boldsymbol{ab}|$$

最后，行列式还具有如下性质：

$$|\boldsymbol{a}(\boldsymbol{b}+\boldsymbol{c})| = |\boldsymbol{ab}| + |\boldsymbol{ac}| \tag{5-1}$$

如图5-5所示，移动两个平行四边形的公共边，形成一个平行四边形，而不改变原来两个平行四边形的面积。

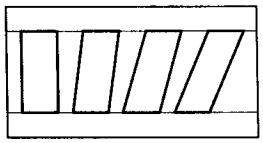


图5-4 平行四边形扭动但面积不变。这4个平行四边形底和高相同，因此面积也相同

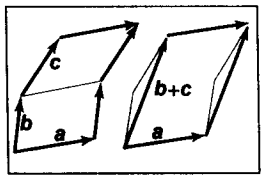


图5-5 式（5-1）隐含的几何图形。左边的两个平行四边形，经变形可以形成右边的一个平行四边形

下面是$\boldsymbol{a}$和$\boldsymbol{b}$的笛卡儿表达式：

$$\begin{aligned}|\boldsymbol{ab}| &= |(x_a\boldsymbol{x} + y_a\boldsymbol{y})(x_b\boldsymbol{x} + y_b\boldsymbol{y})| \\ &= x_a x_b|\boldsymbol{xx}| + x_a y_b|\boldsymbol{xy}| + y_a x_b|\boldsymbol{yx}| + y_a y_b|\boldsymbol{yy}| \\ &= x_a x_b(0) + x_a y_b(+1) + y_a x_b(-1) + y_a y_b(0) \\ &= x_a y_b - y_a x_b\end{aligned}$$

化简上式时用到了：对于任何向量$\boldsymbol{v}$，都有$|\boldsymbol{vv}|=0$。因为在这种情况下，平行四边形的所有边都和$\boldsymbol{v}$在同一直线上，因此面积为0。

在三维空间中，三维向量$\boldsymbol{a}$、$\boldsymbol{b}$和$\boldsymbol{c}$的行列式表示为$|\boldsymbol{abc}|$。对于平行六面体，同样用笛卡儿表达式表示向量，可以推出和平行四边形类似的结论。和二维情况类似，可以推出：

$$\begin{aligned}|\boldsymbol{abc}| &= |(x_a\boldsymbol{x} + y_a\boldsymbol{y} + z_a\boldsymbol{z})(x_b\boldsymbol{x} + y_b\boldsymbol{y} + z_b\boldsymbol{z})(x_c\boldsymbol{x} + y_c\boldsymbol{y} + z_c\boldsymbol{z})| \\ &= x_a y_b z_c - x_a z_b y_c - y_a x_b z_c + y_a z_b x_c + z_a x_b y_c - z_a y_b x_c\end{aligned}$$

正如我们所看到的，采用这种方式计算行列式，随着维数的增加表达式将变得很复杂。在5.2.3节将讨论其他不易出错的行列式计算方法。

将一个向量表示为另外两个向量的线性组合时，自然就要用到行列式。例如，将向量$\boldsymbol{c}$表示为向量$\boldsymbol{a}$和$\boldsymbol{b}$的线性组合：

$$\boldsymbol{c} = a_c\boldsymbol{a} + b_c\boldsymbol{b}$$

120

从图5-6可以看出：

$$|(b_c\boldsymbol{b})\boldsymbol{a}| = |\boldsymbol{ca}|$$

上式成立的原因是，这些平行四边形只是做了一些等面积的拉伸。从上式可以求出b_c：

$$b_c = \frac{|\boldsymbol{ca}|}{|\boldsymbol{ba}|}$$

同样可以得出另一个参数：

$$a_c = \frac{|\boldsymbol{bc}|}{|\boldsymbol{ba}|}$$

这就是二维情况下的克莱姆法则，将在5.2.5节讨论这一法则。

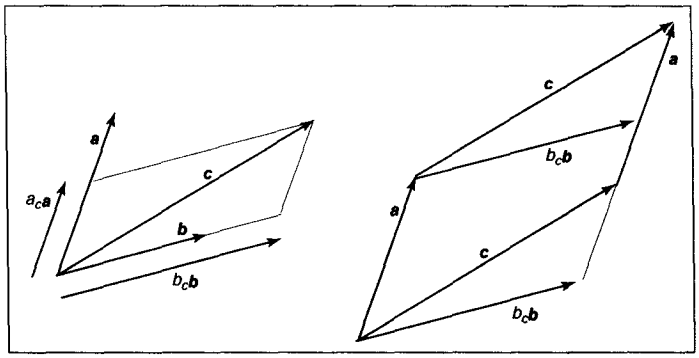


图5-6　左图中，向量c通过两个基向量表示为$a_c a+b_c b$。右图中，可以看出，a、c构成的平行四边形，是由$b_c b$、a构成的平行四边形变形得到的

5.2 矩阵

矩阵就是由遵循一定算术法则的数字元素所组成的阵列。一个两行三列的矩阵如下：

$$\begin{bmatrix} 1.7 & -1.2 & 4.2 \\ 3.0 & 4.5 & -7.2 \end{bmatrix}$$

121

在图形学中，因各种目的频繁用到矩阵，例如空间变换等。在我们的讨论中，假定矩阵中的元素全是实数。本章主要讨论矩阵的运算机制及方阵的行列式。方阵就是行数和列数相等的矩阵。

5.2.1 矩阵运算

矩阵乘以一个常数，等于矩阵中的每个元素都乘以这个常数，例子如下：

$$2\begin{bmatrix} 1 & -4 \\ 3 & 2 \end{bmatrix}=\begin{bmatrix} 2 & -8 \\ 6 & 4 \end{bmatrix}$$

矩阵加法，将对应的元素分别相加，例子如下：

$$\begin{bmatrix} 1 & -4 \\ 3 & 2 \end{bmatrix}+\begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}=\begin{bmatrix} 3 & -2 \\ 5 & 4 \end{bmatrix}$$

矩阵乘法，将第一个矩阵的行与第二个矩阵的列相乘：

$$\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & & \vdots \\ \boxed{a_{i1} \quad \cdots \quad a_{im}} \\ \vdots & & \vdots \\ a_{r1} & \cdots & a_{rm} \end{bmatrix}\begin{bmatrix} b_{11} & \cdots & \boxed{b_{1j}} & \cdots & b_{1c} \\ \vdots & & \vdots & & \vdots \\ b_{m1} & \cdots & b_{mj} & \cdots & b_{mc} \end{bmatrix}=\begin{bmatrix} p_{11} & \cdots & p_{1j} & \cdots & p_{1c} \\ \vdots & & \vdots & & \vdots \\ p_{i1} & \cdots & \boxed{p_{ij}} & \cdots & p_{ic} \\ \vdots & & \vdots & & \vdots \\ p_{r1} & \cdots & p_{rj} & \cdots & p_{rc} \end{bmatrix}$$

上式中，以p_{ij}为例，积的结果为

$$p_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{im}b_{mj}$$

要注意，只有当左边矩阵的列数与右边矩阵的行数相等时，两个矩阵才可以相乘。例如：

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}\begin{bmatrix} 6 & 7 & 8 & 9 \\ 0 & 1 & 2 & 3 \end{bmatrix}=\begin{bmatrix} 0 & 1 & 2 & 3 \\ 12 & 17 & 22 & 27 \\ 24 & 33 & 42 & 51 \end{bmatrix}$$

在多数情况下，矩阵乘法不满足交换律，即：

122

$$AB \neq BA \tag{5-2}$$

同样道理，不能根据$AB = AC$，就推出$B = C$。不过，矩阵乘法满足结合律和分配律：

$$(AB)C = A(BC)$$

$$A(B+C) = AB + AC$$

$$(A+B)C = AC + BC$$

在图形学中，我们通过方阵对用矩阵表示的向量进行变换。例如，对于二维向量$a=(x_a, y_a)$，将它关于y轴映射成向量$a'=(-x_a, y_a)$。这时，可用一个2×2的矩阵和一个2×1的矩阵相乘，后者经常被称为“列向量”。用矩阵表示的运算形式如下：

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_a \\ y_a \end{bmatrix} = \begin{bmatrix} -x_a \\ y_a \end{bmatrix}$$

通过左乘一个“行向量”，可以得到同样的结果：

$$\begin{bmatrix} x_a & y_a \end{bmatrix}\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -x_a & y_a \end{bmatrix}$$

现在，右乘一个列向量基本上已经成为标准。不过在早期的书籍和系统中，还会遇到很多左乘行向量的做法。不同之处在于，变换矩阵需要用它的转置形式。矩阵A的转置矩阵A^T，是将A矩阵的行列互换得到的，举例如下：

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

需要注意的是，在前面的映射例子中，矩阵的转置与原矩阵相同。称这样的矩阵为对称矩阵。两矩阵积的转置满足如下关系：

$$(AB)^T = B^T A^T$$

通过实数的倒数，我们联想到矩阵的逆。我们知道，实数x的倒数是$1/x$，而且x与它的倒数之积等于1。我们需要一个矩阵I，用它来表示“矩阵中的1”。该矩阵必须是一个方阵，而且是单位矩阵，即它的对角线元素为1，而其他元素全为0。例如，一个4×4的单位矩阵如下：

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

123

单位矩阵是对角矩阵的特例，对角矩阵是指所有非零元素都在对角线上的矩阵。对角线元素，就是从矩阵左上角开始，行索引和列索引相等的元素。满足$AA^{-1} = I$的矩阵A^{-1}，称为矩阵A的逆矩阵。例如：

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{-1} = \begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix} \text{ 因为 } \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}\begin{bmatrix} -2.0 & 1.0 \\ 1.5 & -0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

注意A^{-1}的逆矩阵是A，因此有$AA^{-1} = A^{-1}A = I$。另外要注意，两矩阵之积的逆矩阵为

$$(AB)^{-1} = B^{-1}A^{-1}$$

在本章稍后，我们将继续讨论逆矩阵。

5.2.2　矩阵形式的向量运算

当使用笛卡儿坐标时，可以通过矩阵形式来表示向量运算。1×1矩阵的点积运算，可以表示为：

$$\boldsymbol{a}\cdot\boldsymbol{b}=\boldsymbol{a}^T\boldsymbol{b}$$

如果使用两个三维向量，就会得到下式：

$$\begin{bmatrix}x_a & y_a & z_a\end{bmatrix}\begin{bmatrix}x_b\\ y_b\\ z_b\end{bmatrix}=\begin{bmatrix}x_a x_b+y_a y_b+z_a z_b\end{bmatrix}$$

5.2.3　矩阵与行列式

回忆5.1节所讲的内容，行列式包含n个n维向量，将这些向量组合起来，得到有向n维空间物体的体积，它是由这n个向量确定的。例如，二维行列式表示由两个向量所确定的平行四边形的面积。可以通过矩阵来表示行列式运算。

假设有两个二维向量$\boldsymbol{r}$和$\boldsymbol{s}$，行列式表示为$|\boldsymbol{rs}|$。行列式的值等于两向量所组成的平行四边形的有符号面积。假设用笛卡儿坐标表示的两个二维向量为（a, b）和（A, B）（见图5-7）。行列式可以写成列向量的形式，也可以进行简化：

124

$$\left|\begin{bmatrix}a\\ b\end{bmatrix}\quad\begin{bmatrix}A\\ B\end{bmatrix}\right|\equiv\begin{vmatrix}a & A\\ b & B\end{vmatrix}=aB-Ab \tag{5-3}$$

注意一个矩阵的行列式和其转置矩阵的行列式具有相等的值：

$$\begin{vmatrix}a & A\\ b & B\end{vmatrix}=\begin{vmatrix}a & b\\ A & B\end{vmatrix}=aB-Ab$$

这表示，二维空间中的任意平行四边形，都存在和它面积相等、形状不同的平行四边形（见图5-8）。例如在图5-8中，向量（3, 1）和（2, 4）所形成的平行四边形面积为10，而向量（3, 2）和（1, 4）所组成的平行四边形面积也为10。

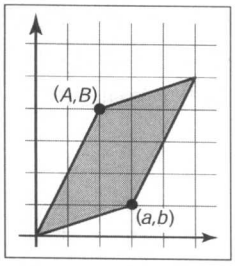


图5-7　式（5-3）中的二维行列式的值，等于由两二维向量组成的平行四边形的面积

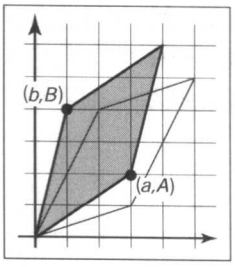


图5-8　错切的平行四边形与图5-7中的平行四边形具有相等的面积

三维行列式的几何含义有助于理解一些公式的含义。例如，通过三点（x_i, y_i, z_i），i=0, 1, 2的平面方程为

$$\begin{vmatrix}x-x_0 & x-x_1 & x-x_2\\ y-y_0 & y-y_1 & y-y_2\\ z-z_0 & z-z_1 & z-z_2\end{vmatrix}=0$$

其中的各列表示从点（x_i, y_i, z_i）到点（x, y, z）的向量。仅当（x, y, z）与其他三个点共面时，以

这三个向量为边的平行六面体的体积为零。几乎所有包含行列式的方程都有类似的几何含义。

像我们刚才看到的，对于多数元素为零的行列式，可以采用蛮力展开的方法计算行列式，还要记住很多正负标记。计算行列式的代数标准方法就是采用拉普拉斯展开形式。用这种方法计算行列式的关键，是求矩阵各元素的余子式。方阵中的每个元素都有自己的余子式，该余子式是具有更少行数和列数的矩阵行列式，带有正负号。对于原来矩阵中的某个元素，划去该元素所在的行和列，余下部分所构成的行列式就对应该元素的余子式。例如，对于10×10的矩阵来说，a_{82}的余子式，就是去掉第8行和第2列后，对应剩下的9×9的矩阵行列式。如果元素的行、列坐标索引之和为偶数，则余子式的符号为正，否则就为负。可以借助下面的棋盘模式帮助记忆余子式的符号：

$$\begin{bmatrix} + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

125

对于下面的4×4矩阵：

$$\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

第一行各元素的余子式分别为

$$a_{11}^c=\begin{vmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{vmatrix},\quad a_{12}^c=-\begin{vmatrix} a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{vmatrix},$$

$$a_{13}^c=\begin{vmatrix} a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \\ a_{41} & a_{42} & a_{44} \end{vmatrix},\quad a_{14}^c=-\begin{vmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{vmatrix}$$

将任一行或任一列的所有元素与它们的余子式分别相乘，再全部加起来，就得到了矩阵行列式的值。例如，上面提到的4×4矩阵，关于第二列的行列式展开形式如下：

$$|\boldsymbol{A}|=a_{12}a_{12}^c+a_{22}a_{22}^c+a_{32}a_{32}^c+a_{42}a_{42}^c$$

我们可以任选一行或者一列，进行与上面类似的计算，都会得到相同的结果。要注意到这种方法的递归性质。

下面是一个3×3的具体矩阵例子，按第一行进行余子式展开的结果为

$$\begin{aligned}\begin{vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{vmatrix} &= 0\begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix}-1\begin{vmatrix} 3 & 5 \\ 6 & 8 \end{vmatrix}+2\begin{vmatrix} 3 & 4 \\ 6 & 7 \end{vmatrix} \\ &= 0(32-35)-1(24-30)+2(21-24) \\ &= 0\end{aligned}$$

由此可以推断，由列向量（行向量也一样，因为转置矩阵的行列式与原行列式相等）所确定的平行六面体的体积等于零。也就是说，这些列向量（行向量）不是线性独立的。可以看出，第3行和第1行之和，正好是第2行的两倍，这就意味着它们是线性相关的。

5.2.4　计算逆矩阵

126　行列式为我们提供一种计算逆矩阵的工具。对于大型矩阵，这种方法的效率很低，不过在图形学中用到的矩阵规模一般都很小。采用这种方法的关键是，对于具有两个相同行的矩阵，其行列式值为零。这一点是显而易见的，对于n维空间物体，如果有两维基向量是相等的，则该物体的体积为零。对于4×4的矩阵A，希望求出它的逆矩阵A^{-1}。逆矩阵求法如下：

$$A^{-1}=\frac{1}{|A|}\begin{bmatrix} a_{11}^c & a_{21}^c & a_{31}^c & a_{41}^c \\ a_{12}^c & a_{22}^c & a_{32}^c & a_{42}^c \\ a_{13}^c & a_{23}^c & a_{33}^c & a_{43}^c \\ a_{14}^c & a_{24}^c & a_{34}^c & a_{44}^c \end{bmatrix}$$

在上式中，将A矩阵中的元素，用它们带符号的余子式代替，然后再对结果进行转置。最后得到的矩阵称为矩阵A的伴随矩阵。伴随矩阵就是A的余子式矩阵的转置矩阵。下面证明这样得出的矩阵为什么是逆矩阵。如果结果是逆矩阵，则AA^{-1}应为单位矩阵。将A的第一行元素乘以伴随矩阵的第一列元素，希望的结果是行列式$|A|$（注意上式中的常数系数$1/|A|$），即

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}\begin{bmatrix} a_{11}^c & \cdot & \cdot & \cdot \\ a_{12}^c & \cdot & \cdot & \cdot \\ a_{13}^c & \cdot & \cdot & \cdot \\ a_{14}^c & \cdot & \cdot & \cdot \end{bmatrix}=\begin{bmatrix} |A| & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

结果与希望的一样，因为伴随矩阵的第一列就是A的第一行元素的余子式，相乘的结果刚好等于A的行列式。类似地，结果矩阵的对角线上的其他元素值都是$|A|$。而将A的第二行元素乘以伴随矩阵的第一列元素，结果为0：

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}\begin{bmatrix} a_{11}^c & \cdot & \cdot & \cdot \\ a_{12}^c & \cdot & \cdot & \cdot \\ a_{13}^c & \cdot & \cdot & \cdot \\ a_{14}^c & \cdot & \cdot & \cdot \end{bmatrix}=\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

其中乘积表示某个矩阵行列式的值：

$$a_{21}a_{11}^c+a_{22}a_{12}^c+a_{23}a_{13}^c+a_{24}a_{14}^c$$

上式对应的矩阵为

$$\begin{bmatrix} a_{21} & a_{22} & a_{23} & a_{24} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

127　因为矩阵的头两行是相等的，所以该矩阵是奇异的，其行列式的值为零。

上面的讨论结果并不局限于4×4的矩阵，采用4×4矩阵只是为了书写方便。任何矩阵的逆矩阵，都等于原矩阵的伴随矩阵除以原矩阵的行列式。伴随矩阵就是余子式矩阵的转置矩阵，其中余子式矩阵通过将原矩阵的各元素用相应的余子式代替得到。例如，一个行列式为6的3×3矩阵的逆矩阵为

$$\begin{bmatrix}1&1&2\\1&3&4\\0&2&5\end{bmatrix}^{-1}=\frac{1}{6}\begin{bmatrix}\begin{vmatrix}3&4\\2&5\end{vmatrix}&-\begin{vmatrix}1&2\\2&5\end{vmatrix}&\begin{vmatrix}1&2\\3&4\end{vmatrix}\\-\begin{vmatrix}1&4\\0&5\end{vmatrix}&\begin{vmatrix}1&2\\0&5\end{vmatrix}&-\begin{vmatrix}1&2\\1&4\end{vmatrix}\\\begin{vmatrix}1&3\\0&2\end{vmatrix}&-\begin{vmatrix}1&1\\0&2\end{vmatrix}&\begin{vmatrix}1&1\\1&3\end{vmatrix}\end{bmatrix}$$

$$=\frac{1}{6}\begin{bmatrix}7&-1&-2\\-5&5&-2\\2&-2&2\end{bmatrix}$$

我们可以动手算算上式，看看结果是否正确。

5.2.5 线性方程组

在图形学中，经常会遇到“n个方程及n个未知数”的线性方程组，例如（n=3），

$$\begin{aligned}3x+7y+2z&=4\\2x-4y-3z&=-1\\5x+2y+z&=1\end{aligned}$$

其中x、y、z是我们要求的未知数。把上式写成矩阵形式，得到：

$$\begin{bmatrix}3&7&2\\2&-4&-3\\5&2&1\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}4\\-1\\1\end{bmatrix}$$

这种系统通常简写为$\boldsymbol{Ax}=\boldsymbol{b}$，式中$\boldsymbol{A}$是已知常数构成的方阵，$\boldsymbol{x}$是未知列向量（在本例中由$x$、$y$、$z$组成），而$\boldsymbol{b}$是已知常数构成的列矩阵。 128

求解这种线性方程组问题的方法有很多。对于小型系统，可以从几何角度出发，采用前面提过的二维克莱姆法则。这里我们先研究代数运算方法。上述方程组的解为

$$x=\frac{\begin{vmatrix}4&7&2\\-1&-4&-3\\1&2&1\end{vmatrix}}{\begin{vmatrix}3&7&2\\2&-4&-3\\5&2&1\end{vmatrix}}\quad y=\frac{\begin{vmatrix}3&4&2\\2&-1&-3\\5&1&1\end{vmatrix}}{\begin{vmatrix}3&7&2\\2&-4&-3\\5&2&1\end{vmatrix}}\quad z=\frac{\begin{vmatrix}3&7&4\\2&-4&-1\\5&2&1\end{vmatrix}}{\begin{vmatrix}3&7&2\\2&-4&-3\\5&2&1\end{vmatrix}}$$

方法是计算两个行列式之比，其中分母是$|\boldsymbol{A}|$，分子是用列向量$\boldsymbol{b}$代替$\boldsymbol{A}$中一列所形成矩阵的行列式。被替换的一列与向量$\boldsymbol{x}$中的未知数对应。例如，y是第二个未知数，那么$\boldsymbol{A}$的第二列被替换。要注意，如果$|\boldsymbol{A}|$=0，上述除法无定义。这是克莱姆法则的另一种情况：如果$\boldsymbol{A}$是奇异的（$|\boldsymbol{A}|$=0），则方程组没有唯一解。

5.2.6 特征值及矩阵对角化

方阵具有特征值和对应的特征向量。特征向量是指与矩阵相乘时方向不变的非零向量。例如，假设矩阵$\boldsymbol{A}$和向量$\boldsymbol{a}$满足下列关系：

$$\boldsymbol{Aa}=\lambda\boldsymbol{a}\qquad(5\text{-}4)$$

这意味着，我们对$\boldsymbol{a}$进行了放大或缩小，但它的方向没有变化。比例系数λ称为与特征向量$\boldsymbol{a}$对应的特征值。矩阵的特征值和特征向量，在实际应用中的作用很大。掌握矩阵的特征值和特征向量，可以帮助我们理解矩阵的几何变换，也为下一节学习奇异值和向量打好基础。

假设矩阵至少有一个特征向量，那么能够通过标准运算找到它。首先，将上式两边都写成方阵与向量的乘积形式：

$$\boldsymbol{Aa}=\lambda\boldsymbol{Ia} \tag{5-5}$$

其中$\boldsymbol{I}$表示单位矩阵。上式重写为

129

$$\boldsymbol{Aa}-\lambda\boldsymbol{Ia}=0 \tag{5-6}$$

由于矩阵乘法满足分配律，重新分配后得到

$$(\boldsymbol{A}-\lambda\boldsymbol{I})\boldsymbol{a}=0 \tag{5-7}$$

只有当矩阵（$\boldsymbol{A}-\lambda\boldsymbol{I}$）为奇异矩阵时，上式才成立，因此它的行列式等于零。除对角线上的元素外，（$\boldsymbol{A}-\lambda\boldsymbol{I}$）中的其他元素就是矩阵$\boldsymbol{A}$中的元素。例如，对于2×2矩阵，特征值满足

$$\begin{vmatrix}a_{11}-\lambda & a_{12}\\ a_{21} & a_{22}-\lambda\end{vmatrix}=\lambda^2-(a_{11}+a_{22})\lambda+(a_{11}a_{22}-a_{12}a_{21})=0 \tag{5-8}$$

这是一个二次方程，肯定有两个关于λ的解。这两个解可能相等，也可能不是实数。对$n\times n$的矩阵进行类似的处理，将会得到一个关于λ的n次多项式。一般地说，对大于四次的多项式进行精确求解是不可能的，因此如果采用解析法，我们只能求出4×4或者更小矩阵的特征值。对于更大的矩阵，只能采用数值分析法。

比较重要的一种特殊情况是，求对称矩阵的特征值（其中$\boldsymbol{A}=\boldsymbol{A}^T$）。对称矩阵的特征值都是实数。如果特征值都不相等，则它们的特征向量是互相正交的。这种矩阵可以写成对角形式：

$$\boldsymbol{A}=\boldsymbol{RDR}^T \tag{5-9}$$

其中$\boldsymbol{R}$是一个正交矩阵，$\boldsymbol{D}$是一个对角矩阵。回想一下，正交矩阵最好是标准正交的，则矩阵各列相互正交，并且每一列元素的平方和等于1。$\boldsymbol{R}$的列是$\boldsymbol{A}$的特征向量，$\boldsymbol{D}$中的非零元素是$\boldsymbol{A}$的特征值。例如，已知下列矩阵：

$$\boldsymbol{A}=\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}$$

那么$\boldsymbol{A}$的特征值是下列方程的解：

$$\lambda^2-3\lambda+1=0$$

为了书写方便，我们将结果用近似值表示：

$$\lambda=\frac{3\pm\sqrt{5}}{2},\quad\approx\begin{bmatrix}2.618\\ 0.382\end{bmatrix}$$

下面求相应的特征向量。先求出齐次方程的非平凡解（即不是$x=y=0$）：

$$\begin{bmatrix}2-2.618 & 1\\ 1 & 1-2.618\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix}=\begin{bmatrix}0\\ 0\end{bmatrix}$$

130

近似为$(x,y)=(0.8507,\ 0.5257)$。注意与二维向量平行的解有无穷多个，但我们只选择具有单位长度的解。同样可以得出与λ_2对应的特征向量$(x,y)=(-0.5257,\ 0.8507)$。也就是说，$\boldsymbol{A}$的对角形式是（由于取了近似值，因此是在一定精度内）：

$$\begin{bmatrix}2 & 1\\ 1 & 1\end{bmatrix}=\begin{bmatrix}0.8507 & -0.5257\\ 0.5257 & 0.8507\end{bmatrix}\begin{bmatrix}2.618 & 0\\ 0 & 0.382\end{bmatrix}\begin{bmatrix}0.8507 & 0.5257\\ -0.5257 & 0.8507\end{bmatrix}$$

在下一章，我们将重新讨论该矩阵作为变换矩阵的几何意义。

5.2.7 奇异值分解

在上面的内容中，我们看到任何对称矩阵都可以对角化。然而图形学中遇到的多数矩阵都不是对称的。幸运的是，可以对这些矩阵进行奇异值分解（SVD）。将矩阵M表示为

$$M = USV$$

其中U和V是正交矩阵，S是对角矩阵。S的对角元素是M的奇异值。计算SVD有标准的方法。首先，定义矩阵$A = MM^T$，假设M可以奇异值分解，则有

$$A = MM^T = (USV)(USV)^T = USVV^TSU^T = US^2U^T$$

上面的推导利用了$(BC)^T = C^TB^T$，正交矩阵的转置矩阵等于它的逆矩阵，以及对角矩阵的转置矩阵就是它本身。这种新形式的优点在于A是对称的，并且US^2U^T是A的特征值对角分解，其中S^2内包含A的特征值。因此，矩阵的奇异值，等于该矩阵与其转置矩阵乘积的特征值的平方根。下面通过具体例子来说明：

$$M = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad A = MM^T = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

前面我们看到过对矩阵M的特征值分解，应该很快得出下面的奇异值分解：

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} \sqrt{2.618} & 0 \\ 0 & \sqrt{0.382} \end{bmatrix} V$$

131

用代数的方法从中求得V：

$$V = S^{-1}U^TM$$

S的逆矩阵是一个对角矩阵，其对角元素是S对角元素的倒数。于是得出

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = U \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} V$$

$$= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix} \begin{bmatrix} 1.618 & 0 \\ 0 & 0.618 \end{bmatrix} \begin{bmatrix} 0.5257 & 0.8507 \\ -0.8507 & 0.5257 \end{bmatrix}$$

该表达式中用标准符号σ表示奇异值。注意对称矩阵的特征值和奇异值是相同的（$\sigma_i = \lambda_i$）。另外，特征值对角化和SVD是相同的。6.1.6节将进一步分析SVD的几何意义。

常见问题

- **为什么矩阵乘法是书中定义的方式，而不是逐个元素相乘？**

用逐个元素相乘来定义矩阵乘法确实是种好方法，而且也有非常好的性质，但在实际中不实用。多数矩阵都用来变换列向量，例如在三维空间中，我们有

$$b = Ma$$

其中a和b是向量，M是一个3×3的矩阵。为了表示像旋转这样的几何变换，b中的每个元素都是a中三个元素同时参与运算的结果。这就需要矩阵M按行或者按列参与运算。现有的矩阵乘法定义，使矩阵组合运算具有我们希望的下列特点：

$$M_2(M_1a) = (M_2M_1)\,a$$

这样，当对向量做二次变换时，就可以用组合矩阵$C = M_2M_1$实现。特别是当用相同的组合矩阵对

很多向量进行变换时，就更加体现了矩阵乘法的价值。总的来说，有点奇怪的矩阵乘法法则，是
132 人们为了需要而做出的定义。

- **有时候特征值和奇异值是相同的，有时候一个是另一个的平方，到底哪种对？**

如果实矩阵M是对称的，那么它的特征值和奇异值是相同的。如果M不是对称的，则矩阵$A=MM^T$是对称的，并且有实数特征值。M和M^T的奇异值是相同的，并且等于A的奇异/特征值的平方根。因此，说到平方根时，讨论的是两种不同（但有着特殊的关系）的矩阵：$A=MM^T$。

文献注记

关于行列式表示体积的很多内容参考了*A Vector Space Approach to Geometry*（Hausner, 1998）一书。Hausner关于向量分析及几何基础知识的讨论非常出色。二维克莱姆法则的几何推导出自*Practical Linear Algebra:A Geometry Toolbox*（Farin & Hansford, 2004）一书。这本书中还有其他线性代数运算的几何解释，例如高斯消元法。关于特征值和奇异值的讨论主要参考了*Linear Algebra and its Applications*（Strang, 1988）。变换矩阵的SVD示例参考了*Computer Graphics and Geometric Modeling*（Salomon, 1999）一书。

习题

1. 采用二维行列式，写出过点(x_0, y_0)和点(x_1, y_1)的二维直线的隐式方程。
2. 证明：如果矩阵的行是标准正交的，那么列也是标准正交的。
3. 证明：对角矩阵的特征值就是它的对角元素。
4. 证明：对于方阵A，AA^T是对称矩阵。
133 5. 证明：对于三个三维向量a、b、c，$|abc| = (a \times b) \cdot c$成立。
6. 证明：由边向量a、b、c构成的四面体体积是$|abc|/6$（见图5-2）。
7. 已知二维三角形的三个顶点的(x, y)坐标，证明该三角形的面积为

$$\frac{1}{2}\begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix}$$

134

第6章

矩 阵 变 换

本章介绍如何利用矩阵乘法来完成对向量的缩放、旋转和平移，以及对位置（点）、位移和表面法向量进行上述变换时，它们之间有什么不同。

另外要介绍，当用相对原点的位移向量来表示点时，如何进行点变换。把用到的图像（钟表图）看成由很多点构成，这些点代表向量的头部，向量的尾部在原点。

6.1 基本二维变换

可以通过矩阵改变二维向量的成分。例如，

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

这种变换能够以各种方式改变向量，特别是用于缩放、旋转和切变。后面将介绍更多的矩阵变换，但是基本变换就包含在上面的简单公式之中。现在沿x轴做水平移动，沿y轴做垂直移动（见图6-1）。

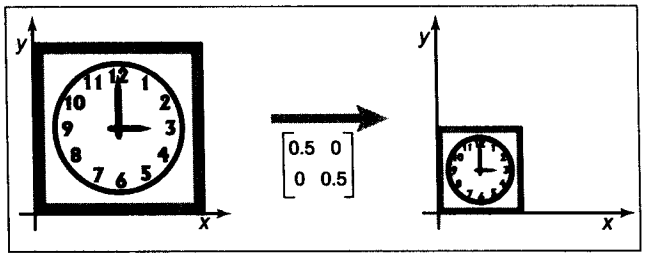


图6-1 两轴方向同时缩小一半：缩放矩阵对角线上的元素就是缩放比例，其他元素值为0

6.1.1 缩放

最基本的变换是缩放。这种变换可以改变图形的长度，也可能改变方向：

$$\text{scale}(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

对于用笛卡儿坐标(x, y)表示的向量，用上面的矩阵进行变换，结果为

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$

例如，同时将x和y缩小一半的矩阵为（见图6-1）

136

$$\text{scale}(0.5,0.5)=\begin{bmatrix}0.5 & 0\\ 0 & 0.5\end{bmatrix}$$

在水平方向减半、在垂直方向增大1.5倍的矩阵为（见图6-2）

$$\text{scale}(0.5,1.5)=\begin{bmatrix}0.5 & 0\\ 0 & 1.5\end{bmatrix}$$

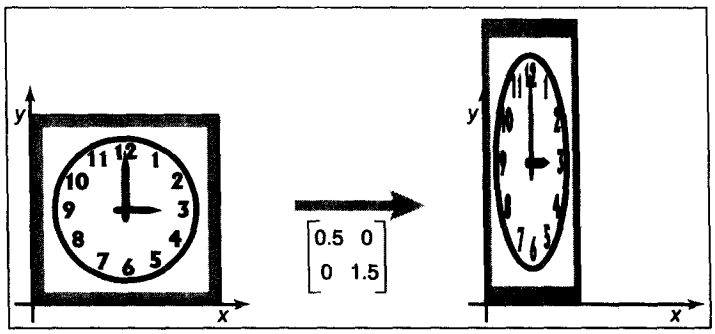


图6-2　x、y轴上的缩放比例不同。缩放矩阵是对角的，且对角线上的元素值不同。可以看到钟表外形由正方形变成了矩形，表面由圆形变成了椭圆形

6.1.2 切变

切变使图形向侧面倾斜，生成的图形就像一摞纸牌，用手横着推过去，最下面的一张保持不动，越往上纸牌位置移动得越多。水平和垂直切变矩阵分别是

$$\text{shear-x}(s)=\begin{bmatrix}1 & s\\ 0 & 1\end{bmatrix},\ \text{shear-y}(s)=\begin{bmatrix}1 & 0\\ s & 1\end{bmatrix}$$

例如，水平切变使竖直线向右倾斜45°（见图6-3）。变换公式为

$$\text{shear-x}(1)=\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$$

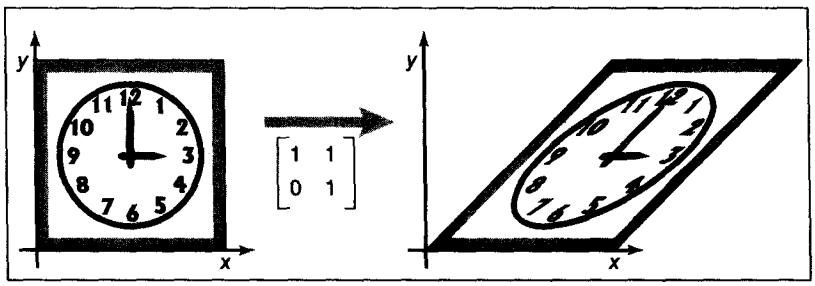


图6-3　x方向的切变矩阵，使图形中的各点向右移动一个y坐标的距离。结果钟表外形由正方形变成了平行四边形，表面变化类似缩放情况，即由圆形变成了椭圆形

类似地，垂直切变的变换公式为（见图6-4）

$$\text{shear-y}(1)=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}$$

我们注意到，切变后钟表外形由正方形变成了平行四边形。而且，表面也由圆形变得像个椭圆。

137　实际上就是个椭圆，在讨论奇异值分解时将会证实这一点。

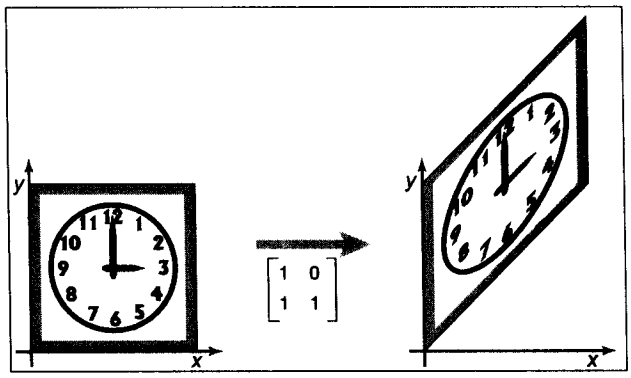


图6-4 *y*方向的切变矩阵，使图形中的各点向上移动一个*x*坐标的距离

对切变的另一种理解是，相对垂直轴（或水平轴）做旋转。相对垂直轴的切变，假设顺时针倾斜*y*轴ϕ角，则切变矩阵为

$$\begin{bmatrix} 1 & \tan\phi \\ 0 & 1 \end{bmatrix}$$

类似地，相对水平轴的切变，假设逆时针倾斜*x*轴ϕ角，则切变矩阵为

$$\begin{bmatrix} 1 & 0 \\ \tan\phi & 1 \end{bmatrix}$$

6.1.3 旋转

假设我们想把一个向量逆时针旋转ϕ角。首先假设向量$\boldsymbol{a}=(x_a, y_a)$，把它旋转$\phi$角之后，得到的向量为$\boldsymbol{b}=(x_b, y_b)$。如果向量$\boldsymbol{a}$与*x*轴的夹角为$\alpha$，向量$\boldsymbol{a}$的长度为$r=\sqrt{x_a^2+y_a^2}$，那么通过定义我们知道

$$x_a = r\cos\alpha$$
$$y_a = r\sin\alpha$$

因为向量$\boldsymbol{b}$由向量$\boldsymbol{a}$旋转而来，则$\boldsymbol{b}$的长度也为*r*。因为$\boldsymbol{b}$由$\boldsymbol{a}$旋转ϕ角得到，所以$\boldsymbol{b}$偏离*x*轴的角度是（$\alpha+\phi$）（见图6-5）。由基本三角公式我们知道 138

$$\begin{aligned} x_b &= r\cos(\alpha+\phi) = r\cos\alpha\cos\phi - r\sin\alpha\sin\phi \\ y_b &= r\sin(\alpha+\phi) = r\sin\alpha\cos\phi + r\cos\alpha\sin\phi \end{aligned} \tag{6-1}$$

将$x_a = r\cos\alpha$和$y_a = r\sin\alpha$代入上面的公式，得到

$$x_b = x_a\cos\phi - y_a\sin\phi$$
$$y_b = y_a\cos\phi + x_a\sin\phi$$

则从$\boldsymbol{a}$到$\boldsymbol{b}$的变换用矩阵形式表示为

$$\text{rotate}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

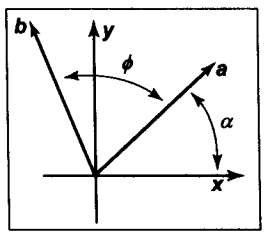


图6-5 式（6-1）的几何解释

例如，把向量旋转π/4弧度（45°）的矩阵为（见图6-6）

$$\begin{bmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}$$

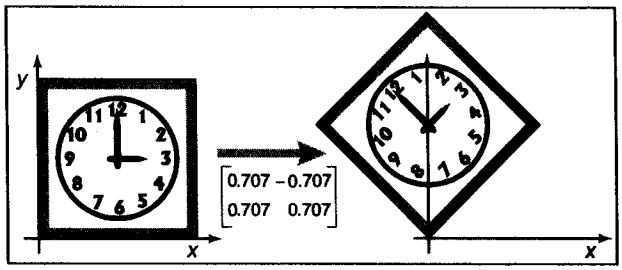


图6-6　旋转45°。注意是逆时针旋转，并且有cos(45°)=sin(45°)≈0.707

沿顺时针方向旋转π/6弧度（30°），在我们的坐标系中就是旋转-π/6弧度，旋转矩阵为（见图6-7）：

$$\begin{bmatrix} \cos\frac{-\pi}{6} & -\sin\frac{-\pi}{6} \\ \sin\frac{-\pi}{6} & \cos\frac{-\pi}{6} \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix}$$

旋转矩阵每行的范数等于1（$\sin^2\phi+\cos^2\phi=1$），并且行与行之间是正交的（$\cos\phi(-\sin\phi)+\sin\phi\cos\phi=0$），
139 因此旋转矩阵是正交矩阵（即每行长度为1，行与行正交）。

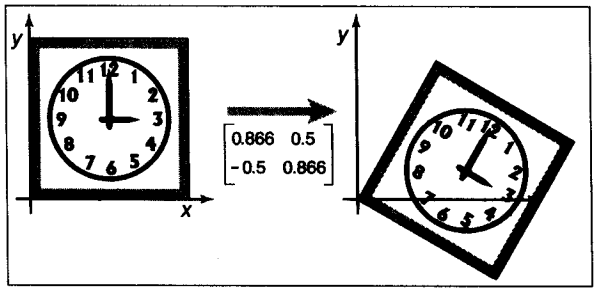


图6-7　旋转-30°。注意是顺时针旋转，并且有cos(−30°) ≈ 0.866, sin(−30°) = −0.5

6.1.4 反射

相对任意坐标轴可以反射一个向量（见图6-8及图6-9），反射矩阵为

$$\text{reflect-y}(s)=\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix},\quad \text{reflect-x}(s)=\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

140 有人会猜测，对角线上两元素都为-1的矩阵也是个反射矩阵，实际上是π弧度的旋转矩阵。

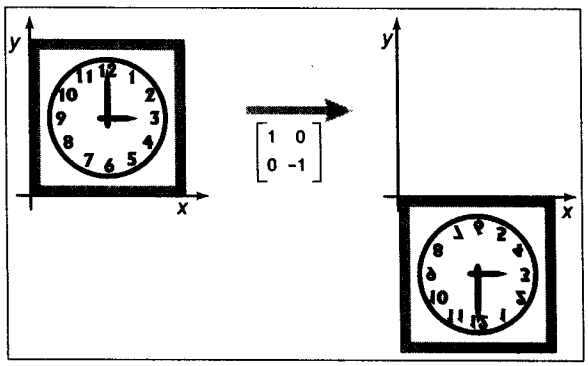


图6-8　对所有y坐标乘以-1得到关于x轴的反射

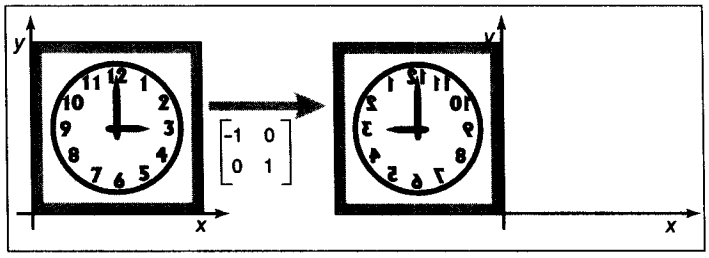


图6-9 对所有x坐标乘以-1得到关于y轴的反射

6.1.5 二维变换组合

图形学程序中，对物体的变换常常不只一种。例如，先用$\boldsymbol{S}$缩放，然后再用$\boldsymbol{R}$进行旋转。对于二维向量$\boldsymbol{v}_1$，将用两步实现：

$$\text{首先}\boldsymbol{v}_2=\boldsymbol{S}\boldsymbol{v}_1\text{，然后}\boldsymbol{v}_3=\boldsymbol{R}\boldsymbol{v}_2$$

也可用另外一种方式表示：

$$\boldsymbol{v}_3=\boldsymbol{R}(\boldsymbol{S}\boldsymbol{v}_1) \tag{141}$$

因为矩阵乘法满足结合率，因此也可以这样写：

$$\boldsymbol{v}_3=(\boldsymbol{R}\boldsymbol{S})\boldsymbol{v}_1$$

换句话说，两个矩阵的变换效果可以用一个矩阵实现，该矩阵等于原来两个矩阵的积，即$\boldsymbol{M}=\boldsymbol{R}\boldsymbol{S}$（见图6-10）。

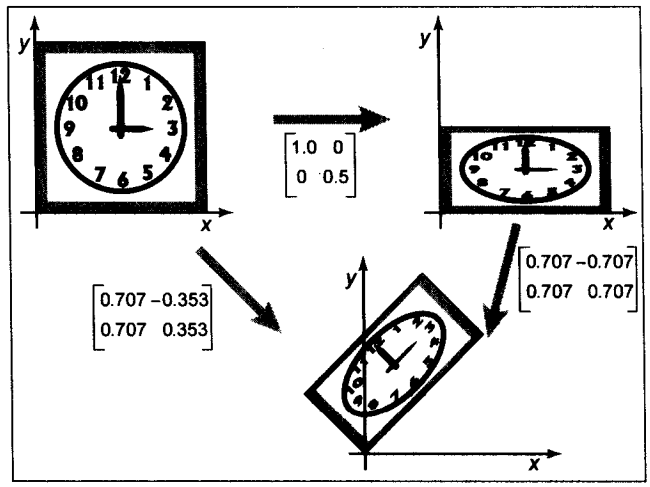


图6-10 按顺序使用两个变换矩阵，效果与一次使用两矩阵之积一样。这是多数图形硬件和软件中的核心概念

一定要记住，变换的顺序是从右向左。在矩阵$\boldsymbol{M}=\boldsymbol{R}\boldsymbol{S}$中，变换顺序是先$\boldsymbol{S}$，后$\boldsymbol{R}$。

举个例子，假设把一个图形先在垂直方向上压缩1/2，然后再旋转$\pi/4$弧度（45°）。结果得到的矩阵为 [142]

$$\begin{bmatrix}0.707 & -0.707\\ 0.707 & 0.707\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 0.5\end{bmatrix}=\begin{bmatrix}0.707 & -0.353\\ 0.707 & 0.353\end{bmatrix}$$

一定要记住，矩阵相乘是不满足交换律的，因此变换顺序很重要。例如，先缩放再旋转与先旋转

再缩放的结果一般是不同的（见图6-11）。

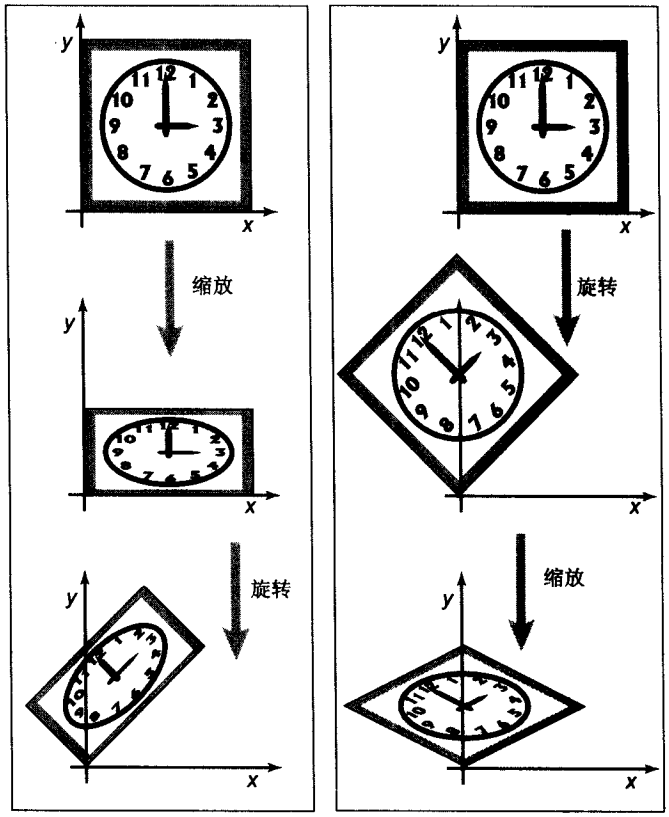


图6-11　进行变换的顺序很重要。在本例的左边，把图形先在y方向压缩1/2，然后旋转45°。右边的变换顺序与左边相反，得到了不同的变换结果

6.1.6　二维变换分解

对于任意的变换矩阵$\boldsymbol{M}$，可将它分解成各种矩阵积的形式，如$\boldsymbol{M}=\boldsymbol{M}_1\boldsymbol{M}_2$，$\boldsymbol{M}=\boldsymbol{M}_3\boldsymbol{M}_4\boldsymbol{M}_5$，等等。

143

有趣的结果是，任何二维变换矩阵都可以分解成旋转、缩放、旋转三个矩阵之积，其中缩放矩阵中的元素可能含有负号（也就是可能包含反射）。在5.2.7节的奇异值分解中，我们已经见过这种形式。如前所述，矩阵$\boldsymbol{M}$可以用乘积形式表示，即$\boldsymbol{M}=\boldsymbol{R}_2\boldsymbol{S}\boldsymbol{R}_1$，其中$\boldsymbol{R}_1$、$\boldsymbol{R}_2$都是旋转矩阵，$\boldsymbol{S}$是缩放矩阵。5.2.7节中的例子实际上就是一个切变矩阵（见图6-12）：

$$\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}=\boldsymbol{R}_2\begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}\boldsymbol{R}_1$$

$$=\begin{bmatrix}0.8507 & -0.5257\\ 0.5257 & 0.8507\end{bmatrix}\begin{bmatrix}1.618 & 0\\ 0 & 0.618\end{bmatrix}\begin{bmatrix}0.5257 & 0.8507\\ -0.8507 & 0.5257\end{bmatrix}$$

$$=\text{rotate}(31.7°)\ \text{scale}(1.618,0.618)\ \text{rotate}(-58.3°)$$

奇异矩阵分解所产生的直接结果是，所有的二维基本变换矩阵都可以表示成旋转矩阵和缩放矩阵的乘积（可以有负值元素）。切变矩阵使用起来很方便，但在表达位势变换时用不到。

对一个对称矩阵，我们可以进行特征值分解。回想5.2.6节中的例子：

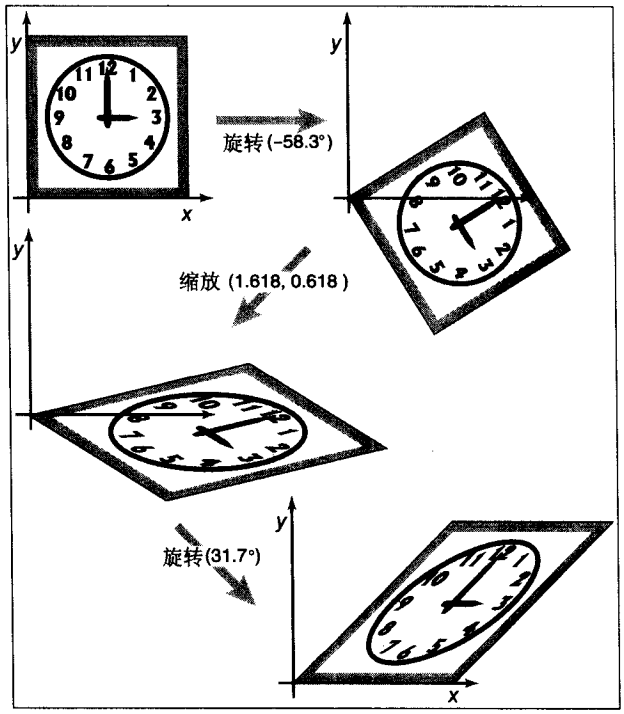


图6-12 切变矩阵的奇异值分解。任何二维矩阵都可以分解成旋转、缩放、旋转三个矩阵的乘积。可以看到，时钟表面的形状必然会变成椭圆形，因为对圆只进行了旋转和缩放变换

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \boldsymbol{R}\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}\boldsymbol{R}^T$$

$$= \begin{bmatrix} 0.8507 & -0.5257 \\ 0.5257 & 0.8507 \end{bmatrix}\begin{bmatrix} 2.618 & 0 \\ 0 & 0.382 \end{bmatrix}\begin{bmatrix} 0.8507 & 0.5257 \\ -0.5257 & 0.8507 \end{bmatrix}$$

$$= \text{rotate}(31.7°)\ \text{scale}(2.618, 0.382)\ \text{rotate}(-31.7°)$$

可以看到，这正是我们做奇异值分解时所得到的结果。因此，对于对称矩阵来说，奇异值分解和特征值分解是一样的。而且只有对称矩阵具有对角形式（$\boldsymbol{RSR}^T$），所以奇异值分解是更一般的工具。通过对角化的对称矩阵的变换，其几何解释就是：

（1）将图形相对x轴转过一个角度（$\boldsymbol{R}^T$为变换矩阵）。

（2）将图形沿x、y轴缩放（λ_1，λ_2）（$\boldsymbol{S}$为变换矩阵）。

（3）将图形沿与原来转向相反的方向转过相同的角度（$\boldsymbol{R}$为变换矩阵）。 144

这三次变换的结果，就是沿任意方向的缩放变换（见图6-13）。例如，根据特征值分解结果，上面的矩阵就表示在31.7°方向上的缩放变换，这个角度是逆时针偏离钟表3点（x轴）的角度。这是钟表上显示的下午两点前的方向，也可以从图中看出来。可以将对角化过程颠倒顺序，并按照角度ϕ项进行整理得

$$\begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1\cos^2\phi + \lambda_2\sin^2\phi & (\lambda_2 - \lambda_1)\cos\phi\sin\phi \\ (\lambda_2 - \lambda_1)\cos\phi\sin\phi & \lambda_2\cos^2\phi + \lambda_1\sin^2\phi \end{bmatrix}$$

可以肯定这是一个对称矩阵，因为在构造之前已经假设它能够进行特征值对角化。

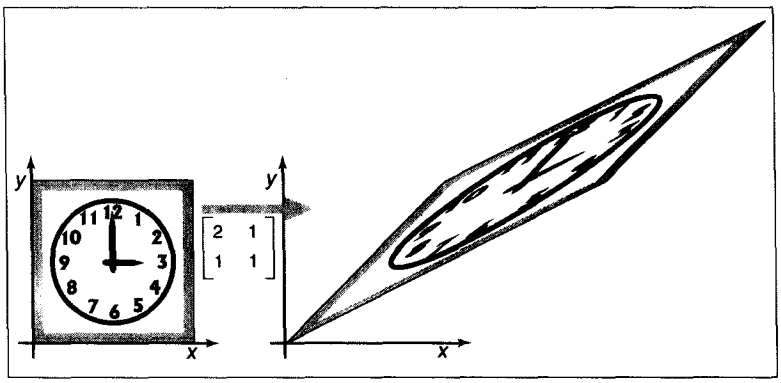


图6-13　对称矩阵总是沿着某个轴的缩放变换。在本例中，是沿ϕ=31.7°方向的缩放，也就是该矩阵的特征向量的方向

总之，通过奇异值分解，每个矩阵都可以分解成旋转、缩放、再旋转三个矩阵的乘积形式。只有对称矩阵可以通过特征值对角化，分解成旋转、缩放、逆旋转三个矩阵的乘积形式，而且这些对称矩阵都是某任意方向的缩放变换。经过稍微复杂的代数运算，对称矩阵的奇异值分解将与特征值分解的结果相同。

另一种分解方法是，用切变来表示非零旋转（Paeth,1990）。下面的等式就是这种分解结果：

$$\begin{bmatrix}\cos\phi & -\sin\phi\\ \sin\phi & \cos\phi\end{bmatrix}=\begin{bmatrix}1 & \dfrac{\cos\phi-1}{\sin\phi}\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \sin\phi & 1\end{bmatrix}\begin{bmatrix}1 & \dfrac{\cos\phi-1}{\sin\phi}\\ 0 & 1\end{bmatrix}$$

145

例如，旋转π/4弧度（45°）的矩阵可以表示为（见图6-14）

$$\text{rotate}(\frac{\pi}{4})=\begin{bmatrix}1 & 1-\sqrt{2}\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \dfrac{\sqrt{2}}{2} & 1\end{bmatrix}\begin{bmatrix}1 & 1-\sqrt{2}\\ 0 & 1\end{bmatrix} \tag{6-2}$$

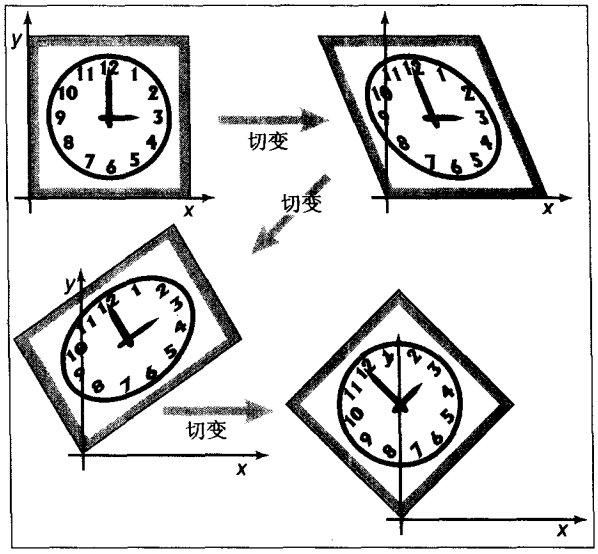


图6-14　任何二维旋转都可以通过三个一定顺序的切变实现。本例中的45°旋转，可以分解成式（6-2）中的结果

这种特殊变换在表示光栅旋转时很有用，因为切变是有效的图像光栅运算。运算结果会产生很多锯齿，但不会产生空洞。还要注意到，对于光栅点（i, j），进行水平切变，可以得到

$$\begin{bmatrix}1 & s\\ 0 & 1\end{bmatrix}\begin{bmatrix}i\\ j\end{bmatrix}=\begin{bmatrix}i+sj\\ j\end{bmatrix}$$

如果把sj进行四舍五入变成最接近的整数，图形的每行都加上某个值，一直到边缘处。不同行所加的这个值不相同。因为同一行内产生同样的位移，因此结果图像表现为无间隙的旋转。对垂直切变结果也一样。这样，就轻松地实现了简单的光栅旋转。 146

6.2 基本三维变换

基本三维变换是对二维变换的扩展。例如，沿笛卡儿坐标轴的缩放：

$$\text{scale}(s_x,s_y,s_z)=\begin{bmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & s_z\end{bmatrix} \tag{6-3}$$

旋转变换要复杂一些，因为旋转轴更多了。然而，如果只是绕z轴旋转，也就是只有x、y坐标发生变化，则可以不改变z的二维变换矩阵：

$$\text{rotate-z}(\phi)=\begin{bmatrix}\cos\phi & -\sin\phi & 0\\ \sin\phi & \cos\phi & 0\\ 0 & 0 & 1\end{bmatrix}$$

同样可以得到绕x、y轴的旋转矩阵：

$$\text{rotate-x}(\phi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix},$$

$$\text{rotate-y}(\phi)=\begin{bmatrix}\cos\phi & 0 & \sin\phi\\ 0 & 1 & 0\\ -\sin\phi & 0 & \cos\phi\end{bmatrix}$$

下一部分将讨论绕任意轴的旋转。

与二维情况一样，可以沿某个特殊的轴进行切变，例如：

$$\text{shear-x}(d_y,d_z)=\begin{bmatrix}1 & d_y & d_z\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

与二维变换一样，任意三维变换矩阵可通过SVD分解成旋转、缩放、再旋转三个矩阵的乘积。任意对称三维矩阵可通过特征值分解方法分解成旋转、缩放、逆旋转三个矩阵的乘积。最后一点，三维旋转矩阵可以分解成三维切变矩阵的乘积。

6.2.1 任意三维旋转

与二维情况一样，三维旋转矩阵都是正交矩阵。在几何上的含义是，矩阵的每行都是笛卡儿坐标上相互正交的单位向量，就像在2.4.5节中所讨论的一样。矩阵的每一列也是互不相同。互相正交的单位向量。这种旋转矩阵有无穷多个。对于下面的矩阵： 147

$$\boldsymbol{R}_{uvw}=\begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}$$

其中，$\boldsymbol{u}=x_u\boldsymbol{x}+y_u\boldsymbol{y}+z_u\boldsymbol{z}$，$\boldsymbol{v}$和$\boldsymbol{w}$也有类似的表达式。由于这三个向量是正交的，因此有

$$\boldsymbol{u}\cdot\boldsymbol{u}=\boldsymbol{v}\cdot\boldsymbol{v}=\boldsymbol{w}\cdot\boldsymbol{w}=1$$

$$\boldsymbol{u}\cdot\boldsymbol{v}=\boldsymbol{v}\cdot\boldsymbol{w}=\boldsymbol{w}\cdot\boldsymbol{u}=0$$

利用向量$\boldsymbol{u}$、$\boldsymbol{v}$、$\boldsymbol{w}$，可以推出旋转矩阵的一些性质，例如：

$$\boldsymbol{R}_{uvw}\boldsymbol{u}=\begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}\begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix}=\begin{bmatrix} x_ux_u+y_uy_u+z_uz_u \\ x_vx_u+y_vy_u+z_vz_u \\ x_wx_u+y_wy_u+z_wz_u \end{bmatrix}$$

注意$\boldsymbol{R}_{uvw}\boldsymbol{u}$的三个行向量都是点积：

$$\boldsymbol{R}_{uvw}\boldsymbol{u}=\begin{bmatrix} \boldsymbol{u}\cdot\boldsymbol{u} \\ \boldsymbol{v}\cdot\boldsymbol{u} \\ \boldsymbol{w}\cdot\boldsymbol{u} \end{bmatrix}=\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}=\boldsymbol{x}$$

同样有$\boldsymbol{R}_{uvw}\boldsymbol{v}=\boldsymbol{y}$, $\boldsymbol{R}_{uvw}\boldsymbol{w}=\boldsymbol{z}$。所以，通过$\boldsymbol{R}_{uvw}$的旋转变换把$\boldsymbol{uvw}$映射到相应的笛卡儿轴上。

如果$\boldsymbol{R}_{uvw}$是具有正交行向量的旋转矩阵，那么$\boldsymbol{R}_{uvw}^T$则是具有正交列向量的旋转矩阵，实际上它也是$\boldsymbol{R}_{uvw}$的逆矩阵（正交矩阵的逆矩阵就是它的转置矩阵）。矩阵转换的重要特点是：代数的逆运算也就是几何上的逆运算。所以，如果$\boldsymbol{R}_{uvw}$是从$\boldsymbol{u}$到$\boldsymbol{x}$的变换，则$\boldsymbol{R}_{uvw}^T$是从$\boldsymbol{x}$到$\boldsymbol{u}$的变换。这个结论对向量$\boldsymbol{v}$和$\boldsymbol{y}$也是一样的：

$$\boldsymbol{R}_{uvw}^T\boldsymbol{y}=\begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}=\begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix}=\boldsymbol{v}$$

因此，总是能够根据正交基来构造旋转矩阵。

如果想绕任意向量$\boldsymbol{a}$进行旋转，可以先构造一个正交基$\boldsymbol{w}=\boldsymbol{a}$，把这个正交基旋转到标准基$\boldsymbol{xyz}$，然后绕$z$轴旋转，再把标准基旋转回到$\boldsymbol{uvw}$基。绕$w$轴旋转角度$\phi$的矩阵表达式为：

$$\begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix}\begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}$$

148

其中，$\boldsymbol{w}$是方向$\boldsymbol{a}$上的单位向量（向量$\boldsymbol{a}$除以自身长度）。但是$\boldsymbol{u}$和$\boldsymbol{v}$怎么得到呢？求$\boldsymbol{u}$和$\boldsymbol{v}$的方法，我们在2.4.6节中已经给出。

注意一点，已知一个旋转矩阵，我们希望知道它的旋转方向。方法是先算出它的一个实特征值（将是$\lambda=1$），对应的特征向量就是旋转的“极轴”，也就是旋转时不发生变化的轴。

6.2.2 法向量变换

多数三维向量用来表示位置（到原点的位移向量）或方向，例如表示光来自哪里，也可以用向量表示*表面法线*。表面法向量垂直于表面的切平面。当对表面进行变换时，对法线的变换方式与我们想象的不一样。例如，用矩阵$\boldsymbol{M}$对表面上的点做变换，那么$\boldsymbol{M}$乘以表面的切向量$\boldsymbol{t}$的结果，将与变换后的表面相切。而经过$\boldsymbol{M}$变换后的法向量$\boldsymbol{n}$可能不再与变换后的表面垂直（见图6-15）。

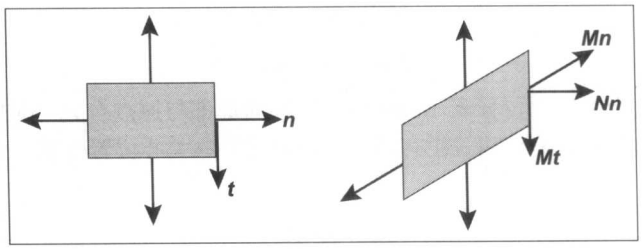


图6-15 对法向量的变换，采用的矩阵与对物体上点的变换相同。图中发生形变的矩形表明，变换后的法向量可能不再与表面垂直。但变换后的切向量仍然与表面相切

可以求出一个变换矩阵$\boldsymbol{N}$，使得向量$\boldsymbol{n}$与变换后的曲面垂直。解决该问题一种方法是，利用表面法向量与切向量垂直这一事实，则它们的点积等于0，用矩阵形式表示为

$$\boldsymbol{n}^T\boldsymbol{t}=\boldsymbol{0} \tag{6-4}$$

如果把希望得到的向量表示为$\boldsymbol{t}_M=\boldsymbol{Mt}, \boldsymbol{n}_N=\boldsymbol{Nn}$，那么我们的目标就是要找到这样的$\boldsymbol{N}$，使得$\boldsymbol{n}_N^T\boldsymbol{t}_M=\boldsymbol{0}$成立。可以通过代数运算求出$\boldsymbol{N}$。首先，把单位矩阵插入点积中，然后利用$\boldsymbol{M}^{-1}\boldsymbol{M}=\boldsymbol{I}$，得到 149

$$\boldsymbol{n}^T\boldsymbol{t}=\boldsymbol{n}^T\boldsymbol{It}=\boldsymbol{n}^T\boldsymbol{M}^{-1}\boldsymbol{Mt}=\boldsymbol{0}$$

尽管对上式的变化还看不出什么，但是加括号后就明显看出了点积形式：

$$(\boldsymbol{n}^T\boldsymbol{M}^{-1})(\boldsymbol{Mt})=(\boldsymbol{n}^T\boldsymbol{M}^{-1})\boldsymbol{t}_M=\boldsymbol{0}$$

等式左边表示行向量与$\boldsymbol{t}_M$垂直。该式对于切平面上的任意切向量都成立。在三维中只有一个方向（或者反方向）与所有的切向量垂直，因此上式左边括号中的内容必然是$\boldsymbol{n}_N$的行向量，也就是$\boldsymbol{n}_N^T$，由此可以推出$\boldsymbol{N}$：

$$\boldsymbol{n}_N^T=\boldsymbol{n}^T\boldsymbol{M}^{-1}$$

再进行转置，得到

$$\boldsymbol{n}_N=(\boldsymbol{n}^T\boldsymbol{M}^{-1})^T=(\boldsymbol{M}^{-1})^T\boldsymbol{n} \tag{6-5}$$

这样就得到了可以正确变换法向量的矩阵，即$\boldsymbol{N}=(\boldsymbol{M}^{-1})^T$，也就是$\boldsymbol{M}$逆矩阵的转置矩阵。该矩阵有可能改变向量$\boldsymbol{n}$的长度。可以对向量$\boldsymbol{n}$乘上标量系数，使其变为$\boldsymbol{n}_N$，但方向不变。回忆5.2.3节的内容，矩阵的逆矩阵等于余子式矩阵的转置除以行列式。因为我们不关心法向量的长度，可以忽略分母得到3×3的矩阵：

$$\boldsymbol{N}=\begin{bmatrix} m_{11}^c & m_{12}^c & m_{13}^c \\ m_{21}^c & m_{22}^c & m_{23}^c \\ m_{31}^c & m_{32}^c & m_{33}^c \end{bmatrix}$$

假设$\boldsymbol{M}$中第i行第j列元素是m_{ij}，则$\boldsymbol{N}$的表达式为

$$\boldsymbol{N}=\begin{bmatrix} m_{22}m_{33}-m_{23}m_{32} & m_{23}m_{31}-m_{21}m_{33} & m_{21}m_{32}-m_{22}m_{31} \\ m_{13}m_{32}-m_{12}m_{33} & m_{11}m_{33}-m_{13}m_{31} & m_{12}m_{31}-m_{11}m_{32} \\ m_{12}m_{23}-m_{13}m_{22} & m_{13}m_{21}-m_{11}m_{23} & m_{11}m_{22}-m_{12}m_{21} \end{bmatrix}$$

150

6.3 平移

我们一直在研究用矩阵$\boldsymbol{M}$来改变向量的方法。二维情况下的变换具有如下形式：

$$x' = m_{11}x + m_{12}y$$
$$y' = m_{21}x + m_{22}y$$

我们不能用这种变换来改变用位移表示的位置，位移向量都与原点有关。回想一下，如果方向及位移向量没有原点，那么谈论移动就没有意义。我们这里只关心位置变化。要改变点的位置，需要如下变换形式：

$$x' = x + x_t$$
$$y' = y + y_t$$

这时就不能用2×2矩阵与（x, y）相乘的形式。可以通过矩阵实现缩放和旋转变换，然后单独进行平移变换，但这个运算过程比较复杂。解决办法是，将计算变换到更高维中。这种方法已经成为大部分图形程序特别是图形硬件芯片的标准。

关键是要看到，基于z轴的三维切变具有下列形式：

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + x_t z \\ y + y_t z \\ z \end{bmatrix}$$

上式几乎与xy坐标系中的二维变换形式一样，只不过多出一项z，在二维情况下z没有意义。下面是关键的一步：在所有的二维点表示中增加一个坐标z=1，于是得到

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_t \\ y + y_t \\ 1 \end{bmatrix}$$

通过把z=1坐标添加到所有的二维点中，就可以把上面的变换写成矩阵形式。举例来说，二维情况下先平移（t_x, t_y），然后旋转ϕ角度，就可以得到

$$\boldsymbol{M} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}$$

151

现在二维旋转矩阵变成了3×3的矩阵，其中对应“平移”的元素为0。采用这种形式，也就是用沿z=1轴的切变表示平移，就可以用一个组合三维矩阵来表示任意多的二维切变、二维旋转、二维平移。有趣的是，三维矩阵的最后一行元素往往是（0，0，1），因此不用真正保存它们，当两个矩阵相乘时只需记住它们在那儿就行。

新形式带来一个新问题。当我们只做平移，并不想改变方向或者位移向量时，怎么办？这时，可以把第三个坐标元素设为0，也就是：

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

这就是我们希望向量应有的特性。所以，二维中的第三个坐标要么是1，要么是0，取决于改变位置还是改变方向。我们往往称这种坐标为齐次坐标（Roberts, 1965; Riesenfeld, 1981; Penna & Patterson, 1986）。实际上不用保存齐次坐标，因此可以将位置和向量区分开来。例如：

$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$是一个位置，$\begin{bmatrix} 3 \\ 2 \\ 0 \end{bmatrix}$是一个位移或方向

在后面的透视变换中可以看到，齐次坐标采用0和1之外的其他值用处还是很大的。

上面的表示方法可以类推到三维情况。我们增加第4个坐标轴，也就是齐次坐标，然后再进行平移：

$$\begin{bmatrix}1&0&0&x_t\\0&1&0&y_t\\0&0&1&z_t\\0&0&0&1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}x+x_t\\y+y_t\\z+z_t\\1\end{bmatrix}$$

同样，向量的第4个坐标是0，这样向量就不受平移的影响。

有意思的是，如果先用包含切变与旋转的任意矩阵进行变换，然后进行平移（平移在后），相乘的结果是

$$\begin{bmatrix}1&0&0&x_t\\0&1&0&y_t\\0&0&1&z_t\\0&0&0&1\end{bmatrix}\begin{bmatrix}a_{11}&a_{12}&a_{13}&0\\a_{21}&a_{22}&a_{23}&0\\a_{31}&a_{32}&a_{33}&0\\0&0&0&1\end{bmatrix}=\begin{bmatrix}a_{11}&a_{12}&a_{13}&x_t\\a_{21}&a_{22}&a_{23}&y_t\\a_{31}&a_{32}&a_{33}&z_t\\0&0&0&1\end{bmatrix}$$

152

于是，任意矩阵都可以看成是由缩放/旋转和平移两部分组成，从上述结果可以很清楚地看出来。

很重要的一类变换是刚体变换。刚体变换只包含平移变换和旋转变换，所以没有对目标的拉伸或收缩。这种变换只是对上面的a_{ij}进行旋转操作。

窗口变换

在图形学中，往往需要建立一个变换矩阵，它将矩形$[a, A]\times[b, B]$中的点变换到矩形$[c, C]\times[d, D]$中。这可以通过缩放和平移两次变换实现。但是采用三次变换会更直观一些（见图6-16）：

（1）把点（a, b）移动到原点。

（2）缩放矩形，使它与目标矩形大小一样。

（3）把原点移动到点（c, d）。

153

要记住先用右边的变换矩阵。这个过程可以写为

$$\text{window} = \text{translate}(c,\text{d})\ \ \text{scale}\left(\frac{C-c}{A-a},\frac{D-\text{d}}{B-b}\right)\ \ \text{translate}(-a,-b)$$

$$=\begin{bmatrix}1&0&c\\0&1&\text{d}\\0&0&1\end{bmatrix}\begin{bmatrix}\frac{C-c}{A-a}&0&0\\0&\frac{D-\text{d}}{B-b}&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-a\\0&1&-b\\0&0&1\end{bmatrix}\qquad(6\text{-}6)$$

$$=\begin{bmatrix}\frac{C-c}{A-a}&0&\frac{cA-Ca}{A-a}\\0&\frac{D-\text{d}}{B-b}&\frac{\text{d}B-Db}{B-b}\\0&0&1\end{bmatrix}$$

得到这种形式的结果，读者不会感到奇怪；而对于这三个矩阵的构造过程，也应该不会有问题。

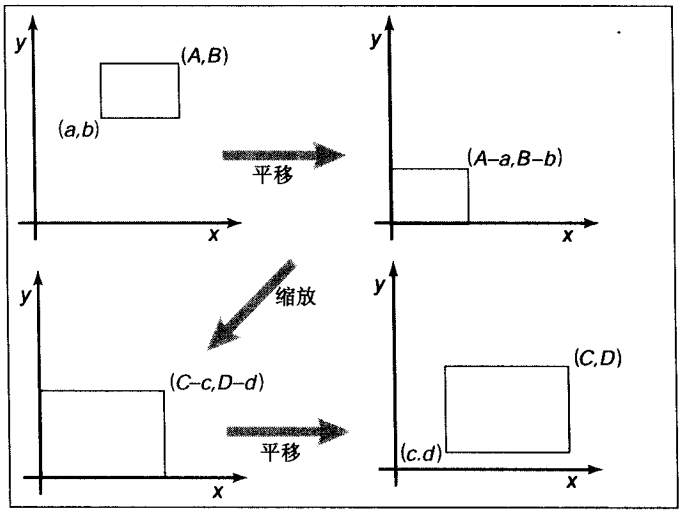


图6-16　变换矩形窗口。先把矩形的左下角移动到原点，然后通过缩放变换到新的大小，再把原点移动到目标矩形的左下角

6.4　变换矩阵的逆

我们总是用代数方法求逆矩阵，实际上如果知道要进行什么变换，可以利用几何方法求逆矩阵。例如，缩放矩阵（s_x,s_y,s_z）的逆是缩放矩阵（$1/s_x,1/s_y,1/s_z$）。旋转矩阵的逆就是它的转置矩阵。平移矩阵的逆就是反方向的平移矩阵。对于矩阵$\boldsymbol{M}=\boldsymbol{M}_1\boldsymbol{M}_2\cdots\boldsymbol{M}_n$，其逆矩阵为$\boldsymbol{M}^{-1}=\boldsymbol{M}_n^{-1}\cdots\boldsymbol{M}_2^{-1}\ \boldsymbol{M}_1^{-1}$。

有意思的是，我们也可以用SVD来求逆矩阵。我们知道任何矩阵都可以分解为旋转、缩放、旋转三个矩阵相乘的形式，那么求逆矩阵直接把他们顺序倒过来就行了。例如对三维情况，我们有

$$\boldsymbol{M}=\boldsymbol{R}_1\text{scale}(\sigma_1,\sigma_2,\sigma_3)\boldsymbol{R}_2$$

根据上面的法则，我们可以很容易地写出

$$\boldsymbol{M}^{-1}=\boldsymbol{R}_2^T\text{scale}(1/\sigma_1,1/\sigma_2,1/\sigma_3)\boldsymbol{R}_1^T$$

6.5　坐标变换

前面讨论的都是通过矩阵变换来改变坐标点的位置。也可以认为是点所在的坐标系发生变
154 换。例如在图6-17中，我们用两种方法来表示移动。根据不同的应用背景，无论选用哪种方法都可以。举例来说，假设有一个关于城市和轿车的模型，我们可以用适当的方式来表示城市和轿车的相对坐标的变化。这可以通过改变城市或轿车的坐标系来完成。直观的方式是移动轿车的坐标系，但是也可以改变城市的坐标系。具体选择哪一种方式并不重要，关键是思路要前后一致，并提供文档资料说明。

经常要用到多个笛卡儿坐标系，其中每个坐标系都有自己的基向量和原点。一般地，有一个“全局”或者“标准”坐标系。在二维空间中，标准坐标系通常用$\boldsymbol{o}$表示原点，而用$\boldsymbol{x}$和$\boldsymbol{y}$作为右手标准正交基向量（见图6-18）。其他坐标系，如用$\boldsymbol{e}$表示原点，用$\boldsymbol{u}$和$\boldsymbol{v}$表示右手标准正交基向量。
155 一定要注意，标准坐标系的$\boldsymbol{o}$、$\boldsymbol{x}$和$\boldsymbol{y}$从不显式存储，它们是所有其他坐标系的参考系。在标准坐标系中，经常将$\boldsymbol{p}$的位置用有序对表示，这是对向量表示的简便记法：

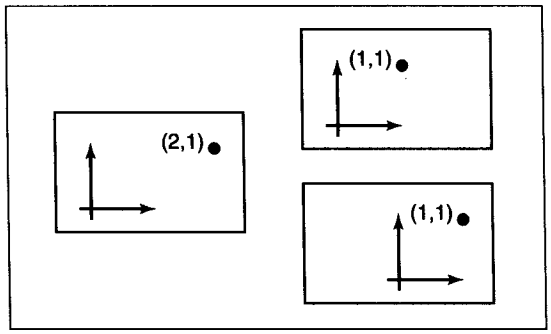


图6-17 把点（2, 1）平移到（-1, 0）。如果把平移看作是物理位置的移动，则结果如图中右上方所示。如果认为坐标系发生变化（本例中是坐标原点移动），则结果如图中右下方所示（在本例中是原点的移动）。两种方法只表明人们处理问题的策略不同，而最终点和坐标轴的相对位置是相同的

$$\boldsymbol{p}=(x_p,y_p)\equiv \boldsymbol{o}+x_p\boldsymbol{x}+y_p\boldsymbol{y}$$

例如，在图6-18中，(x_p,y_p)=(2.5, 0.9)。注意数据对(x_p,y_p)隐含地假设原点为$\boldsymbol{o}$。类似地，可用另一个公式表示$\boldsymbol{p}$：

$$\boldsymbol{p}=(u_p,v_p)\equiv \boldsymbol{e}+u_p\boldsymbol{u}+v_p\boldsymbol{v} \tag{6-7}$$

在图6-18中，有(u_p,v_p)=(0.5, -0.7)。另外，默认原点$\boldsymbol{e}$与$\boldsymbol{u}$、$\boldsymbol{v}$共同表示一个坐标系。利用本章介绍的矩阵知识，可以对坐标系进行平移：

$$\begin{bmatrix}x_p\\y_p\\1\end{bmatrix}=\begin{bmatrix}1&0&x_e\\0&1&y_e\\0&0&1\end{bmatrix}\begin{bmatrix}x_u&x_v&0\\y_u&y_v&0\\0&0&1\end{bmatrix}\begin{bmatrix}u_p\\v_p\\1\end{bmatrix}$$

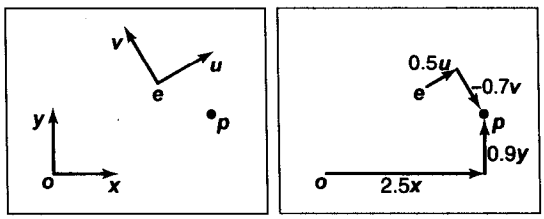


图6-18 点$\boldsymbol{p}$可以在任何一种坐标系中表示

注意，这时假设对点$\boldsymbol{e}$及向量$\boldsymbol{u}$、$\boldsymbol{v}$的表示是相对标准坐标系的。上式左边是xy坐标系中的坐标。反过来，我们会得到下式：

$$\begin{bmatrix}u_p\\v_p\\1\end{bmatrix}=\begin{bmatrix}x_u&y_u&0\\x_v&y_v&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&-x_e\\0&1&-y_e\\0&0&1\end{bmatrix}\begin{bmatrix}x_p\\y_p\\1\end{bmatrix}$$

与二维情况类似，在三维空间中可以得到

$$\begin{bmatrix}x_p\\y_p\\z_p\\1\end{bmatrix}=\begin{bmatrix}1&0&0&x_e\\0&1&0&y_e\\0&0&1&z_e\\0&0&0&1\end{bmatrix}\begin{bmatrix}x_u&x_v&x_w&0\\y_u&y_v&y_w&0\\z_u&z_v&z_w&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}u_p\\v_p\\w_p\\1\end{bmatrix} \tag{6-8}$$

$$\begin{bmatrix} u_p \\ v_p \\ w_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \tag{6-9}$$

156

常见问题

- **我可以不使用矩阵而直接编写变换代码吗？**

可以，但在实际中很难推导，不容易调试，而且不比矩阵形式更有效。当前所有的图形学API都使用这种矩阵形式。因此即使你使用图形函数库，也必须弄明白矩阵形式。

- **矩阵的最后一行总是（0,0,0,1），我必须保存这一行吗？**

不必，除非你采用透视变换（见第7章）。

文献注记

有关正态变换特性的内容参考了*Properties of Surface Normal Transformations*（Turkowski，1990）一书。20世纪90年代中期的许多论述中，向量都采用行向量，而且是左乘，例如$\boldsymbol{b}=\Delta\boldsymbol{aM}$。如果采用我们的表示方法，就是$\boldsymbol{b}^T=\boldsymbol{a}^T\boldsymbol{M}^T$。如果求一个旋转矩阵$\boldsymbol{R}$，将向量$\boldsymbol{a}$变换为等长的向量$\boldsymbol{b}$，即$\boldsymbol{b}=\boldsymbol{Ra}$，则可以采用对标准正交基做两次旋转的方法。更有效的方法参见*Efficiently Building a Matrix to Rotate One Vector to Another*（Möller & Hughes, 1999）一书。

习题

1. 证明旋转矩阵的逆矩阵就是它的转置矩阵。
2. 写出使目标移动(x_m, y_m, z_m)的4×4三维矩阵。
3. 写出使目标绕y轴旋转θ角的4×4三维矩阵。
4. 写出在所有方向缩小目标50%的4×4三维矩阵。
5. 写出使目标顺时针旋转90°的二维旋转矩阵。
6. 将习题5中的矩阵，写成三个切变矩阵的乘积形式。
7. 用文字说明下面二维变换矩阵的作用。

$$\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

157

8. 写出使二维点绕点$\boldsymbol{p}=(x_p, y_p)$旋转$\theta$角的3×3矩阵。
9. 将标准正交的三维向量$\boldsymbol{u}=(x_u, y_u, z_u)$，$\boldsymbol{v}=(x_v, y_v, z_v)$和$\boldsymbol{w}=(x_w, y_w, z_w)$，变换成标准正交的三维向量$\boldsymbol{a}=(x_a, y_a, z_a)$，$\boldsymbol{b}=(x_b, y_b, z_b)$，$\boldsymbol{c}=(x_c, y_c, z_c)$，也就是$\boldsymbol{Mu}=\boldsymbol{a}$，$\boldsymbol{Mv}=\boldsymbol{b}$，$\boldsymbol{Mw}=\boldsymbol{c}$。写出这样的4×4旋转矩阵。

158

10. 上题中的4×4旋转矩阵的逆矩阵是什么？

第7章

观　察

我们可以将上一章中学习的变换工具直接用于建立三维线段的图像。本章中将学习空间中三维线段的三维正射投影和透视投影观察方法，暂不考虑“隐藏线”消除（图7-1）。注意在正射投影下，三维空间中的平行线在图像中是平行的，但是在透视投影下，它们可能就不平行了。本章中假设三维线段由两个端点定义，端点的形式为(x,y,z)。下一章中将利用BSP树和z缓冲器算法对透明物体进行隐藏线和隐藏面的消除，并将使用三角形的面而不是三角形的边。

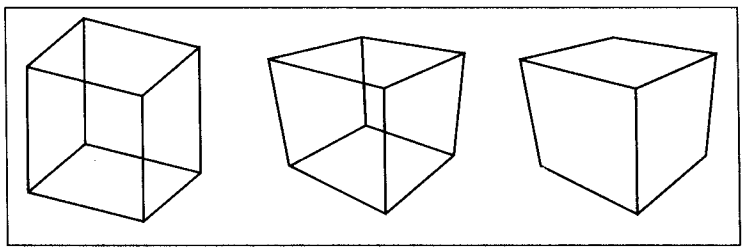


图7-1　左：正射投影。中：透视投影。右：带有隐藏线消除的透视投影

7.1　绘制标准视体

现在从这样一个问题开始，问题的答案在任何观察条件下都将用到。假设要把直线沿z轴正方向映射到屏幕上。这里将把对物体的投影限制在标准[1]视体之内。视体可以通过所有笛卡儿坐标在+1和−1之间的三维点来定义。这个立方体可以定义为$(x,y,z)\in[-1,1]^3$（图7-2）。

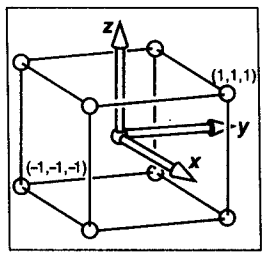


图7-2　标准视体是一个立方体，其边长为2，中心在原点

如果屏幕有$n_x\times n_y$个像素，我们把$x=-1$投影到屏幕的左半边，把$x=+1$投影到屏幕的右半边，把$y=-1$投影到屏幕的下半边，把$y=+1$投影到屏幕的上半边。注意这可能会把正方形区域$[-1,+1]^2$投影到非正方形区域。这并不成问题，因为当x和y转换为像素坐标时，可以设置不同的缩放参数。现在假设所有要绘制的线段都完全在标准视体内。后面讨论裁剪时将放宽这个假设。

回想一下，第3章中把像素扩展为一个有限的正方形，坐标从像素中心有0.5个单位的过冲，最小像素的中心坐标为(0, 0)；一个$n_x\times n_y$的屏幕具有整数个像素中心，其边界定义为$[-0.5, n_x-0.5]\times[-0.5, n_y-0.5]$。这就是式（6-6）中定义的窗口变换。

1. 术语“标准”会出现在很多语境中，通常指某些惯用的变量或者形状处于“完美”的状态。例如，如果我们要定义一个“标准圆”，可能将定义一个中心在原点半径为1的圆。

$$\begin{bmatrix} x_{\text{pixel}} \\ y_{\text{pixel}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix} \tag{7-1}$$

注意这里不考虑z值。实践中z值将被保存下来，但是如果所有的面都已正确地排序也可以不保存。为了使矩阵小一点，现在我们忽略z值。

另外，回想一下在某些API中y轴方向是向下的。现在考虑这种情况，因为它是一个很好的练习。因为y值要被翻转，它建立的变换会有点奇怪。把标准视体内的点转换为像素坐标的变换只是窗口变换（式（6-6））的一个变形，但是由于存在翻转操作，我们将从下面的草图中进行推导。

160

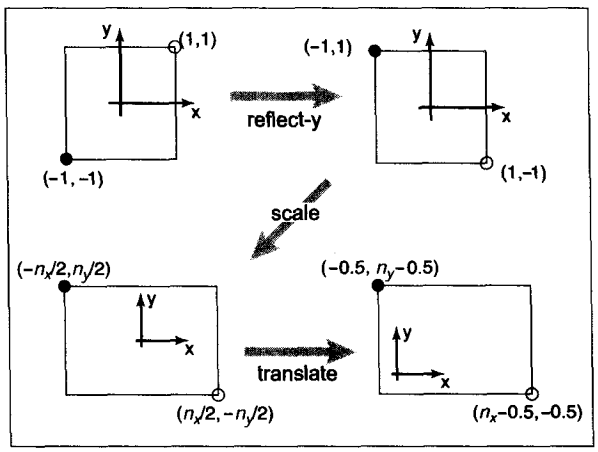


图7-3　当屏幕y坐标从上向下递增时，把标准视体中的xy项转换为像素坐标的变换中包括一个翻转变换

按照图7-3中的过程完成这个变换，变换公式为

$$\begin{bmatrix} x_{\text{pixel}} \\ y_{\text{pixel}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{n_x-1}{2} \\ 0 & 1 & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{n_x}{2} & 0 & 0 \\ 0 & \frac{n_y}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\ 0 & -\frac{n_y}{2} & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix} \tag{7-2}$$

同时利用求出的多个矩阵进行变换可以省去很多麻烦。例如，式（7-1）可以用3个简单的矩阵实现。如果把反射矩阵、缩放矩阵和平移矩阵的初始化程序作为一个矩阵库，在很多情况下都可以重用这些矩阵。同时这样得到的代码易于调试；注意把式（7-2）与图7-3相关联是很简单的。如果手工展开完整的代数公式可能会出错，并且公式包含的信息可能比较少；通过矩阵乘积，可

以看到先进行了缩放/翻转，然后进行平移。如果用展开的公式就很难能看出来。所以在实现过程中要尽量使用矩阵相乘。除了多一点预处理时间之外，这并不会影响系统的效率，而且程序将更易于修改，正确率会更高。 161

7.2 正射投影

通常要在空间的某些区域内，而不是标准视体内绘制直线。其基本步骤是利用矩阵M把每一条线段的端点a和b转换为标准视体中的点Ma和Mb。矩阵M可以处理观察和投影中的所有几何模型。

最简单的情况是当视体为一个轴向的立方体$[l, r]\times[b, t]\times[n, f]$，如图7-4所示。这个立方体称为正射投影视体（orthographic view volume），它的边界通过下面的平面界定：

$$x = l \equiv \text{左平面},$$
$$x = r \equiv \text{右平面},$$
$$y = b \equiv \text{下平面},$$
$$y = t \equiv \text{上平面},$$
$$z = n \equiv \text{近平面},$$
$$z = f \equiv \text{远平面}。$$

这样定义是假设观察者沿着z轴负方向观察，头指向y轴方向[1]。隐含的条件是$n > f$，这可能不符合我们的直觉。但是如果假设整个正射投影视体的z值为负，那么当且仅当$n > f$时平面$z = n$是距离观察者最近的平面；这里f是一个比n小的数，即绝对值比n大的负数。

这个概念如图7-5所示。从正射投影视体到标准视体的变换只是把6.3.1节中的二维窗口变换变成了三维变换。

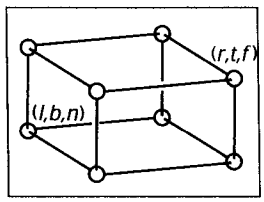


图7-4 正射投影视体

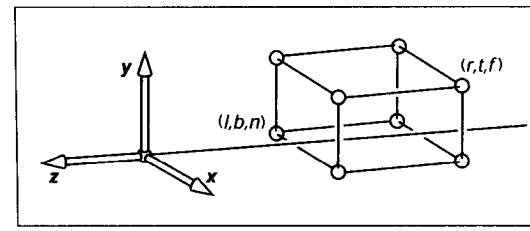


图7-5 正射投影视体沿着z轴负方向，所以f是一个比n小的负数，$n > f$

这个变换把$y = b$对应为$y = -1$，$y = t$对应为$y = +1$，$x = l$对应为$x = -1$，$z = n$对应为$z = 1$，$z = f$对应为$z = -1$，变换可以看成是先缩放后平移，或者先平移后缩放。注意$[n, f]$的取值，f是z轴上一个比n小的负数。n代表“近平面”，f代表“远平面”。在很多程序中可能会把它们翻转过来，后面将进行详细讨论。现在给出符合大多数人直觉的原始公式： 162

$$\begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ z_{\text{canonical}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{7-3}$$

1. 许多程序员发现x轴指向右y轴向上更符合直觉。在右手坐标系中，这意味着我们向z轴负方向看。某些系统在观察时使用左手坐标系，这样观察方向就沿着z轴正方向。哪一种方式最好是个人爱好问题，本书中假设使用右手坐标系。本章末尾的文献注记中给出了讨论使用左手坐标系的参考文献。

注意因为这里计算了z值，所以使用的是4×4的变换矩阵，并且$n-f$是一个正数。

为了在正射投影视体内绘制三维线段，把线段投影到xy坐标屏幕上，并忽略掉z坐标。可以把式（7-1）和式（7-3）合并起来实现投影。但是因为式（7-1）是一个二维变换，所以要在z上添加一个“空操作”。

$$\boldsymbol{M}_o=\begin{bmatrix}\frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2}\\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2}\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{2}{r-l} & 0 & 0 & 0\\ 0 & \frac{2}{t-b} & 0 & 0\\ 0 & 0 & \frac{2}{n-f} & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & -\frac{l+r}{2}\\ 0 & 1 & 0 & -\frac{b+t}{2}\\ 0 & 0 & 1 & -\frac{n+f}{2}\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{7-4}$$

注意在程序中把这三个矩阵乘在一起形成了一个矩阵$\boldsymbol{M}_o$，然后对点进行操作：

$$\begin{bmatrix}x_{\text{pixel}}\\ y_{\text{pixel}}\\ z_{\text{canonical}}\\ 1\end{bmatrix}=\boldsymbol{M}_o\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

z坐标的取值范围是[−1, 1]。现在先不这样用，但是它在后面z缓冲器检测算法中是很有用的。

163　注意也可以按照三维窗口变换得到式（7-4）。把式（6-6）推广到三维空间中是很直接的，为了把三维立方体$[a, A]\times[b, B]\times[c, C]$对应到三维立方体$[d, D]\times[e, E]\times[f, F]$上，有

$$\text{三维窗口}=\begin{bmatrix}\frac{D-\mathrm{d}}{A-a} & 0 & 0 & \frac{\mathrm{d}A-Da}{A-a}\\ 0 & \frac{E-e}{B-b} & 0 & \frac{eB-Eb}{B-b}\\ 0 & 0 & \frac{F-f}{C-c} & \frac{fC-Fc}{C-c}\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{7-5}$$

式（7-4）把$[l, r]\times[t, b]\times[n, f]$映射到$[-0.5, n_x-0.5]\times[-0.5, n_y-0.5]\times[-1, 1]$，可以直接把这种映射代入式（7-5）中得到需要的矩阵。

这样绘制三维线段的程序就变得简单而有效了。假设线段的端点为$\boldsymbol{a}_i$和$\boldsymbol{b}_i$：

```
compute M_o
for each line segment(a_i, b_i) do
  p = M_o a_i
  q = M_o b_i
  drawline(x_p, y_p, x_q, y_q)
```

这是利用矩阵变换机制简化图形学程序的第一个例子。

任意视点

我们希望可以在三维空间中任意改变视点和观察方向。指定观察者的位置和方向的方法有很多，这里将使用下列方法（见图7-6）：

- 眼睛位置$\boldsymbol{e}$。
- 凝视方向$\boldsymbol{g}$。
- 向上向量$\boldsymbol{t}$。

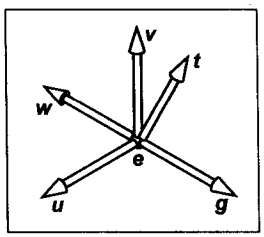


图7-6 用户通过眼睛位置e：凝视方向g和一个向上向量t指定了一个观察。建立一个右手坐标系：w指向g的反方向，v在g和t所确定的平面中

眼睛位置是眼睛看出去的地方，如果把图形学看作是一个拍照片的过程，它就是镜片的中心。凝视方向是一个向量，观察者沿向量方向观察。假设观察者站在地面上，用一个平面把观察者的头分为左右两半，向上向量就位于该平面上，方向为“指向天空”的方向。使用这些向量就可以建立一个以e为原点，以uvw为坐标轴的坐标系：

$$w = -\frac{g}{\|g\|}$$

$$u = \frac{t \times w}{\|t \times w\|}$$

$$v = w \times u$$

164

注意如果所有要变换的点都在以e为原点的uvw坐标系中，那就不需要作任何变换了，但是它们存储在以o为原点的xyz坐标系中，如图7-7所示。要使用已经建立的机制，只需要把线段的顶点坐标转换为相对于原点e的偏移(u, v, w)坐标。同样的，可以把这个变换看作是把e移动到原点，并且使uvw与xyz平行。所以观察变换是

$$M_v = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

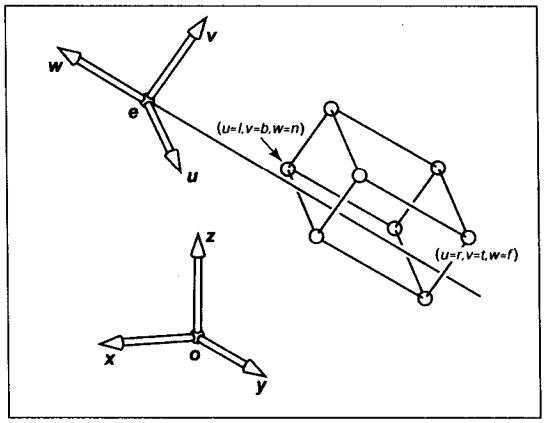


图7-7 对于任意的观察，需要改变这些点把它们存储在“正确”的坐标系中。在上述情况下，原点为e，坐标偏移用uvw表示

如果把点p和M_v相乘，就可以把它与新的坐标轴“对齐”。这样只需要对前面沿着z轴观察的程序作一点调整即可：

compute $\boldsymbol{M}_v$
compute $\boldsymbol{M}_o$
$\boldsymbol{M} = \boldsymbol{M}_o\boldsymbol{M}_v$
for each line segment$(\boldsymbol{a}_i,\boldsymbol{b}_i)$ **do**
  $\boldsymbol{p} = \boldsymbol{M}\boldsymbol{a}_i$
  $\boldsymbol{q} = \boldsymbol{M}\boldsymbol{b}_i$
  drawline(x_p, y_p, x_q, y_q)

这证明了利用变换矩阵实现算法是非常有效的。一旦建立了矩阵变换机制就几乎不需要任何
165 代码。

7.3 透视投影

为了获得透视的效果，距离观察者远的线段要画得比距离观察者近的线段短。我们希望能够自动绘制出符合美术传统的三点透视的效果（见图7-8）。事实上只要遵守透视中简单的数学规律
166 就可以了：将物体直接向眼睛投影，并画在投影线与眼睛前面的观察平面相交的位置。

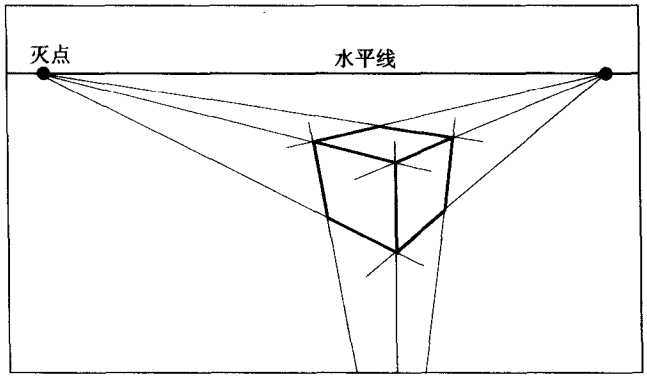


图7-8　在三点透视中，画家要选择平行线相交处的灭点。水平平行线将相交在水平线上。每一组平行线都有自己的灭点。如果我们能够按照正确的几何原理进行投影自然就可以遵守这些规律

实际上，如果视点在原点，观察方向为z轴的负方向，物体在屏幕上的大小与$1/z$成正比。式（7-6）可以准确地表示出图7-9中的几何图形：

$$y_s = \frac{\mathrm{d}}{z}y \tag{7-6}$$

其中y是点在y轴方向到原点的距离，y_s是点在屏幕上画的位置。

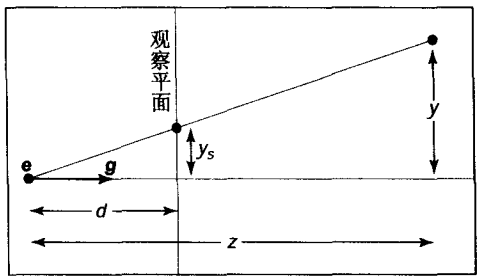


图7-9　式（7-6）的几何解释。观察者的眼睛在$\boldsymbol{e}$点，凝视方向为$\boldsymbol{g}$（z轴负方向）。观察平面距眼睛的距离为d。某一点向$\boldsymbol{e}$点做投影，投影线与观察平面相交的位置就是画该点的位置

我们希望能利用前面为正射投影建立的矩阵机制来绘制透视图像，那样只需要在前面得到的矩阵上乘以另一个矩阵，就能直接利用原有的算法。可以通过几何变换使通过视点的投影线平行于z轴，如图7-10所示。注意无论点被变换到z轴的什么位置都没关系，因为在正射投影中将会忽略z值。如果要使上述的变换能同正射投影一起使用，那么要将观察平面设为$z=n$，观察平面上$(u,v)\in[l,r]\times[b,t]$的部分就会显示出来。上述变换如图7-11所示。上述变换有这样一个性质，所有通过眼睛的直线都变换为平行线，而它们与平面$z=n$的交点并没有变（图7-12）。

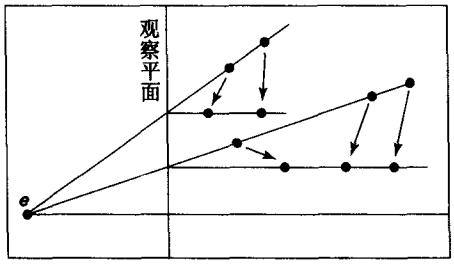


图7-10 只要所有通过眼睛的直线都平行于凝视方向，透视投影就可以通过正射投影的方法计算。无论通过眼睛的直线与观察平面相交于什么位置，转换后的直线都会与观察平面相交

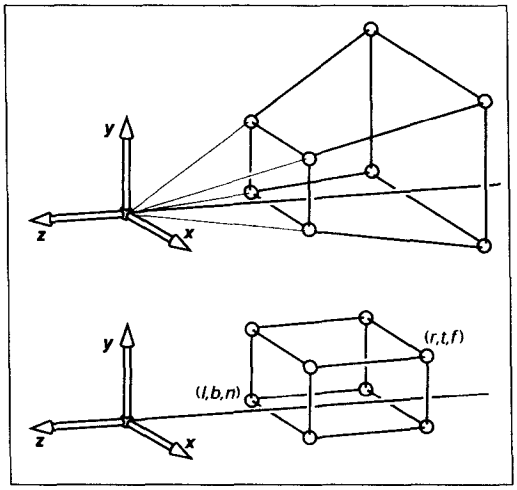


图7-11 透视投影不改变$z=n$平面上的点，并把透视投影体后面大的矩形$z=f$ 映射到正射投影体后面小的矩形$z=f$上

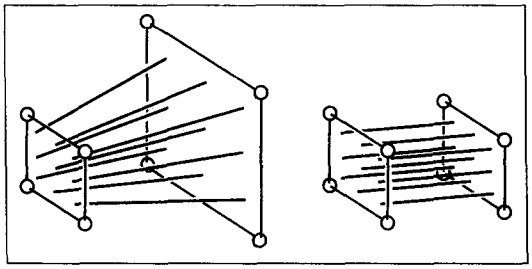


图7-12 透视投影把所有通过原点/眼睛的直线映射到平行于z轴的直线上，而并不移动$z=n$平面中直线上点的位置

希望能写出一个实现上述变换的矩阵，但是现在还不行；目前建立的矩阵机制中并没有式（7-6）中除以z的运算。不过有一个简单而漂亮的扩展方法能够实现“透视中z值的除法运算”。关键是要使用点的第4维坐标，目前它一直都被设为1，现在允许把它设为其他的值。 167 ~ 168

（三维点的）第4维坐标称为齐次坐标h。这个坐标实际上表示了其他三维坐标的缩放程度。因为最终只关心(x,y,z)，所以定义下面的等式：

$$\begin{bmatrix} hx \\ hy \\ hz \\ h \end{bmatrix} \equiv \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

这样，如果点的第4维（齐次）坐标不是1，也可以计算，但是按照传统的笛卡儿坐标计算之前必须把其他的三个坐标除以h。例如，图7-13中齐次的4维“直线”$x = 1.5h$上所有点的x坐标都是相同的，如

$$\begin{bmatrix} 6 \\ -2 \\ 8 \\ 2 \end{bmatrix} \xrightarrow{\text{齐次化}} \begin{bmatrix} 3 \\ -1 \\ 4 \\ 1 \end{bmatrix}$$

齐次化过程中只要除以第4维坐标就可以得到“真实”的位置。这样可以构造透视矩阵

$$\boldsymbol{M}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{n+f}{n} & -f \\ 0 & 0 & \dfrac{1}{n} & 0 \end{bmatrix}$$

169

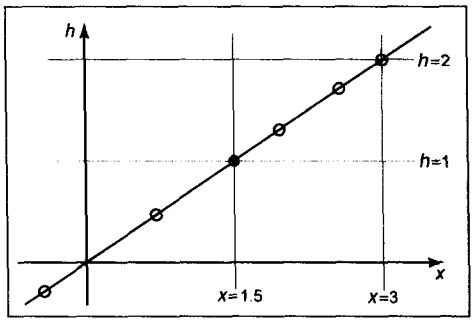


图7-13　齐次值$x = 1.5$可以用直线$x = 1.5h$上任意的点表示，如图中的空心圆点。然而，要把x解释成传统的笛卡儿坐标，首先要除以h得到$(x, h) = (1.5, 1)$，如图中的黑点

有许多矩阵都可以作为投影矩阵，但是它们都会使z轴产生非线性变形。图7-11和图7-12中的矩阵具有非常好的性质：它使平面$z = n$上的点保持不变，并使平面$z = f$上点的x和y坐标按照一定比例进行“压缩”，而z坐标不变。矩阵作用在点(x,y,z)上的效果是

$$\boldsymbol{M}_p \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z\dfrac{n+f}{n} - f \\ \dfrac{z}{n} \end{bmatrix} \xrightarrow{\text{齐次化}} \begin{bmatrix} \dfrac{nx}{z} \\ \dfrac{ny}{z} \\ n+f-\dfrac{fn}{z} \\ 1 \end{bmatrix}$$

就像看到的那样，x和y被缩放了，而更重要的是，它们被z除了。因为n和z（在视体内）都是负的，所以在x和y方向都没有“翻转”。尽管并不明显，这个变换的确保持了位于平面$z = n$和$z = f$之间的z值的相对顺序（见本章后面的习题）。这样在使用该矩阵之后就可以进行深度排序。这在后面进行隐藏面消除时是非常重要的。

对于齐次点$\boldsymbol{p} = h\boldsymbol{p}$，或者更显式表示的$(x, y, z, 1) = (hx, hy, hz, h)$，因为$\boldsymbol{M}(h\boldsymbol{p}) = (h\boldsymbol{M})\boldsymbol{p} = \boldsymbol{M}\boldsymbol{p}$，所以可以使用任意的矩阵$\boldsymbol{M}$，并乘以一个任意常数。为此，可以把投影矩阵乘以$n$，使之更漂亮：

$$\boldsymbol{M}_p = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

有时需要求$\boldsymbol{M}_p$的逆矩阵，例如当进行拾取操作时需要把屏幕坐标和z值映射到原来的空间中。逆矩阵是

$$\boldsymbol{M}_p^{-1} = \begin{bmatrix} \frac{1}{n} & 0 & 0 & 0 \\ 0 & \frac{1}{n} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{fn} & \frac{n+f}{fn} \end{bmatrix}$$

正如前面提到的，如果把齐次变换矩阵乘以一个常数，它的效果并不发生变化。这对逆矩阵也是成立的；矩阵及其逆矩阵只需要产生一个对角线上具有单位值的对角阵。所以上面的逆矩阵是任何透视矩阵的逆矩阵。把它乘以 nf 可以得到另一个逆矩阵： 170

$$\boldsymbol{M}_p^{-1} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & fn \\ 0 & 0 & -1 & n+f \end{bmatrix}$$

投影矩阵的价值在于，一旦我们使用这个矩阵就可以利用一个正射投影变换获得标准视体。这样，所有的正射投影机制都用到了，而我们需要添加的只不过是一个矩阵和一个除法运算。令人高兴的还有我们没有“浪费”4×4矩阵的最后一行！

然而现在的问题是如何确定特定透视投影下的l, r, b, t？它们定义了我们观察的“窗口”。透视矩阵并不改变平面$z = n$上x和y值，所以可以在这个平面上指定(l, r, b, t)。

我们希望把透视投影矩阵集成到正射投影的体系中。关键是透视矩阵假设我们是沿z轴负方向观察的，所以应该先应用观察矩阵$\boldsymbol{M}_v$，再应用透视矩阵。这样透视观察的完整矩阵集是

$$\boldsymbol{M} = \boldsymbol{M}_o \boldsymbol{M}_p \boldsymbol{M}_v$$

最终的算法是

compute $\boldsymbol{M}_o$
compute $\boldsymbol{M}_v$
compute $\boldsymbol{M}_p$
$\boldsymbol{M} = \boldsymbol{M}_o \boldsymbol{M}_p \boldsymbol{M}_v$
for each line segment$(\boldsymbol{a}_i, \boldsymbol{b}_i)$ **do**
  $\boldsymbol{p} = \boldsymbol{M}\boldsymbol{a}_i$
  $\boldsymbol{q} = \boldsymbol{M}\boldsymbol{b}_i$
  drawline$(x_p/h_p, y_p/h_p, x_q/h_q, y_q/h_q)$

注意程序中除了增加了一个矩阵外，唯一的变化是用齐次坐标h除x和y坐标。

如果得到的透视矩阵是错的，它的齐次坐标值可能被修改了，那么平移和缩放操作就不对了。但是注意到对于齐次点的移动，有

$$\begin{bmatrix}1&0&0&t_x\\0&1&0&t_y\\0&0&1&t_z\\0&0&0&1\end{bmatrix}\begin{bmatrix}hx\\hy\\hz\\h\end{bmatrix}=\begin{bmatrix}hx+ht_x\\hy+ht_y\\hz+ht_z\\h\end{bmatrix}\xrightarrow{\text{齐次化}}\begin{bmatrix}x+t_x\\y+t_y\\z+t_z\\1\end{bmatrix}$$

171

对于其他的变换也有相似的效果（见本章后面的习题）。

矩阵$\boldsymbol{M}_o\,\boldsymbol{M}_p$相乘后通常称为投影矩阵。把$\boldsymbol{M}_o$和$\boldsymbol{M}_p$相乘得

$$\boldsymbol{M}_{\text{projection}}=\begin{bmatrix}\frac{2n}{r-l}&0&\frac{l+r}{l-r}&0\\0&\frac{2n}{t-b}&\frac{b+t}{b-t}&0\\0&0&\frac{f+n}{n-f}&\frac{2fn}{f-n}\\0&0&1&0\end{bmatrix}$$

这个矩阵或类似矩阵经常出现在文档中，当你认识到它们通常是几个简单矩阵相乘的结果时，它们就没有那么神秘了。

许多像 *OpenGL*(Shreiner, Neider, Woo, & Davis, 2004)一样的 API 使用是和这里一样标准视体。通常它们也使用用户指定的 n 和 f 的绝对值。*OpenGL* 的投影矩阵是

$$\boldsymbol{M}_{\text{OpenGL}}=\begin{bmatrix}\frac{2|n|}{r-l}&0&\frac{r+l}{r-l}&0\\0&\frac{2|n|}{t-b}&\frac{t+b}{t-b}&0\\0&0&\frac{|n|+|f|}{|n|-|f|}&\frac{2|f||n|}{|n|-|f|}\\0&0&-1&0\end{bmatrix}$$

其他API将n和f分别设为0和1。为了提供效率，Blinn（J. Blinn, 1996）建议将标准视体设为$[0, 1]^3$。所有这样的设置都要略微修改投影矩阵。

7.4　透视变换的性质

透视变换的一个重要性质是它把直线映射到直线，平面映射到平面。另外它把视体内的线段映射到标准视体中的线段。为了证明这个性质，考虑下面的线段：

$$\boldsymbol{q}+t(\boldsymbol{Q}-\boldsymbol{q})$$

172　当用一个4×4的矩阵$\boldsymbol{M}$进行变换后，它是一个点，其齐次坐标可能会发生变化：

$$\boldsymbol{Mq}+t(\boldsymbol{MQ}-\boldsymbol{Mq})\equiv\boldsymbol{r}+t(\boldsymbol{R}-\boldsymbol{r})$$

齐次化的三维线段是

$$\frac{\boldsymbol{r}+t(\boldsymbol{R}-\boldsymbol{r})}{h_r+t(h_R-h_r)}\tag{7-7}$$

如果式（7-7）可以写成下面的形式：

$$\frac{\boldsymbol{r}}{h_r}+f(t)\left(\frac{\boldsymbol{R}}{h_R}-\frac{\boldsymbol{r}}{h_r}\right)\tag{7-8}$$

则所有齐次化的点都位于一条三维直线上。把式（7-7）进行蛮力处理可以得到下面的形式：

$$f(t)=\frac{h_R t}{h_r+t(h_R-h_r)} \tag{7-9}$$

它也证明了线段的确映射到线段，并保留了点的顺序（习题8），即所有点没有进行重新排序，也没有被“撕开”。

变换把线段映射到线段，同时把三角形的点和边映射到另一个三角形的点和边。所以，它把三角形映射到三角形，把平面映射到平面。

7.5 视域

我们可以通过设置(l, r, b, t)和n的值任意定义窗口，有时希望定义一个比较简单的系统，该系统是从窗口的中心看出去。这隐含了一个限制条件：

$$l=-r$$

$$b=-t$$

如果再增加一个限制，像素是正方形的，即图像中没有变形，那么r与t的比值应该等于水平和垂直方向像素数之比：

$$\frac{n_x}{n_y}=\frac{r}{t}$$

一旦指定了n_x和n_y就只有一个自由度了。这往往用视域（field-of-view）来定义，在图7-14中用θ表示。为了区别于左右边之间的角度或者对角线间的角度，有时称它为垂直视域。从图中可以看出

$$\tan\frac{\theta}{2}=\frac{t}{|n|}$$

173

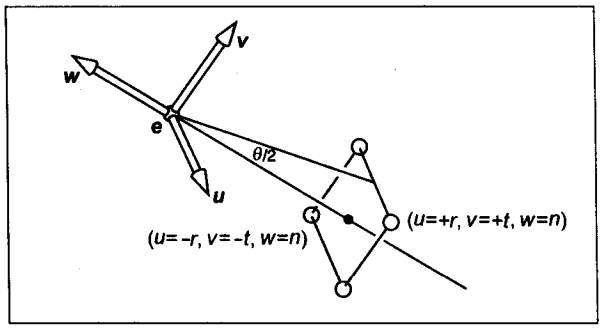


图7-14 视域θ是屏幕的上边和下边相对于视点的张角

如果指定了n和θ，就可以得出t，并且可以通过编程应用于更一般的观察系统中。在某些系统中很难用程序将n设置为一个合理的值，所以我们还有一点自由度。

常见问题

- **在实践中正射投影有用吗？**

在需要判断相对长度的应用中它是很重要的。像在某些医学可视化应用系统中，正射投影计算很简单，而透视投影计算代价非常高。

- **在透视投影下我画的镶嵌球体像一个椭圆。这是一个错误吗？**

不，这是对的。如果眼睛相对于屏幕的位置和虚拟观察者相对于景物的位置相同，这些椭圆就会看起来像圆，因为它们是从某个特定的角度观察的。

- **透视矩阵是否以反序把负的z值映射为正的z值？这是否会出现问题？**

是的。对z的变换公式是

$$z' = n + f - \frac{fn}{z}$$

174 所以$z = +\varepsilon$转换为$z' = -\infty$，$z = -\varepsilon$转换为$z' = \infty$。因此尽管所有点都能正确地投影到屏幕上，但跨过$z = 0$的线段将被“撕开”。如果所有物体都在视体内当然就不会被撕开。通常可以用视体进行裁剪来保证这一点。不过因为存在撕开的现象，裁剪本身也会变得很复杂，这将在第12章中讨论。

文献注记

本章关于观察矩阵的讨论来自*Real-Time Rendering*（Möller & Haines, 1999）、*OpenGL Programming Guide*（Shreiner et al., 2004）、*Computer Graphics*（Hearn & Baker, 1986）和*3D Game Engine Design*（Eberly, 2000）。

习题

1. 用代数方法证明透视矩阵保持了视体内z值的顺序。
2. 设有一个4×4的矩阵，它的上面三行的数是任意的，最后一行是(0, 0, 0, 1)，证明点$(x, y, z, 1)$和(hx, hy, hz, h)在齐次化后变换为同一个点。
3. 验证本章中给出的$\boldsymbol{M}_p^{-1}$形式的正确性。
4. 验证到标准矩阵$\boldsymbol{M}_{\text{projection}}$的完全透视把$(r, t, n)$映射到$(1, 1, -1)$。
5. 写出$n = 1$、$f = 2$的透视矩阵。
6. 对于点$\boldsymbol{p} = (x, y, z, 1)$，如果用习题3中的透视矩阵进行变换，得到的齐次化的和非齐次化的结果各是多少？
7. 对于眼睛位置$\boldsymbol{e} = (0, 1, 0)$，观察向量$\boldsymbol{g} = (0, -1, 0)$，向上向量$\boldsymbol{t} = (1, 1, 0)$，用于坐标旋转的正交基$\boldsymbol{uvw}$是多少？
8. 证明：对于透视变换，起点在视体内的线段齐次化后确实映射到标准视体内的线段上。并证明两个线段上点的相对顺序是相同的。提示：证明式（7-9）中$f(t)$具有以下性质：$f(0) = 0$，$f(1) = 1$，
175 对于所有的$t \in [0,1]$，f的导数为正，并且齐次坐标并不改变正负号。

第8章

隐藏面消除

现在已经知道如何在屏幕上绘制一个三角形，即把一个三角形的顶点从三维空间投影到标准视体中，如果再进行隐藏面消除（hidden surface elimination），也就是只让最近的面对观察者可见，就能获得更好的真实性。有很多方法可以实现隐藏面消除；这里只介绍两种最常用的方法：BSP树（Fuchs, Kedem, & Naylor, 1980）和z缓冲器方法（Catmull, 1975）。光线跟踪也可以看作是一种隐藏面消除的方法，但是因为它不太符合标准的投影—光栅化过程，所以把它作为一章单独进行讨论。

隐藏面消除算法很多（Sutherland，Sproull, & Schumacker, 1974），但除了上述三种外，大都很少应用于实践。

8.1 BSP树

对同一个几何模型，如果要从不同的视点生成多幅图像，就像在类似游戏的应用中那样，可以利用二叉空间划分（Binary Space Partition，BSP）树算法对所有的表面按照从前到后的顺序进行排序。BSP树的关键是它利用预处理过程建立了一个数据结构，该结构对任何的视点都是适用的。那么当视点改变时，就可以利用同样的数据结构，而不必进行修改。 177

8.1.1 BSP树算法概述

BSP树算法是画家算法（painter's algorithm）的一个例子。画家算法从后向前依次画出所有的对象，新的多边形将可能会覆盖以前的多边形，如图8-1所示。算法的实现如下：

```
sort objects back to front relative to viewpoint
for each object do
  draw object on screen
```

第一步（排序）的问题是，多个对象的相对顺序往往很难定义，即使是每两个对象之间的顺序。问题如图8-2所示，其中三个三角形形成了一个循环。 178

BSP树只能用于特定的场景，即场景中所有的多边形都不穿越其他多边形所在的平面。利用预处理过程可以放宽这个限制。在下面的讨论中，假设三角形是唯一的图元，不过这种思想可以扩展到任意多边形。

下面通过两个三角形T_1和T_2说明BSP树的基本思想。首先回忆（见2.7节）一下包含T_1的平面的隐式方程：$f_1(\boldsymbol{p}) = 0$。隐式平面有一个重要性质，即对于所有位于平面一侧的点$\boldsymbol{p}^+$，$f_1(\boldsymbol{p}^+) > 0$；对于所有位于平面另一侧的点$\boldsymbol{p}^-$，$f_1(\boldsymbol{p}^-) < 0$。利用这个性质，可以判断平面$T_2$位于$T_1$的哪一侧。再假设$T_2$的所有三个顶点都位于平面的同一侧。为了讨论方便，假设T_2全部位于平面$f_1(\boldsymbol{p}^-) < 0$的一侧。那么对于任何视点$\boldsymbol{e}$都可以按照正确的顺序画出$T_1$和$T_2$：

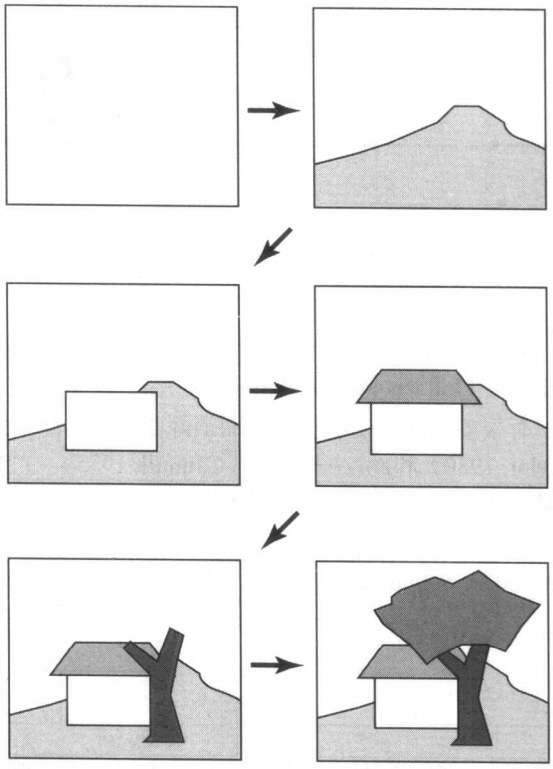


图8-1　画家算法从一幅空的图像开始，然后从后向前画出整个场景，一次绘制一个对象，并覆盖原有的图像。这样就自动消除了隐藏面

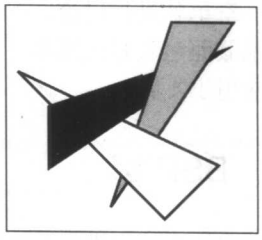


图8-2　如果相对于特定的眼睛位置，对象不能完全按照从后到前的顺序排序，将会产生一个循环

if ($f_1(\boldsymbol{e}) < 0$) **then**
  draw T_1
  draw T_2
else
  draw T_2
  draw T_1

如果T_2和$\boldsymbol{e}$位于包含T_1的平面的同一侧，从$\boldsymbol{e}$点进行观察时，T_2不可能被T_1完全或者部分遮挡，所以先画T_1是安全的。如果T_2和$\boldsymbol{e}$分别位于包含T_1的平面的两侧，从$\boldsymbol{e}$点进行观察时，T_1不可能被T_2完全或者部分遮挡，按照相反的顺序画也是安全的（图8-3）。

179

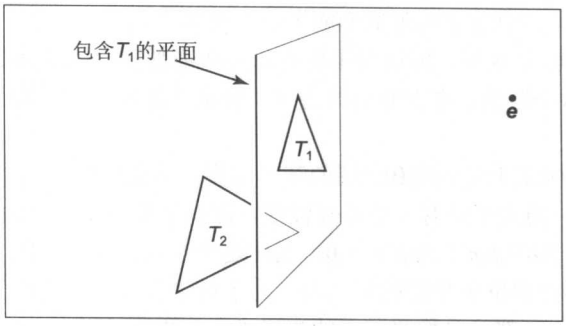


图8-3　当T_2和$\boldsymbol{e}$分别位于包含T_1平面的两侧，那么先画T_2再画T_1是安全的。如果T_2和$\boldsymbol{e}$位于包含T_1平面的同一侧，那么T_1应该在T_2之前画出。这就是BSP树算法的核心思想

只要任何对象都没有穿越T_1所定义的平面，上述结果可以扩展到很多对象上。如果使用以T_1为根结点的二叉树数据结构，树的负分支包含所有顶点满足$f_i(\boldsymbol{p}) < 0$的三角形，树的正分支包含所有顶点满足$f_i(\boldsymbol{p}) > 0$的三角形。那么可以用下面的代码按照正确的顺序画出所有对象：

```
function draw(bsptree tree, point e)
if (tree.empty) then
   return
if (f_tree.root(e) < 0) then
   draw(tree.plus, e)
   rasterize tree.triangle
   draw(tree.minus, e)
else
   draw(tree.minus, e)
   rasterize tree.triangle
   draw(tree.plus, e)
```

上述代码的好处是它对任意的视点$\boldsymbol{e}$都是适用的，所以树可以预先计算。注意，如果一个子树本身是一棵树，树的根结点根据包含它的平面把其他三角形分成两组，那么上述代码仍然是适用的。如果递归调用在更高一层终止，算法的效率可能会提高一些，而且代码依然很简单。图8-4是这段代码的图示。正如2.7.2节讨论的，点$\boldsymbol{p}$所在平面可以由三个不共线的点$\boldsymbol{a}$、$\boldsymbol{b}$和$\boldsymbol{c}$定义，其隐式方程为

$$f(\boldsymbol{p}) = ((\boldsymbol{b}-\boldsymbol{a})\times(\boldsymbol{c}-\boldsymbol{a}))\cdot(\boldsymbol{p}-\boldsymbol{a}) = 0 \tag{8-1}$$

下列形式的隐式方程可以存为(A, B, C, D)。

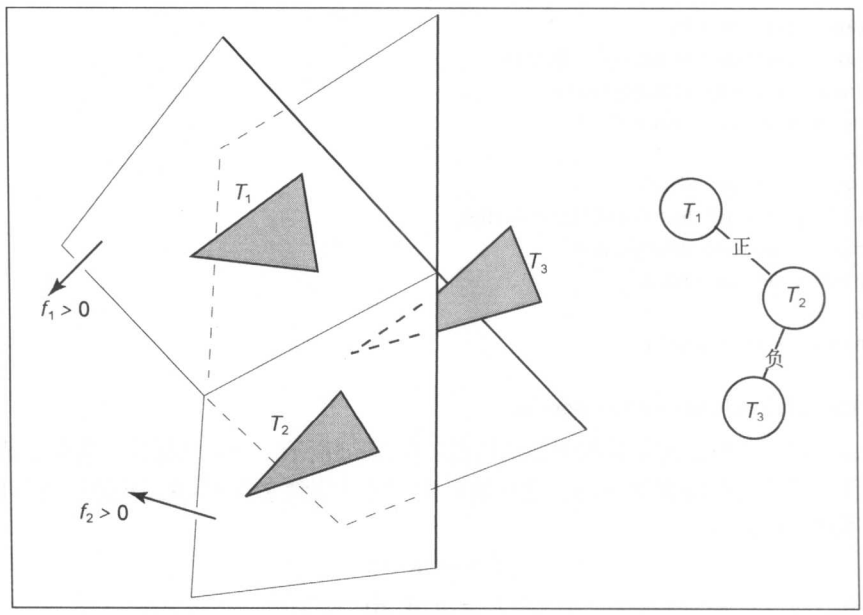


图8-4 三个三角形及其对应的BSP树。“正”和“负”分别通过右子树和左子树的位置表示

$$f(x, y, z) = Ax + By + Cz + D = 0 \tag{8-2}$$

式（8-1）和式（8-2）是等价的，就像隐式方程的梯度就是三角形的法线。式（8-2）的梯度是$\boldsymbol{n} = (A, B, C)$，这就是法向量

$$\boldsymbol{n} = (\boldsymbol{b}-\boldsymbol{a})\times(\boldsymbol{c}-\boldsymbol{a})$$

把平面上任意一点，如$\boldsymbol{a}$，代入上面的公式，就可以求得D：

$$\begin{aligned} D &= -Ax_a - By_a - Cz_a \\ &= -\boldsymbol{n}\cdot\boldsymbol{a} \end{aligned}$$

180

从而可以得到下面的形式：

$$\begin{aligned} f(\boldsymbol{p}) &= \boldsymbol{n}\cdot\boldsymbol{p} - \boldsymbol{n}\cdot\boldsymbol{a} \\ &= \boldsymbol{n}\cdot(\boldsymbol{p}-\boldsymbol{a}) \\ &= 0 \end{aligned}$$

这和式（8-1）是相同的，其中$\boldsymbol{n}$是通过叉乘计算的。究竟使用平面的哪个形式，只存顶点，存$\boldsymbol{n}$和顶点，还是存$\boldsymbol{n}$、D和顶点可能只是个人喜好问题——时间空间的均衡最好由具体情况决定。对调试而言，使用式（8-1）可能是最好的。

上述代码还不能适用于一般的情况，因为不能保证一个三角形完全位于一个平面的一侧。三角形可能有两个顶点在平面的一侧，而第三个顶点在另一侧。这个问题可以这样解决，即用平面把三角形分成几个小三角形。

8.1.2 构造 BSP 树

181

如果数据集中没有任何三角形穿越其他平面，即所有的三角形都在其他三角形的同一侧，那么就可以利用上述代码遍历BSP树，并可以利用下面的代码构造BSP树：

```
tree-root = node(T_1)
for i ∈ {2, ···, N} do
  tree-root.add(T_i)

function add (triangle T)
if (f(a) < 0 and f(b) < 0 and f(c) < 0) then
  if ( negative subtree is empty) then
    negative-subtree = node(T)
  else
    negative-subtree.add(T)
else if (f(a) > 0 and f(b) > 0 and f(c) > 0) then
  if positive subtree is empty) then
    positive-subtree = node(T)
  else
    positive-subtree.add(T)
else
  we have assumed this case is impossible
```

需要修正的只有三角形穿越了分割平面的情况，如图8-5所示。为简化起见，假设三角形的顶点$\boldsymbol{a}$和$\boldsymbol{b}$在平面的一侧，顶点$\boldsymbol{c}$在另一侧。这种情况下，可以找到交点$\boldsymbol{A}$和$\boldsymbol{B}$，并把三角形切成三个三角形，其顶点分别是：

$$\begin{aligned} T_1 &= (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{A}) \\ T_2 &= (\boldsymbol{b}, \boldsymbol{B}, \boldsymbol{A}) \\ T_3 &= (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{c}) \end{aligned}$$

如图8-6所示。顶点的顺序是很重要的，这样可以保证法线和原来的三角形是相同的。如果假设$f(\boldsymbol{c}) < 0$，下面的代码可以把这三个三角形添加到BSP树中，假设左子树和右子树都不为空：

```
positive-subtree = node (T_1)
positive-subtree = node (T_2)
negative-subtree = node (T_3)
```

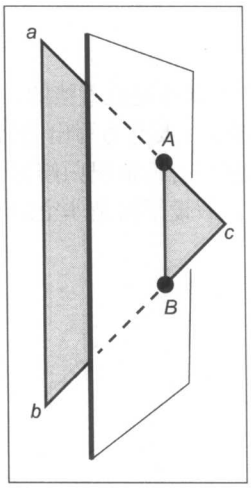


图8-5 当一个三角形穿越一个平面时，将有一个顶点在平面的一侧，有两个顶点在另一侧

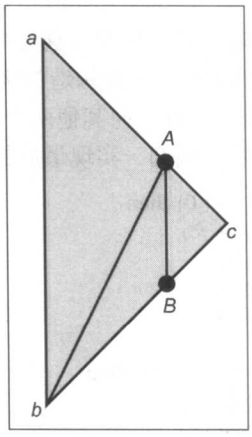


图8-6 当切割一个三角形时，它被分成三个三角形，新的三角形都不再穿越分割平面

当顶点距离分割平面非常近时，如果只简单地实现上述算法，数据精度问题将会带来巨大的问题。例如，如果有两个顶点在分割平面的一侧，而另一个顶点在另一侧，但是它距离平面非常近，那么将创建这样的三个三角形：一个和原来差不多的三角形，一个细长的三角形和一个面积几乎为零的三角形。如果能够提前检测出这种特殊情况，不对它进行分裂将会更好一些。我们可能以为这种情况很少，但是因为很多模型具有镶嵌平面，其中有很多三角形共享顶点，所以这种情况是经常发生的，必须小心进行处理。可以用一些简单的操作来处理，如下：

182

```
function add (triangle T)
fa = f(a)
fb = f(b)
fc = f(c)
if ( abs (fa) < ε ) then
  fa = 0
if ( abs (fb) < ε ) then
  fb = 0
if ( abs (fc) < ε ) then
  fc = 0
if (fa ≤ 0 and fb ≤ 0 and fc ≤ 0) then
  if ( negative subtree is empty) then
    negative-subtree = node(T)
  else
    negative-subtree.add(T)
else if (fa ≥ 0 and fb ≥ 0 and fc ≥ 0) then
  if (positive subtree is empty) then
    positive-subtree = node(T)
  else
    positive-subtree.add(T)
else
  cut triangle into three triangles and add to each side
```

该算法把 f 值在距平面 ε 范围内的顶点都作为正侧或者负侧。常数 ε 是用户给定的一个很小的正实数。上述方法是检查浮点数是否相等的一个例子，它是很有用的，因为零值是设定的而不是计算出的。建议不要比较浮点数的相等性，而且这里也并没有那样做。

8.1.3 分割三角形

完成上述“把三角形分割为三个三角形并添加到两边中”的操作是很直接的，但是很繁琐。
183 我们应该利用BSP树预先构造的优点，因为预处理的效率并不重要。要努力写出简洁紧凑的代码。假定c在平面的一侧，而其他两个顶点在另一侧，这样就可以把所有的情况归结为一种。通过交换顶点这很容易做到。实现最后的else语句的代码（简单起见，假设子树是非空的）如下：

```
if (fa * fc ≥ 0) then
  swap (fb, fc)
  swap (b,c)
  swap (fa, fb)
  swap (a,b)

else if (fb * fc ≥ 0) then
  swap (fa, fc)
  swap (a,c)
  swap (fa, fb)
  swap (a,b)
compute A
compute B
T1 = (a, b, A)
T2 = (b, B, A)
T3 = (A, B, c)
if (fc ≥ 0) then
  negative-subtree.add(T1)
  negative-subtree.add(T2)
  positive-subtree.add(T3)
else
  positive-subtree.add(T1)
  positive-subtree.add(T2)
  negative-subtree.add(T3)
```

上述代码利用了这样的事实：如果a和b是同号的则它们的乘积为正——所以得到第一个if语句。如果要进行顶点的交换，必须进行两次交换以保持顶点按逆时针顺序排列。注意如果一个顶点恰好在平面上，上述代码也是正确的，但是生成的三角形有一个面积为零。可以直接忽略这种情况，这并不是很危险，因为光栅化程序必须要处理屏幕上面积为零的三角形（例如侧向的三角形）。当然也可以增加一个判断，不把面积为零的三角形加入到BSP树中。最后再加一种特殊情
184 况，就是当fa、fb或者fc中的一个为零时，把原来的三角形分为两个三角形。

为了计算A和B，需要计算线段和隐式平面的交点。例如，连接a和c的线段的参数方程是

$$\boldsymbol{p}(t) = \boldsymbol{a} + t(\boldsymbol{c} - \boldsymbol{a})$$

把$\boldsymbol{p}(t)$代入平面$\boldsymbol{n}\cdot\boldsymbol{p} + D = 0$可以得到线段与平面的交点：

$$\boldsymbol{n} \cdot (\boldsymbol{a} + t(\boldsymbol{c} - \boldsymbol{a})) + D = 0$$

则t为

$$t = -\frac{\boldsymbol{n}\cdot\boldsymbol{a} + D}{\boldsymbol{n}\cdot(\boldsymbol{c} - \boldsymbol{a})}$$

把这个解称为t_A，就可以写出A的求解式：

$$\boldsymbol{A} = \boldsymbol{a} + t_A(\boldsymbol{c} - \boldsymbol{a})$$

同理可以得到B。

8.1.4 优化BSP树

与遍历BSP相比，我们并不太关心构造BSP树的效率，因为这只是一个预处理过程。遍历树的时间与树的结点个数成比例（树的平衡性并不重要）。每一个三角形（包括通过分裂得到的三角形）都对应一个结点。结点的个数依赖于三角形插入到树中的顺序。例如，如果图8-7中的T_1是根结点，树中将有两个结点，但是如果T_2是根结点将有更多的结点，因为这时T_1将被分开。

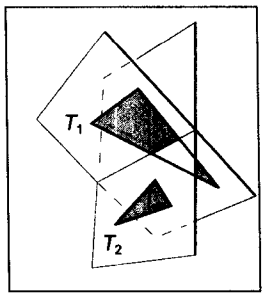


图8-7 用T_1作为BSP树的根结点，得到的树只有两个结点。用T_2作为根结点，需要把T_1分开，所以得到的树将会大一些

三角形插入树中的“最佳”顺序是很难确定的。对于N个三角形将有$N!$个可能的顺序。通常比较所有的顺序是不可能的。而只能从一个随机序列开始比较固定数目的几个序列，并把其中最好的作为最终的树。

上述分裂算法把一个三角形分裂为三个三角形。如果把三角形分裂为一个三角形和一个四边形效率将会更高一些。但如果输入模型只包含三角形，这样做可能没有意义，不过在支持任意多边形的应用中就很有用了。

185

8.2 z缓冲器

在几乎所有的视频游戏和图形PC上都能找到z缓冲器算法。当然它也是一个很有用的软件算法（Cook, Carpenter, & Catmull, 1987）。z缓冲器算法利用了下列事实：真正的问题是找出距离每一个像素中心最近的多边形；这比找出连续视域内所有多边形在深度方向正确的序列要简单得多。

8.2.1 z缓冲器算法

z缓冲器算法是非常简单的。在每一个像素处，存储一个z值，即当前像素中心到已经光栅化的三角形的最近距离。但对一个三角形进行光栅化时，正如3.6节中讨论的，每一个像素处的深度可以使用重心坐标对顶点的深度值进行插值得到。仅当z值到观察者更近时才把rgb值和z值写入光栅中。首先将z缓冲器初始化为一个最远值。现在假设z为正值。如果在具体实现中的使用了负的z值，就像上一章中那样，那么必须把下面的“小于”检测改成“大于”检测。我们使用下面的操作，而不是直接写rgb值：

```
function setpixel(int i, int j, rgb c, real z)
if (z < z-buffer(i, j)) then
    z-buffer(i, j) = z
    screen(i, j) = c
```

算法很简单，这就是为什么z缓冲器算法非常适合于硬件实现的原因。只要有用作“z缓冲器”的快速内存，计算是很简单的。

z缓冲器中的最后结果不依赖于三角形光栅化的顺序，如图8-8所示。但是有一个特殊情况就是当与某个像素对应的两个三角形的z深度相同的时候。在这种情况下，可以采用任何一种约定，最常用的是假设最后画的三角形的优先级最高。一个更好的想法就是不作任何的约定，那么优化程序可以重新对光栅化过程进行排序。

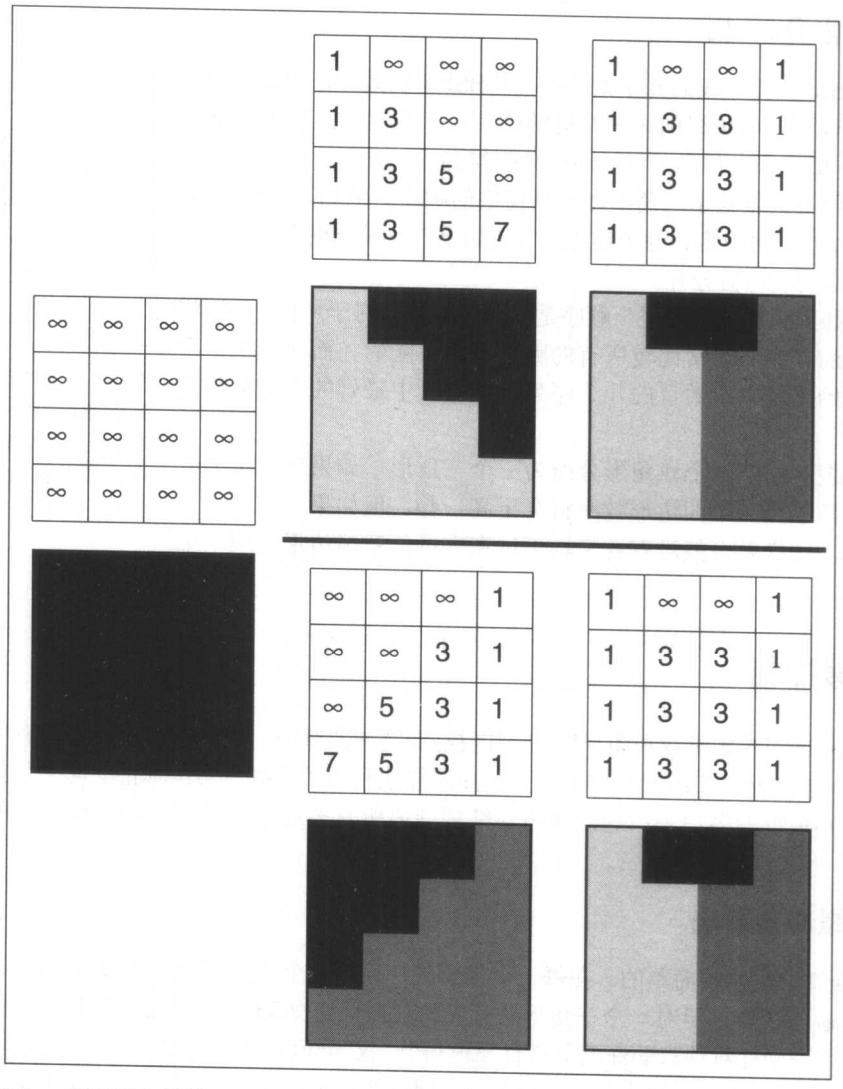


图8-8　z缓冲器算法按照两种可能的顺序对两个三角形进行光栅化。第一个三角形被完全光栅化。第二个三角形计算了每一个像素的深度，但是其中有三个像素在进行深度比较时没有被选中，所以这三个像素没有画出来。但不管怎样，最后的图像是相同的

8.2.2　整数 z 缓冲器

186 ~ 187

实践中，存储在缓冲器中的z值是非负的整数。这比使用真正的浮点数好，因为用作z缓冲器的快速内存比较贵，所以要尽量使它最小。

使用整数会引起一些精度问题。如果使用具有B个值$\{0, 1, \cdots, B-1\}$的整数范围，可以把0映射到最近的裁剪平面$z=n$，把$B-1$映射到最远的裁剪平面$z=f$。注意现在假设z、n和f都是正的。在取负数的情况下得到的结果将是相同的，但自变量的修正更容易。把每一个z值发到一个“桶”中，其深度值$\Delta z=(f-n)/B$。如果内存性能不是很高，我们就不会使用整数z缓冲器，所以尽量减小B是很有用的。

如果为存储z值分配了b位，那么$B=2^b$。我们需要足够的位数以保证任何位于另一个三角形前面的三角形都会映射到不同的深度区域中。

例如，如果要绘制的场景中的三角形之间至少相距一米，那么$\Delta z < 1$产生的图像应该不会有伪影。有两种方式可以使Δz更小：把n和f移得更近或者提高b。如果b是固定的（在某些API或特定的硬件平台上是这样的），那么调整n和f是唯一的选择。

当要生成透视图像时，必须要小心处理z缓冲器的精度问题。上面Δz的值是在透视除法后使用的。回忆一下在7.3节中，透视除法的结果是

$$z = n + f - \frac{fn}{z_w}$$

实际深度区域的宽度是和世界深度z_w相关的，而不是透视除法后的深度z。对上述公式两边同时求微分可以估计深度区域的大小：

$$\Delta z \approx \frac{fn\Delta z_w}{z_w^2}$$

深度区域的大小是随深度变化的。在世界空间中的区域大小是

$$\Delta z_w \approx \frac{z_w^2 \Delta z}{fn}$$

注意Δz的值和前面讨论的一样。最宽的深度区域对应的是$z' = f$，其中

$$\Delta z_w^{\max} \approx \frac{f\Delta z}{n}$$

注意$n = 0$的情况，如果我们不想丢失眼睛前面的所有对象，自然会选择$n = 0$，但是这将会得到一个无限大的区域——一种非常坏的情况。为了使$\Delta z_w^{\max}$尽量的小，要尽量减小f，增加n。所以小心地选择n和f是非常重要的。 188

常见问题

- **均匀距离z缓冲器是否比包括非线性透视矩阵的标准z缓冲器更好？**

这要根据不同情况判断。非线性的一个“特征”是z缓冲器在接近眼睛的地方分辨率高，然后随距离增大而减小。如果使用细节层次系统，几何模型在距离上是不精确的，z缓冲器的“不均匀性”正好适用于这种情况。

- **软件z缓冲器有用吗？**

是的。大部分使用三维图形的电影大量使用了Pixar开发的软件z缓冲器（Cook et al., 1987）。

习题

1. 给定N个三角形，要构建BSP树，最少要添加多少个三角形？最多能添加多少？
2. 假设要为飞行仿真设计一个整数z缓冲器，在这个场景中对象之间的距离至少是1 m，距离观察者最近为4 m，最远可能会达到100 km。为了保证不会出现视觉错误，z缓冲器需要多少位？假设视觉错误在靠近观察者时才出现，即距离小于100 m。在这种情况下需要多少位？ 189

第9章 表面明暗处理

为了使物体更有立体感，可以使用明暗处理，即表面用颜色来“画”。本章论述了最常用的启发式明暗处理方法。前两种方法，漫反射明暗处理和Phong明暗处理，是70年代提出来的，目前大部分的图形库中都提供。最后一种方法，艺术化明暗处理，使用美术惯例为物体指定颜色。它可以创建类似技术绘图的图像，这在很多应用中都是必要的。

9.1 漫反射明暗处理

世界上许多物体的表面看起来是“不光滑”的，这表明物体并不总是有光泽的。例如纸、未完成的木制品或者未抛光的石头等。当视点变化时，这类物体的颜色基本不会改变。例如，如果你盯着纸上的特定一点，然后开始移动，并保持凝视方向始终在这一点上，这一点的颜色将基本保持不变。这种不光滑的物体被认为具有朗伯物体特性。本节讨论如何实现这类物体的明暗处理。值得注意的是本章中所有的公式都是在世界坐标系中计算的，而不是在经过投影变换后弯曲的坐标系。否则法线之间的夹角就会发生变化，明暗处理就不对了。

191

9.1.1 朗伯明暗处理模型

朗伯物体遵守*朗伯余弦定律*（Lambert's cosine law），即物体表面的颜色c与表面法线和光线入射方向夹角的余弦成比例（Gouraud, 1971）：

$$c \propto \cos\theta$$

或其向量形式为

$$c \propto \boldsymbol{n} \cdot \boldsymbol{l}$$

式中$\boldsymbol{n}$和$\boldsymbol{l}$如图9-1所示。这样，表面的颜色将随着表面法线和光线方向夹角的余弦而变化。注意通常假设向量$\boldsymbol{l}$不依赖于物体的位置。这个假设等价于相对于物体的大小而言，光线距离物体非常远。这种“远距离”光线通常被称为*方向性光*（directional light），因为它的位置只用一个方向来指定。

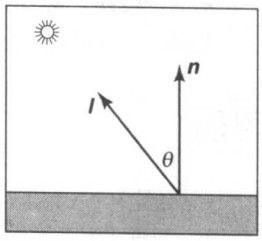


图9-1 朗伯定律的图示。$\boldsymbol{n}$和$\boldsymbol{l}$都是单位向量

改变光源的强度，或者改变表面的反射率都可以使物体表面变亮或者变暗。漫反射系数c_r是表面所有反射光线的一部分。对不同的颜色分量，它将是不同的。例如，如果表面反射的红色入射光远大于蓝色入射光，表面是红色的。如果假设表面颜色与表面反射的光线成比例，那么漫反射系数c_r（一种RGB颜色）也必须包括进来：

$$c \propto c_r \boldsymbol{n} \cdot \boldsymbol{l} \tag{9-1}$$

式（9-1）的右边是一个RGB颜色，其所有分量都在[0, 1]范围中。在保持RGB分量在[0, 1]范围内

的同时，我们希望增加光线强度的作用。这就需要增加RGB强度项c_l，它自身也在 [0, 1] 范围内：

$$c = c_r c_l \boldsymbol{n} \cdot \boldsymbol{l} \tag{9-2}$$

这个形式非常方便，但是因为点积可以是负的，如果c在[0, 1]范围外，它仍然能计算RGB分量。当表面指向远离光线的方向时，点积是负的，如图9-2所示。

可以在式（9-2）中添加“max”函数来检测这种情况：

$$c = c_r c_l \max(0, \boldsymbol{n} \cdot \boldsymbol{l}) \tag{9-3}$$

处理“负”光线的另外一种方式是使用绝对值：

$$c = c_r c_l |\boldsymbol{n} \cdot \boldsymbol{l}| \tag{9-4}$$

192

式（9-4）在物理上似乎是不合理的，实际上它与式（9-3）中在相反方向有两条光线的情况是相符的。所以它常被称为双面光照（two-sided lighting）（图9-3）。

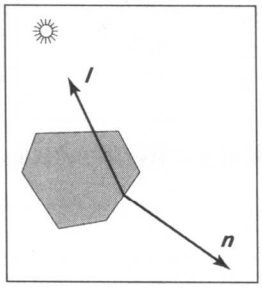


图9-2 当表面指向远离光线的方向时，它就得不到任何光线。这种情况可以通过检查$\boldsymbol{l}$和$\boldsymbol{n}$的点积是否为负来验证

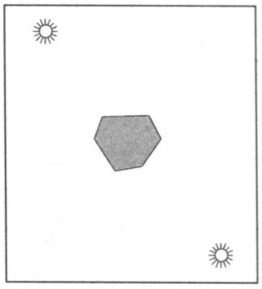


图9-3 使用式（9-4），双面光照公式与假设有两个同色的方向相反的光源是等价的

9.1.2 环境明暗处理

式（9-3）的明暗处理中有一个问题，即所有法向背向光源的点都将是黑色的。在实际生活中，光线被到处反射，有一些光线可能从所有的方向入射。另外还常常有天窗提供“环境”光照。要处理这个问题，一种方式是使用多个光源。常用的策略是在眼睛位置放一个较暗的光源，从而所有可见点都将接受到一些光线。另一种方式是使用式（9-4）中的双面光照。更常用的方法是添加一个环境项（Gouraud, 1971）。就是在式（9-3）中添加一个常数项：

$$c = c_r (c_a + c_l \max(0, \boldsymbol{n} \cdot \boldsymbol{l}))$$

直觉上，可以把环境颜色c_a看成场景中所有表面的平均颜色。如果要保证计算的RGB颜色在[0, 1]范围内，那么$c_a + c_l \leqslant (1, 1, 1)$。否则如果RGB值大于1，程序应该把该值置为1。

9.1.3 基于顶点的漫反射明暗处理

如果把式（9-1）应用到一个由三角形组成的物体上，它看起来是往往由很多小平面组成的。而这些三角形通常是对一个光滑表面的近似。为了避免出现小平面的现象，可以把表面的法线放在三角形的顶点上（Phong, 1975），然后在每一个顶点上应用式（9-3），使用顶点处的法向（见图9-4）。这将为三角形每一个顶点赋一种颜色，这里可以使用3.6节中的重心插值法进行插值。

在三角形顶点上进行明暗处理的一个问题是需要获得法线。许多模型都会提供法线。如果把光滑模型转换为镶嵌模型，那么当建立三角形时就可以创建法线。如果模型的顶点处没有法线，193
而又希望进行光滑的明暗处理，那么可以用多种启发式算法计算法线。最简单的方法是计算共享

一个顶点的所有三角形法线的平均值，然后在该顶点使用平均法线。平均法线不会自动成为单位长度向量，但因为要把它用于明暗处理，所以需要把它转换为单位向量。

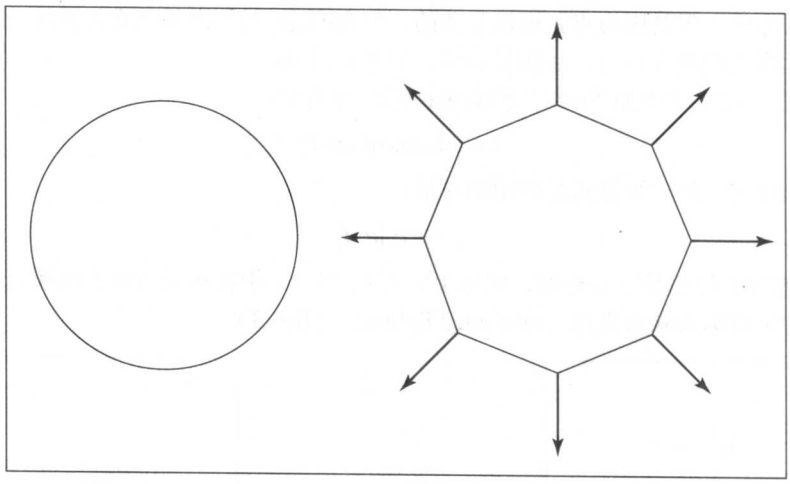


图9-4　用正八边形（右图）近似一个圆（左图）。顶点法线记录了原始曲线的表面法线

9.2　Phong 明暗处理

有些表面本质上是不光滑的，但是它们具有高光（highlight）。这样的表面如抛光的瓷砖地面、光泽涂料和白板等。高光在表面上会随着视点的移动而移动。这意味着必须要在公式中添加一个指向眼睛的单位向量**e**。如果仔细地观察高光，就会发现它们实际上就是反射光；有时这些反射光是模糊的。高光的颜色就是光线的颜色——表面的颜色看起来对它没有影响。这是因为反射发生在物体表面，穿过表面并携带物体颜色的光线被散射开了。

9.2.1　Phong 光照模型

我们希望在适当的位置添加一个同光源颜色相同的模糊“斑点”。
194 点的中心应该画在到眼睛的方向$\boldsymbol{e}$与自然反射方向$\boldsymbol{r}$相重合的地方，如图9-5所示。这里“重合”在数学上等价于“σ等于零”。我们希望高光有非零面积，这样当σ比较小时眼睛也能看到一些高光。

给定$\boldsymbol{r}$，希望能用这样一种启发式函数：即当$\boldsymbol{e}=\boldsymbol{r}$时最亮，当$\boldsymbol{e}$背离$\boldsymbol{r}$移动时逐渐减弱。显然它们之间的余弦函数是一个候选函数：

$$c = c_l(\boldsymbol{e}\cdot\boldsymbol{r})$$

使用这个公式会有两个问题。第一是点积可以是负的。这可以通过一个“if”语句来解决，当点积为负时，该语句将颜色设为零。更严重的问题
195 是用这个公式产生的高光比实际生活中看到的范围要大得多。

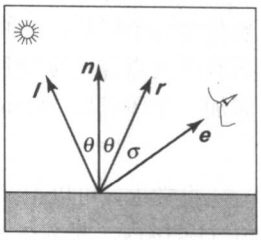


图9-5　Phong光照模型的图示。若σ较小，人眼将会看到一束高光

高光最大值的位置是正确的，颜色也是正确的，但是它太大了。在不减小最大值的情况下，可以通过提高幂次把它变窄一点。

$$c = c_l \max(0,\boldsymbol{e}\cdot\boldsymbol{r})^p \tag{9-5}$$

这里p称为Phong指数（Phong exponent），它是一个正实数（Phong，1975）。改变Phong指数后对高光的作用效果见图9-6。

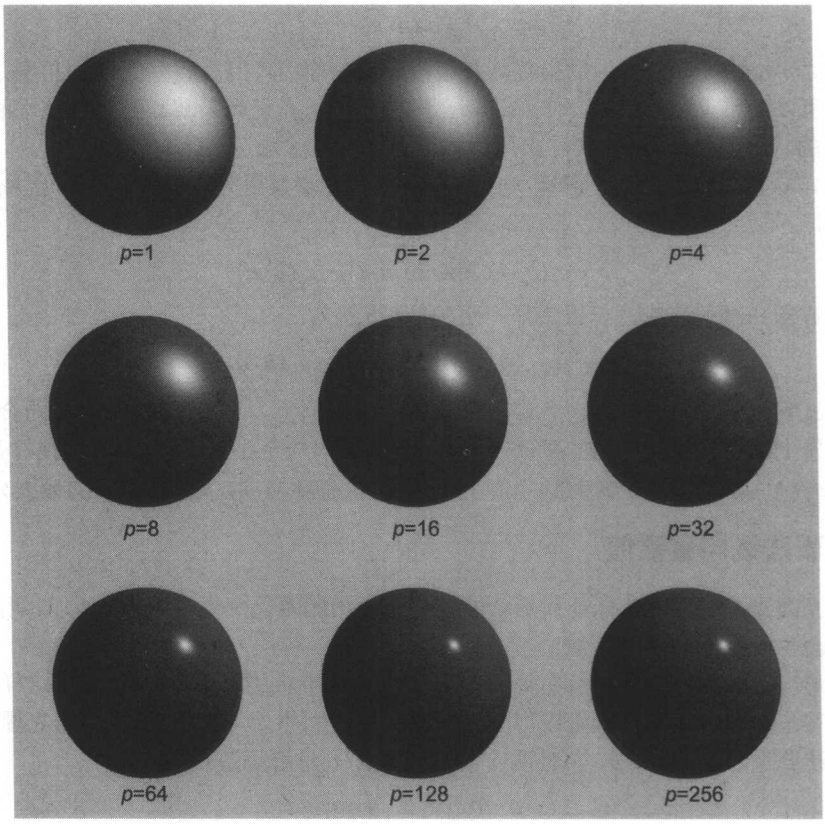


图9-6 Phong指数作用在高光特性上的效果。这里使用式（9-5）计算高光。另外还有漫反射分量，它使物体具有光泽，但呈现非金属外观。图片经Nate Robins授权使用（另见彩图IV）

为了实现式（9-5），首先需要计算单位向量$\boldsymbol{r}$。给定单位向量$\boldsymbol{l}$和$\boldsymbol{n}$，$\boldsymbol{r}$是向量$\boldsymbol{l}$关于$\boldsymbol{n}$的对称向量。图9-7表明该向量可以如下计算：

$$\boldsymbol{r} = -\boldsymbol{l} + 2(\boldsymbol{l} \cdot \boldsymbol{n})\boldsymbol{n} \tag{9-6}$$

这里点乘用来计算$\cos\theta$。

另一个基于式（9-5）的启发式模型不需要检查作为指数的底是否为负（Warn，1983）。它计算$\boldsymbol{l}$和$\boldsymbol{e}$之间的半角向量$\boldsymbol{h}$，而不是$\boldsymbol{r}$（图9-8）：

$$\boldsymbol{h} = \frac{\boldsymbol{e} + \boldsymbol{l}}{\|\boldsymbol{e} + \boldsymbol{l}\|}$$

当$\boldsymbol{h}$靠近$\boldsymbol{n}$时产生高光，即当$\cos\omega = \boldsymbol{h} \cdot \boldsymbol{n}$接近1时产生高光。这隐含着下面的规则：

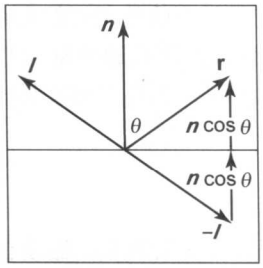


图9-7 计算向量$\boldsymbol{r}$的图示

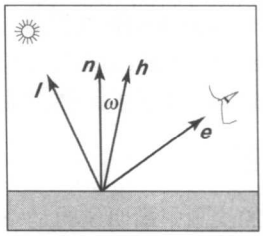


图9-8 单位向量$\boldsymbol{h}$是$\boldsymbol{l}$和$\boldsymbol{e}$之间的半角

$$c = c_l(\boldsymbol{h}\cdot\boldsymbol{n})^p \tag{9-7}$$

这里指数p将和式（9-5）中的指数具有相似的控制作用，但是$\boldsymbol{h}$和$\boldsymbol{n}$之间的角度是$\boldsymbol{e}$和$\boldsymbol{r}$之间角度的一半，所以细节上稍有差别。使用$\boldsymbol{n}$和$\boldsymbol{h}$之间余弦的优点是当眼睛和光线都在平面上面时，该角度总是正的。缺点是计算$\boldsymbol{h}$需要一个平方根运算和一个除法运算。

实际上，除了高光外，我们希望大部分材质都具有漫反射效果。那么可以合并式（9-3）和式（9-7）得

$$c = c_r(c_a + c_l\max(0,\boldsymbol{n}\cdot\boldsymbol{l})) + c_l(\boldsymbol{h}\cdot\boldsymbol{n})^p \tag{9-8}$$

如果允许用户减弱高光，可以增加一个控制项c_p：

$$c = c_r(c_a + c_l\max(0,\boldsymbol{n}\cdot\boldsymbol{l})) + c_l c_p(\boldsymbol{h}\cdot\boldsymbol{n})^p \tag{9-9}$$

控制项c_p是一个RGB颜色，它允许用户改变高光的颜色。这对于满足$c_p=c_r$的金属是很有用的，因为金属上的高光就是金属的颜色。另外，把c_p作为一个小于1的非彩色值往往是很有用的。
196 例如，设置$c_p = 1 - M$将保证光源的颜色值小于1并且没有环境项，这里M是c_r的最大分量。

9.2.2 表面法线向量插值

具有高光的光滑表面与相同几何形状的朗伯表面相比颜色变化得更快。所以，用法线向量进行明暗处理会生成令人厌烦的伪像。

在多边形上对法线插值，然后在每一个像素上应用Phong明暗处理可以减少这样的问题。这样不需要把三角形设得非常小就可以得到较好的图像。回想一下在第3章中，当光栅化一个三角形时，我们计算重心坐标(α，β，γ)对顶点颜色c_0、c_1、c_2进行插值：

$$c = \alpha c_0 + \beta c_1 + \gamma c_2 \tag{9-10}$$

也可以使用同样的公式对表面法线$\boldsymbol{n}_0$、$\boldsymbol{n}_1$、$\boldsymbol{n}_2$进行插值：

$$\boldsymbol{n} = \alpha\boldsymbol{n}_0 + \beta\boldsymbol{n}_1 + \gamma\boldsymbol{n}_2 \tag{9-11}$$

式（9-9）可以用来计算每一个像素处的法线$\boldsymbol{n}$。注意从式（9-11）得到的$\boldsymbol{n}$通常不是单位向量。在用它进行明暗处理前先转换为单位向量将会得到更好的视觉效果。这种法线插值称为Phong法线插值（Phong normal interpolation）（Phong，1975）。

9.3 艺术化明暗处理

朗伯明暗处理方法和Phong明暗处理方法是基于某些启发式函数的，用这些函数可以模拟物体在真实世界中的外观。艺术化明暗处理用来模仿人类艺术家的绘画（Yessios，1979；Dooley & Cohen，1990；Saito & Takahashi，1990；L. Williams，1991）。这样的明暗处理在很多应用中看起来都是有优势的。例如，汽车制造商户雇佣艺术家为汽车用户手册绘图。这显然比用更“真实”的照片要昂贵得多，可见当需要表达特定类型的信息时，使用艺术家的技巧可能更有优势。本小节介绍如何使微妙的阴影线图看起来像人工绘制的图。生成这样的图像通常称为非真实感绘制（non-photorealistic rendering），但是我们避免用这个术语，因为很多非真实感绘制技术是用来提高效率的，这和艺术家的习惯没关系。

9.3.1 线图

在人类的绘画中最明显的是轮廓线，而它在实际生活中其实是不存在的。假设有一个三角形
197 集合，其中某些三角形相互共享边。现在考虑共享一条边的两个三角形，如果其中一个朝向观察

者，另外一个背向观察者，那么应该把这条边作为轮廓线画出来。这种情况可以通过检测两个法线$\boldsymbol{n}_0$和$\boldsymbol{n}_1$判断：

$$\text{如果 } (\boldsymbol{e}\cdot\boldsymbol{n}_0)(\boldsymbol{e}\cdot\boldsymbol{n}_1)\leqslant 0 \text{ 则画轮廓线}$$

这里**e**是从边指向眼睛的向量，可以是边上或三角形上的任意一点。如果$f_i(\mathbf{p})=0$是两个三角形的隐式平面方程，检测可以写成：

$$\text{如果 } f_0(\boldsymbol{e})f_1(\boldsymbol{e})\leqslant 0 \text{ 则画轮廓线}$$

可能还需要画出多边形模型的所有可见边。为此可以用第8章中任意一个隐藏面消除方法画出背景颜色，然后用黑色画出每一个三角形的轮廓。实际上这样得到的将是模型的轮廓。不幸的是，如果多边形代表一个光滑表面，实际上大部分边都不需要画。然而也可能需求画出所有的折痕，即几何模型中的棱角。那么可以用下列启发式阈值来检测折痕：

$$\text{如果 } \boldsymbol{n}_0\cdot\boldsymbol{n}_1\leqslant \text{threshold 则画折痕}$$

把上述检测与轮廓线检测合在一起将得到很漂亮的线图。

9.3.2 冷暖明暗处理

当艺术家对线图进行明暗处理时，他们经常用低强度的明暗处理给出表面弯曲的感觉，并对物体赋予相应的颜色（Gooch，Shirley，& Cohen，1998）。朝一个方向的表面画成冷色，如蓝色，而朝相反方向的表面画成暖色，如橙色。通常这些颜色饱和度并不高，也不会很暗。这样黑色轮廓线可以表现得更好。总之，这就产生类似卡通的效果。设置“暖”光线$\boldsymbol{l}$的方向并用余弦函数对颜色进行调整就可以得到这样的效果，温暖常数k_w定义在[0, 1]上：

$$k_w=\frac{1+\boldsymbol{n}\cdot\boldsymbol{l}}{2}$$

则颜色c是冷色c_c和暖色c_w的线性组合：

$$c=k_wc_w+(1-k_w)c_c$$

198

很多c_w和c_c都可以得到合理的效果。下面的数据是一个很好的例子：

$$c_c=(0.4,0.4,0.7)$$
$$c_c=(0.8,0.6,0.6)$$

图9-9显示了传统Phong光照和这种艺术化明暗处理的比较。

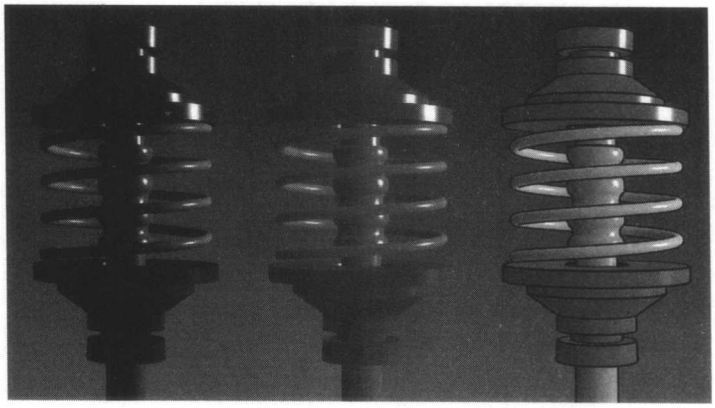


图9-9 左：Phong光照图像。中：没有轮廓线，冷暖明暗处理的效果不好。右：具有轮廓线的冷暖明暗处理。图片经Amy Gooch授权使用（另见彩图V）

常见问题

- **本章中所有的明暗处理看起来就像做了很多简化。对吗？**

是的。但是这些简化是精心设计的，在实际中证明是很有用的。将来可能会有更有根据的算
199 法，包括物理、心理和色调映射等。然而图像的质量是逐渐提高的。

- **我讨厌调用pow()。当执行Phong光照时有其他方式避免它吗？**

一种简单的方式只用自身是2的幂的项，如2, 4, 8, 16,…。实际上在大部分应用中这不是一个难以解决的问题。使用查找表也可以，但是这样做速度往往不会得到很大提高。

习题

1. 使用漫反射或者Phong明暗处理模拟月亮，效果很不好。原因是什么？
2. 使用漫反射或者Phong明暗处理模拟天鹅绒，效果很不好。原因是什么？
200 3. 为什么大部分塑料物体上的高光看起来是白色的，而在金属上的看起来是金色的？

第10章

光线跟踪

光线跟踪是一种生成真实感图像的方法；它在像素级别上确定图像中的可见面（Appel, 1968; Kay & Greenberg, 1979; Whitted, 1980）。跟z缓冲器算法和BSP树算法不同，光线跟踪是逐个像素地处理，而不是逐个图元地处理。对于屏幕空间中含有大型物体的场景，这使得光线跟踪相对较慢。然而它有很多优点，在进行批量绘制时甚至在一些交互式应用中都可以选择这种方法。

光线跟踪主要的好处是，相对而言它计算阴影和反射比较简单。另外，由于光线跟踪的低渐近时间复杂度，补偿了模型预处理的时间，所以光线跟踪适合于“浏览”巨大的模型（Snyder & Barr, 1987; Muuss, 1995; Parker, Martin, et al., 1999; Wald, Slusallek, Benthin, & Wagner, 2001）。

用传统的z缓冲器实现的三维交互程序中，能够用鼠标选择一个物体常常是很有用的。鼠标点在像素(i, j)处，被拾取的物体是通过这个像素可见的物体。如果光栅化过程中有物体标识缓冲器，这只需要在该缓冲器中查找一下像素(i, j)处的值。然而，如果没有提供这样的缓冲器，可以用“光线相交检测”进行蛮力几何计算确定哪个物体是可见的。所以，光线跟踪对只使用标准图形API的编程者也是有用的。 201

本章也讨论了分布式光线跟踪（distribution ray tracing）（Cook, Porter, & Carpenter, 1984），它从图像中的每个像素发出多条随机光线，同时解决反走样、软阴影、模糊反射和景深问题。

10.1 基本光线跟踪算法

光线跟踪最简单的用处就是生成与用z缓冲器和BSP树生成的相似的图像。本质上那些方法保证从每一个像素能够“看到”正确的物体，并根据物体的材质特性、表面法线和光学几何确定像素的颜色。

图10-1是光线跟踪的基本观察的图示，这跟前面第7章中见到的相同。该图形放在以眼睛位置e为原点的uvw坐标系中。光线跟踪的关键是确定观察平面$w = n$中与像素中心相对应的位置，如图10-2所示。一条“光线”，事实上就是一条有向三维直线，从e发出并指向该点。如图10-3所示，图中光线与两个三角形相交，但是只有第一个三角形被射中，返回T_2。 202

基本光线跟踪的程序的结构是

```
Compute u, v, w basis vectors
for each pixel do
  compute viewing ray
  find first object hit by ray and its surface normal n
  set pixel color to value based on material, light, and n
```

像素颜色可以用上一章中的明暗处理方程计算。

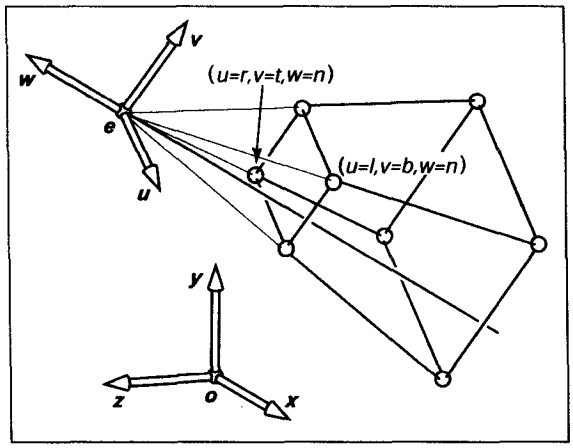


图10-1　我们看过去的三维窗口同第7章中的相同。窗口边界坐标很简单，它定义在以e为原点的uvw坐标系中

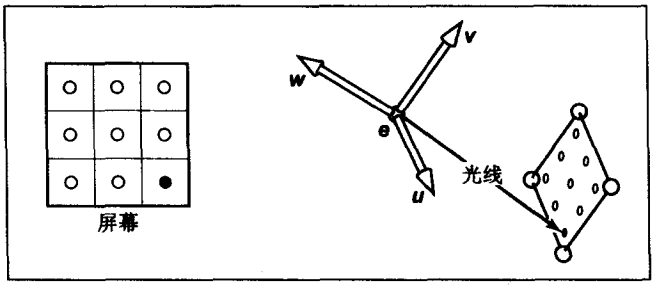


图10-2　屏幕上的采样点映射到三维窗口中一个类似的阵列中。观察光线发射到每一个这样的位置

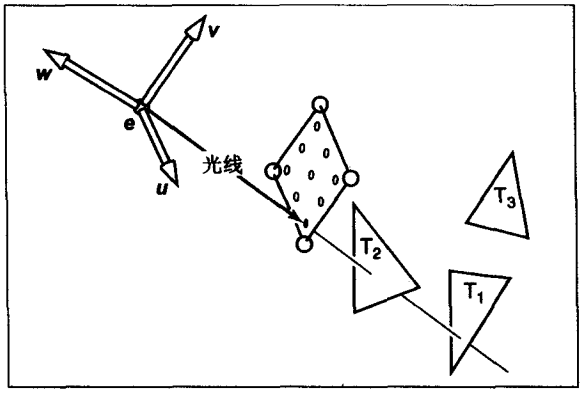


图10-3　光线被“跟踪”到场景中，第一个被射中的物体是从像素可见的物体。在上面的情况下返回三角形T_2

10.2　计算观察光线

首先需要确定光线的数学表达形式。一条光线实际上只是一个起点和一个传播方向；三维参
203 数直线是最理想的。如2.8.1节中讨论的，从眼睛e到屏幕上一点s的三维参数直线（图10-4）定义为

$$p(t) = e + t(s - e)$$

这应该解释为，“从e出发沿着向量$(s-e)$前进一定的距离t找到点p”。所以给定t，可以确定点p。注意$p(0)=e$，$p(1)=s$。同时要注意对于正的t，如果$t_1<t_2$，$p(t_1)$比$p(t_2)$距离眼睛更近。同样，如果$t<0$，则$p(t)$在眼睛后面。这些事实在搜索眼睛前面被光线射中的最近物体时是很有用的。注意这里重载了变量t，屏幕上面的v坐标也用了它。

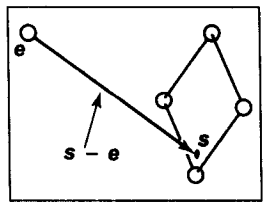


图10-4 从眼睛到屏幕上一点的光线

为了计算观察光线，需要知道e（这是给定的）和s。找到s看起来有些困难。实际上，利用投影变换的机制相对是比较简单的，即在观察中用到的对直线和三角形进行投影的机制。

首先，在以e为原点的uvw坐标系中找到s的坐标。对于屏幕上所有的点，$w_s=n$，如图10-2所示。通过一个窗口变换可以得到uv坐标，该变换把$[-0.5, n_x-0.5]\times[-0.5, n_y-0.5]$映射到$[l, r]\times[b, t]$上：

$$u_s = l+(r-l)\frac{i+0.5}{n_x}$$

$$v_s = b+(t-b)\frac{j+0.5}{n_y}$$

其中(i, j)是像素的索引。这样就得到s的uvw坐标。根据定义，可以转换为笛卡儿坐标：

$$\boldsymbol{s}=\boldsymbol{e}+u_s\boldsymbol{u}+v_s\boldsymbol{v}+w_s\boldsymbol{w} \tag{10-1}$$

还可以使用矩阵形式（式（6-8））：

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_e \\ 0 & 1 & 0 & y_e \\ 0 & 0 & 1 & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & x_w & 0 \\ y_u & y_v & y_w & 0 \\ z_u & z_v & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_s \\ v_s \\ w_s \\ 1 \end{bmatrix} \tag{10-2}$$

这即是式（10-1）的矩阵形式。另外还可以把它和窗口变换的矩阵形式结合在一起，但是这样做似乎没有必要，除非你更喜欢公式的矩阵形式。 204

10.3 光线与物体相交

给定一条光线$e+td$，希望能找到满足$t>0$并与之相交的第一个物体。后面将证明求解更一般的问题是很有用的，即在区间$[t_0, t_1]$上，针对$[0,\infty)$的观察光线找第一个交点的问题。在本节中将对球体和三角形进行求解，下一节中将讨论多个对象的情况。

10.3.1 光线与球相交

给定一条光线$e+td$和一个隐式表面$f(p)=0$，要求出它们相交的位置。当满足下列隐式方程时出现交点：

$$f(\boldsymbol{p}(t))=0$$

即

$$f(\boldsymbol{e}+t\boldsymbol{d})=0$$

圆心为$\mathbf{c}=(x_c, y_c, z_c)$半径为$R$的球可用隐式方程表示为

$$(x-x_c)^2+(y-y_c)^2+(z-z_c)^2-R^2=0$$

$$(\boldsymbol{p}-\boldsymbol{c})\cdot(\boldsymbol{p}-\boldsymbol{c})-R^2=0$$

任何满足上述方程的点都在球上。如果把光线$\boldsymbol{p}(t)=\boldsymbol{e}+t\boldsymbol{d}$上的点带入上述方程，可以得到光线与球相交点的$t$值：

$$(\boldsymbol{e}+t\boldsymbol{d}-\boldsymbol{c})\cdot(\boldsymbol{e}+t\boldsymbol{d}-\boldsymbol{c})-R^2=0$$

重新排列各项得

$$(\boldsymbol{d}\cdot\boldsymbol{d})t^2+2\boldsymbol{d}\cdot(\boldsymbol{e}-\boldsymbol{c})t+(\boldsymbol{e}-\boldsymbol{c})\cdot(\boldsymbol{e}-\boldsymbol{c})-R^2=0$$

这里，除了参数t外所有的都是已知的，所以这是传统的t的二次方程，即它有如下形式：

$$At^2+Bt+C=0$$

205 该方程的解在2.2节中讨论过。二次解中根号下的项B^2-4AC称为*判别式*，它说明有多少实数解。如果判别式为负，它的平方根为虚数，球和直线没有交点。如果判别式为正，则有两个解：一个解是光线进入球的位置，另一个是离开的位置。如果判别式为零，光线与球相切并只相交于一点。代入球的方程中，并消除公因子得

$$t=\frac{-\boldsymbol{d}\cdot(\boldsymbol{e}-\boldsymbol{c})\pm\sqrt{(\boldsymbol{d}\cdot(\boldsymbol{e}-\boldsymbol{c}))^2-(\boldsymbol{d}\cdot\boldsymbol{d})\,((\boldsymbol{e}-\boldsymbol{c})\cdot(\boldsymbol{e}-\boldsymbol{c})-R^2)}}{(\boldsymbol{d}\cdot\boldsymbol{d})}$$

实际实现中，在计算其他项之前应该首先检查判别式的值。如果球只是作为更复杂物体的包围盒，那么只需要判断是否射中它；检测判别式的符号。

如2.7.1节中讨论的，点$\boldsymbol{p}$处的法向量是通过梯度$\boldsymbol{n}=2(\boldsymbol{p}-\boldsymbol{c})$给定的。单位法线是$(\boldsymbol{p}-\boldsymbol{c})/R$。

10.3.2　光线与三角形相交

计算光线与三角形相交的算法有很多。这里将使用重心坐标的形式表示包含三角形的参数平面，因为它不像三角形顶点一样需要长期存储（Snyder & Barr, 1987）。

为了使光线与参数平面相交，建立下面的方程组，其所有的笛卡儿坐标都满足：

$$x_e+tx_d=f(u,v)$$
$$y_e+ty_d=g(u,v)$$
$$z_e+tz_d=h(u,v)$$

其中有三个方程，三个未知数(t、u和v)，所以可以通过数值计算得到未知数。幸运的话，还可以通过解析法得到。

如果这里的参数表面是一个参数平面，如2.11.2节讨论的那样，参数方程可以写成向量形式。如果三角形的顶点为$\boldsymbol{a}$、$\boldsymbol{b}$和$\boldsymbol{c}$，那么满足下列条件时将会相交：

206

$$\boldsymbol{e}+t\boldsymbol{d}=\boldsymbol{a}+\beta(\boldsymbol{b}-\boldsymbol{a})+\gamma(\boldsymbol{c}-\boldsymbol{a}) \qquad (10\text{-}3)$$

交点$\boldsymbol{p}$将位于$\boldsymbol{e}+t\boldsymbol{d}$，如图10-5所示。从2.11.2节还可以知道当且仅当$\beta>0$，$\gamma>0$，且$\beta+\gamma<1$时交点在三角形内。否则交点在该平面上但在三角形外。如果没有解，要么是三角形退化了，要么是光线平行于包含三角形的平面。

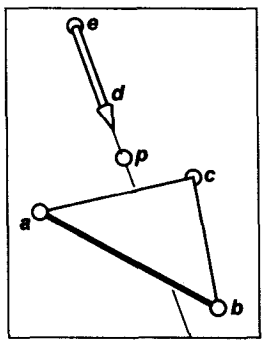


图10-5　光线与包含三角形的平面交在$\boldsymbol{p}$点

为了求公式（10-3）中的t、β和γ，把它的向量形式展开成三个坐标的三个方程：

$$x_e+tx_d=x_a+\beta(x_b-x_a)+\gamma(x_c-x_a)$$
$$y_e+ty_d=y_a+\beta(y_b-y_a)+\gamma(y_c-y_a)$$
$$z_e+tz_d=z_a+\beta(z_b-z_a)+\gamma(z_c-z_a)$$

它可以重写成一个标准线性方程：

$$\begin{bmatrix} x_a-x_b & x_a-x_c & x_d \\ y_a-y_b & y_a-y_c & y_d \\ z_a-z_b & z_a-z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a-x_e \\ y_a-y_e \\ z_a-z_e \end{bmatrix}$$

求解这样的3×3线性方程组的经典算法中最快的是克莱姆法则。它给出的解是

$$\beta = \frac{\begin{vmatrix} x_a-x_e & x_a-x_c & x_d \\ y_a-y_e & y_a-y_c & y_d \\ z_a-z_e & z_a-z_c & z_d \end{vmatrix}}{|A|}$$

$$\gamma = \frac{\begin{vmatrix} x_a-x_b & x_a-x_e & x_d \\ y_a-y_b & y_a-y_e & y_d \\ z_a-z_b & z_a-z_e & z_d \end{vmatrix}}{|A|}$$

$$t = \frac{\begin{vmatrix} x_a-x_b & x_a-x_c & x_a-x_e \\ y_a-y_b & y_a-y_c & y_a-y_e \\ z_a-z_b & z_a-z_c & z_a-z_e \end{vmatrix}}{|A|}$$

其中矩阵A是

$$A = \begin{bmatrix} x_a-x_b & x_a-x_c & x_d \\ y_a-y_b & y_a-y_c & y_d \\ z_a-z_b & z_a-z_c & z_d \end{bmatrix}$$

$|A|$表示A的行列式。3×3的行列式具有可以提取的公共子项。下列是用哑变量表示的线性方程组：

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix}$$

207

由克莱姆法则得到

$$\beta = \frac{j(ei-hf)+k(gf-\mathrm{d}i)+l(\mathrm{d}h-eg)}{M}$$

$$\gamma = \frac{i(ak-jb)+h(jc-al)+g(bl-kc)}{M}$$

$$t = -\frac{f(ak-jb)+e(jc-al)+\mathrm{d}(bl-kc)}{M}$$

其中

$$M = a(ei-hf)+b(gf-\mathrm{d}i)+c(\mathrm{d}h-eg)$$

可以重用某些数字以减少数字运算，例如“*ei*减*hf*”。

光线与三角形相交算法中我们需要的是线性解，它应该有提前终止的条件。所以函数看起来应该是这样的：

```
boolean raytri (ray r, vector3 a, vector3 b, vector3 c, interval[t0,t1])
compute t
if (t < t0) or (t > t1) then
```

```
  return false
compute γ
if (γ < 0) or (γ > 1) then
  return false
compute β
if (β < 0) or (β > 1 − γ) then
  return false
return true
```

10.3.3 光线与多边形相交

给定一个多边形，其m个顶点为$\boldsymbol{p}_1$到$\boldsymbol{p}_m$，表面法线为$\boldsymbol{n}$，首先用下列隐式方程计算光线$\boldsymbol{e} + t\boldsymbol{d}$和包含多边形的平面的交点：

$$(\boldsymbol{p} - \boldsymbol{p}_1) \cdot \boldsymbol{n} = 0$$

设$\boldsymbol{p} = \boldsymbol{e} + t\boldsymbol{d}$，求解$t$得

$$t = \frac{(\boldsymbol{p}_1 - \boldsymbol{e}) \cdot \boldsymbol{n}}{\boldsymbol{d} \cdot \boldsymbol{n}}$$

208

这就可以计算$\boldsymbol{p}$。如果$\boldsymbol{p}$在多边形内，那么光线射中多边形，否则射不中。

把点$\boldsymbol{p}$和多边形顶点都投影到xy平面上，就可以在该平面上判断$\boldsymbol{p}$是否在多边形内。最简单的方法是从$\boldsymbol{p}$点发出一条二维光线，然后数一下光线和多边形边界的交点个数（Sutherland et al., 1974; Glassner, 1989）。如果交点个数为奇数，则点在多边形内，否则在外面。因为一条光线要穿入多边形也必定穿出多边形，这样就生成交点对，而只有从内部发出的光线不会创建这样的点对，所以这个方法是正确的。为了使计算简单，可以让二维光线沿着x轴传播：

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \end{bmatrix} + s \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

对于$s \in (0, \infty)$，计算光线和边（如(x_1, y_1, x_2, y_2)）的交点是很简单的。

然而当多边形投影到xy平面上成为一条线时就出现问题了。为了避免这种情况，可以在xy、yz和zx平面中选一个最好的。如果允许用索引操作访问点，例如$\boldsymbol{p}(0) = x_p$，那么就可以用下面的程序实现：

```
if (abs(z_n) > abs(x_n)) and (abs(z_n) > abs(x_n)) then
  index0 = 0
  index1 = 1
else if (abs(y_n) > abs(x_n)) then
  index0 = 0
  index1 = 2
else
  index0 = 1
  index1 = 2
```

现在，所有计算中都可以使用$\boldsymbol{p}$(index0)而不是x_p。

10.4 光线跟踪程序

现在我们知道对给定的像素如何生成观察光线，如何找到与物体的交点。这很容易扩展成一个生成图像的程序，该程序和前面章节中z缓冲器或者BSP树的代码相似：

209

```
for each pixel do
```

```
compute viewing ray
if (ray hits an object with t ∈ [0,∞)) then
  Compute n
  Evaluate lighting equation and set pixel to that color
else
  set pixel color to background color
```

其中语句“if ray hits an object...”可以用一个函数实现，该函数检测交点是否在区间$t \in [t_0, t_1]$中：

```
hit = false
for each object o do
  if (object is hit at ray parameter t and t ∈ [t0,t1]) then
    hit = true
    hitobject = o
    t1 = t
return hit
```

在实际实现中，可能需要返回被射中物体的引用，或者至少是它的法线和材质特性。这常常是通过传递一个包含该信息的记录/结构来实现的。在面向对象的实现中，用名为*surface*等的类是一个好主意，通过它可以派生出*triangle*、*sphere*、*surface-list*等类。任何可能与光线相交的物体都在这个类下。光线跟踪程序可以得到模型中所有“表面”的引用，并且新类型的物体和高效的结构都可以透明地加到系统中。

面向对象的光线跟踪程序设计

就像前面提到的，光线跟踪程序中关键的类层次是组成模型的几何物体。它们应该是某些几何物体类的子类，并且它们应该支持*hit*函数（Kirk & Arvo, 1988）。为了避免使用“物体”这个术语引起混淆，通常用*surface*作为类名。有了这个类，就可以创建一个具有通用接口的光线跟踪程序，即它对模型的元素几乎不作假设，只使用球体进行调试。重要的是任何可能被光线“射中”的物体应该是这个类层次中的一部分，例如即使是一组表面的集合也应该视为表面类的一个子类。这包括一些高效的结构，如包围体层次；它们可能会被光线射中，所以也在这个类中。 210

例如，“抽象类”或者“基类”要定义hit函数，也要定义包围盒函数，后面将证明这是非常有用的：

```
class surface
virtual bool hit (ray e + td, real t0, real t1, hit-record rec)
virtual box bounding-box()
```

这里(t_0, t_1)是光线上射中物体部分的区间，rec是一个通过引用传递的记录；它包含这样的数据，如当函数hit返回真时的交点参数t。box的类型是一个三维“包围盒”，即两个点，定义了一个与坐标平面平行并包含该平面的立方体。例如，对于一个球体，该函数的实现如下：

```
box sphere :: bounding - box()
vector3 min = center - vector3(radius, radius, radius)
vector3 max = center + vector3(radius, radius, radius)
return box(min, max)
```

另一个有用的类是材质。它允许我们把材质特性抽象出来，以后可以透明地增加材质。把物体和材质相关联的简单方法是在surface类中加一个指向材质的指针，尽管这可能需要进行更多的编程。一个大问题是如何处理纹理；它们是作为材质类的一部分还是独立在材质外呢?第11章中将更多地讨论这个问题。

10.5　阴影

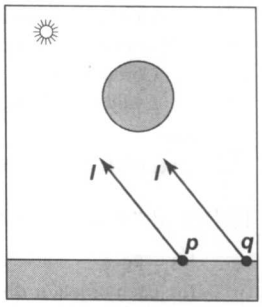


图10-6　点$\boldsymbol{p}$不在阴影中，而点$\boldsymbol{q}$在阴影中

一旦有了基本的光线跟踪程序，就可以很容易地添加阴影。回忆一下在第9章中，光线来自于某个方向$\boldsymbol{l}$。现在想象自己在一个正在进行明暗处理的平面上的一点$\boldsymbol{p}$上，如果沿着方向$\boldsymbol{l}$“观察”，看到的是一个物体，那么那该点在阴影中。如果看不到任何物体，光线就没有被遮住。

图10-6中，光线$\boldsymbol{p}+t\boldsymbol{l}$没有射中任何物体，所以不在阴影中。点$\boldsymbol{q}$在阴影中因为光线$\boldsymbol{q}+t\boldsymbol{l}$射中了一个物体。因为光线非常“遥远”，所以向量$\boldsymbol{l}$对两个点来讲是相同的。后面将放宽这个假设。为了与观察光线相区分，判断在阴影内或外的光线称为阴影光线（shadow ray）。

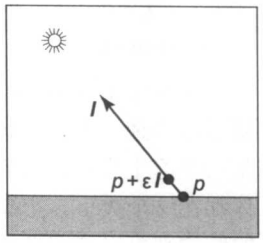


图10-7　通过检测以ε开始的区间，可以避免数值的不精确性引起光线射中$\boldsymbol{p}$所在平面的问题

为了计算阴影，需要在算法中添加一个if语句以确定点是否在阴影中。在简单的实现中，将在$t\in[0,\infty)$范围内检测阴影光线，但是由于数值的不精确性，它可能会与$\boldsymbol{p}$所在的表面相交。为了避免这个问题，常用的调整方法是在$t\in[\varepsilon,\infty)$范围内检测，其中ε是一个小的正常数（图10-7）。

211

如果利用公式（9-9）的Phong光照实现阴影光线，那么程序是：

```
function raycolor(ray e + td, real t0, real t1)
hit - record rec, srec
if (scene → hit(e + td, t0, t1, rec)) then
  p = e + rec.td
  color c = rec.cr  rec.ca
  if (not scene → hit(p + sl, ε, ∞, srec)) then
    vector3 h = normalized(normalized(l) + normalized(−d))
    c = c + rec.cr  cl max(0, rec.n · l) + cl rec.cp (h · rec.n)^rec.p
  return c
else
  return background-color
```

注意，在两种情况下都添加了环境颜色。如果有多个光源，可以发出一条阴影光线并分布计算每一个光源的漫反射/Phong项。在上面的程序中假设$\boldsymbol{d}$和$\boldsymbol{l}$都没有单位化。以后如果希望能简洁地增加实例，这对$\boldsymbol{d}$是非常重要的。

10.6　镜面反射

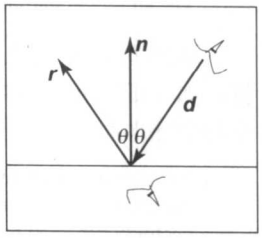


图10-8　当观察理想镜面时，沿方向$\boldsymbol{d}$的观察者将看到表面“下面”的观察者沿方向$\boldsymbol{r}$所看到的所有东西

在光线跟踪程序中添加镜面反射是很简单的。关键的事实如图10-8所示，图中观察者沿方向$\boldsymbol{e}$看到了从表面沿方向$\boldsymbol{r}$看到的物体。利用Phong光照反射公式（9-6）的变体，可以计算向量$\boldsymbol{r}$。因为在这种情况下向量$\boldsymbol{d}$指向表面，所以有一个符号变化：

$$\boldsymbol{r}=\boldsymbol{d}+2(\boldsymbol{d}\cdot\boldsymbol{n})\boldsymbol{n} \qquad (10\text{-}4)$$

在真实世界中，光线从表面上反射时会损失部分能量，这种损失对不同的颜色是不同的。例如，金子反射黄光的效率比蓝光高，所以它改变了物体反射的颜色。在函数*raycolor*中添加一个递归调用就可以实现了：

$$\text{color } c = c + c_s\,\text{raycolor}(\boldsymbol{p}+s\boldsymbol{r},\varepsilon,\infty)$$

其中c_s是反射的RGB颜色。我们必须保证在$s \in [\varepsilon, \infty)$范围内检测，其原因同处理阴影光线相同；我们不希望反射光线射中生成它的物体。 212

上述递归调用的问题是它可能永远都不会终止。例如，如果光线从一个房间中发出，它会永远反射下去。可以增加一个最大递归深度来修正这个问题。如果仅当c_s不为零（黑色）时生成一条反射光线，程序效率将会更高。

10.7 折射

另外一种镜面物体是电介质（dielectric）——一种折射光线的透明物体。钻石、玻璃、水和空气都是电介质。电介质也过滤光线：某些玻璃过滤的红色和绿色的光线比蓝色光线多，所以玻璃表现为绿的色调。当光线穿过折射率为n的媒介到折射率为n_t的媒介时，一些光线穿过了，并且发生了弯曲，如图10-9中所示的$n_t > n$。Snell法则是

$$n \sin\theta = n_t \sin\phi$$

通常计算两个向量之间夹角的正弦没有计算余弦方便，正如这里作的那样，余弦计算只是两个单位向量的点乘。使用三角恒定式$\sin^2\theta + \cos^2\theta = 1$，可以得出余弦折射关系：

$$\cos^2\phi = 1 - \frac{n^2(1-\cos^2\theta)}{n_t^2}$$

注意如果n和n_t对调了，那么 θ和 ϕ也应该对调，如图10-9右边所示。 213

为了把$\sin\phi$和$\cos\phi$转换为一个三维向量，在$\boldsymbol{n}$和$\boldsymbol{d}$所在平面内建立一个二维正交基。

从图10-10中可以看到$\boldsymbol{n}$和$\boldsymbol{b}$形成了折射平面的一个正交基。根据定义，可以用这组正交基描述$\boldsymbol{t}$：

$$\boldsymbol{t} = \sin\phi\boldsymbol{b} - \cos\phi\boldsymbol{n}$$

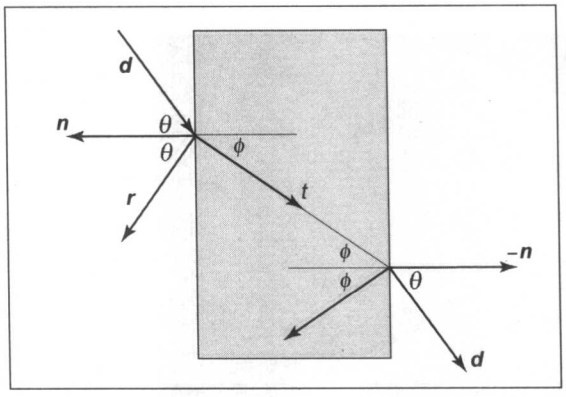


图10-9 Snell法则描述了角度ϕ如何依赖角度θ、物体和周围介质的反射率

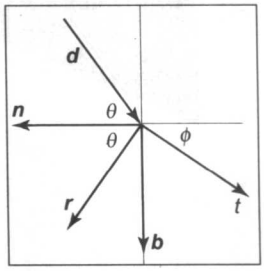


图10-10 向量$\boldsymbol{n}$和$\boldsymbol{b}$形成了一个正交基，它与传播向量$\boldsymbol{t}$平行

既然可以用同样的正交基描述$\boldsymbol{d}$，而$\boldsymbol{d}$是已知的，所以可以求解$\boldsymbol{b}$：

$$\boldsymbol{d} = \sin\theta\boldsymbol{b} - \cos\theta\boldsymbol{n}$$

$$\boldsymbol{b} = \frac{\boldsymbol{d} + \boldsymbol{n}\cos\theta}{\sin\theta}$$

这意味着可以利用已知的变量求解$\boldsymbol{t}$：

$$\boldsymbol{t} = \frac{n(\boldsymbol{d} + \boldsymbol{n}\cos\theta)}{n_t} - \boldsymbol{n}\cos\phi$$
$$= \frac{n(\boldsymbol{d} - \boldsymbol{n}(\boldsymbol{d}\cdot\boldsymbol{n}))}{n_t} - \boldsymbol{n}\sqrt{1 - \frac{n^2(1-(\boldsymbol{d}\cdot\boldsymbol{n})^2}{n_t^2}}$$

注意，无论n和n_t哪个大这个公式都是适用的。马上就有一个问题："如果平方根号下面的数字为负应该怎么办？"在这种情况下，没有折射光线，所有的能量都被反射了。这就是所谓的完全内部反射（total internal reflection），这就是玻璃物体有华丽外表的原因。

根据菲涅尔公式（Fresnel equations），电介质的折射率随着折射角度而变化。要实现接近菲涅尔公式的效果，一种好方法是使用Schlick近似（Schlick, 1994a）：

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

其中R_0是法线上的反射系数：

$$R_0 = \left(\frac{n_t - 1}{n_t + 1}\right)^2$$

注意上述$\cos\theta$项通常是指在空气中的角度（内部和外部相对法线较大的那个角）。

根据Beer定律，对于杂质分布均匀的物体，像典型的玻璃，透过光线的强度将被降低。当光线穿过介质时，它的强度按照公式$\mathrm{d}I = -CI\mathrm{d}x$降低，其中$\mathrm{d}x$是距离。所以$\mathrm{d}I/\mathrm{d}x = -CI$。可以求解
214 该方程并得到指数$I = k\exp(-Cx) + k'$。降低的程度可以用RGB降低常数a描述，即经过单位距离后降低的程度。代入边界条件，有$I(0) = I_0$，$I(1) = aI(0)$。前者意味着$I(x) = I_0\exp(-Cx)$，后者意味着$I_0 a = I_0\exp(-C)$，所以$-C = \ln(a)$。这样最终的公式是

$$I(s) = I(0)\mathrm{e}^{-\ln(a)s}$$

其中$I(s)$是距离表面s处的光线强度。实践中，可以用眼睛反向计算a，因为这样的数据很难找到。Beer定律的效果可以从图10-11中看到，图中玻璃表现为绿的色调。

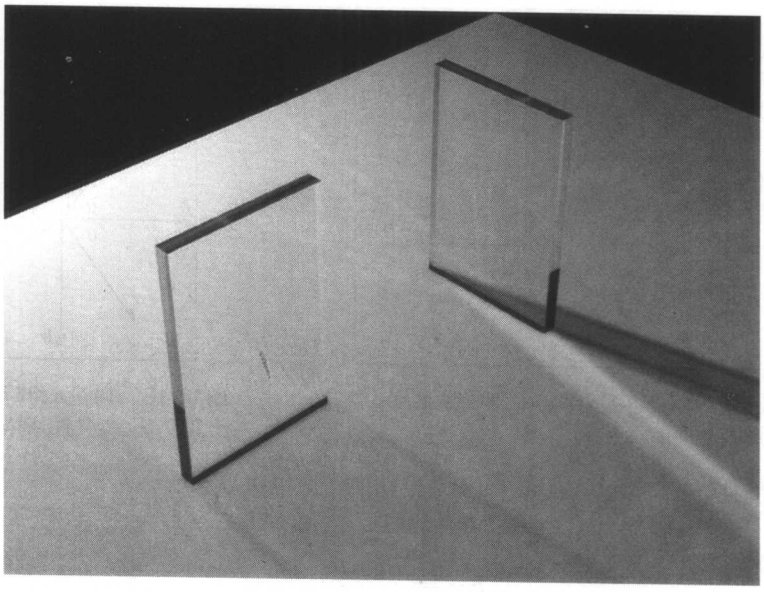


图10-11　玻璃的颜色受到完全内部反射和Beer定律的影响。光线透过和反射的数量由菲涅尔公式决定。地面上复杂的光照是通过第23章中的粒子跟踪方法计算的（另见彩图VI）

为了在代码中添加透明材质，需要有一种方法确定光线什么时候进入一个物体。最简单的方法是假设所有的物体都包含在空气中，其折射率接近为1.0，并且表面法向指向“外”（指向空气）。计算光线和满足上述假设的电介质的代码如下：

```
if (p is on a dielectric)) then
  r = reflect(d,n)
  if (d·n < 0) then
    refract(d,n,n,t)
  c = -d·n
  k_r = k_g = k_b = 1
else
  k_r = exp(-a_r t)
  k_g = exp(-a_g t)
  k_b = exp(-a_b t)
  if refract(d,-n,1/n,t) then
    c = t·n
else
  return k * color(p + tr)
R_0 = (n-1)^2/(n+1)^2
R = R_0 + (1 - R_0)(1 - c)^5
return k(R color(p + tr) + (1 - R) color(p + tt))
```

215

上述代码假设自然对数包含在常数(a_r, a_g, a_b)中。如果有完全的内部反射，那么函数*refract*返回假，否则它填写参数列表中最后一个参数。

10.8 实例化

光线跟踪的一个完美的特点是它允许自然地进行实例化（instancing）。实例化的基本思想是在物体被显示之前用变换矩阵对物体的所有点进行变形。例如，如果把一个单位圆（二维的）在x和y方向按照(2, 1)比例分别进行缩放，然后把它旋转45°，接着再沿x方向平移一个单位，结果是一个椭圆，其偏心率为2，长轴沿$x=-y$方向，中心为(0, 1)（图10-12）。使该实体变成一个“实例”的关键是要存储圆和合成变换矩阵。这样椭圆显式的构建过程成为将来绘制时的运算过程。

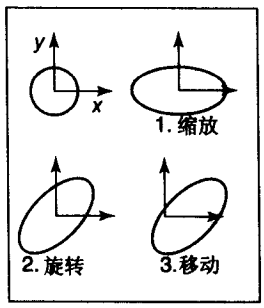


图10-12 一个实例，圆经过一系列三次变换后成为一个椭圆

光线跟踪中实例化的优点是可以选择求交运算的空间。如果基本物体由点集组成，$\boldsymbol{p}$是其中一个点，那么用矩阵$\boldsymbol{M}$对该点集变换后就形成了变换后的物体，点$\boldsymbol{p}$变换为$\boldsymbol{Mp}$。如果希望求光线$\boldsymbol{a}+t\boldsymbol{b}$与变换后的物体的交点，可以求*逆向变换后的光线*与未变换的物体的交点（图10-13）。在未变换的空间（如图10-13右侧）中计算有两个潜在的优势：

(1) 未变换物体的求交过程可能比较简单，如圆比椭圆简单；

216

(2) 许多变换后的物体可以和未变换的物体共享数据，这就可以减少存储，如在汽车的交通阻塞场景中，单个汽车只是几个基本模型（未变换的）的变形而已。

正如6.2.2节中讨论的，表面法线向量的变换是不同的。利用这种思想和图10-13中表示的概念，可以确定光线和用矩阵$\boldsymbol{M}$变换后的物体的交点。如果建立surface类型的实例类，需要建立一个hit函数：

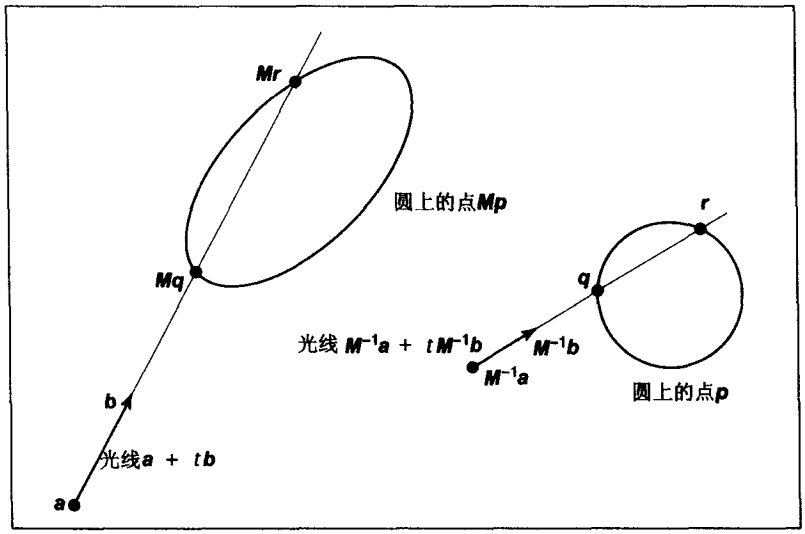


图10-13　两个空间中的光线相交问题只是相互之间简单的变换。用一个圆和矩阵M定义物体。在变换后的（世界）空间中的光线可以通过位置a和方向b指定

```
instrance::hit(ray a + tb, real t0, real t1, hit-record rec)
ray r' = M^-1 a + tM^-1 b
if (base-object → hit(r', t0, t1, rec)) then
  rec.n = (M^-1)^T rec.n
  return true
else
  return false
```

上面的程序中一个漂亮的地方是不需要改变参数rec.t，因为它在哪个空间中都是相同的。同
217 样注意不需要计算或保存矩阵M。

这就带来非常重要的一点：光线方向b不能变成单位长度的向量，否则上述机制就没用了。为此，没有必要把光线方向限制为单位向量。

为了进行实体纹理映射，可能需要记录交点的局部坐标，并把它返回在hit-record中。这就是参数rec.t中得到的光线r'。

为了实现类实例中的包围盒函数，只需要考虑基本物体包围盒的8个角，并用M对它们进行变换，就得到由这8个点构成的包围盒。这不一定能构造出最紧密的包围盒，但是它很通用并且易于实现。

10.9　次线性的光线与物体相交

在前面光线与物体相交的伪代码中，所有的物体循环进行相交检测。对于N个物体，就是$O(N)$的线性搜索，所以当N很大的时候速度会很慢。像大多数搜索程序一样，如果在预处理时创建一个有序的数据结构，光线与物体相交就可以用“分治法”在次线性时间内计算。这有很多种技术。

本节中详细讨论三种技术：包围体层次（Rubin & Whitted, 1980; Whitted, 1980; Goldsmith & Salmon, 1987）、均匀空间子分法（Cleary, Wyvill, Birtwistle & Vatti, 1983; Fujimoto, Tanaka, & Iwata, 1986; Amanatides & Woo, 1987）和二叉空间划分法（Glassner, 1984; Jansen, 1986; Havran, 2000）。
218 图10-14显示了前两种技术。

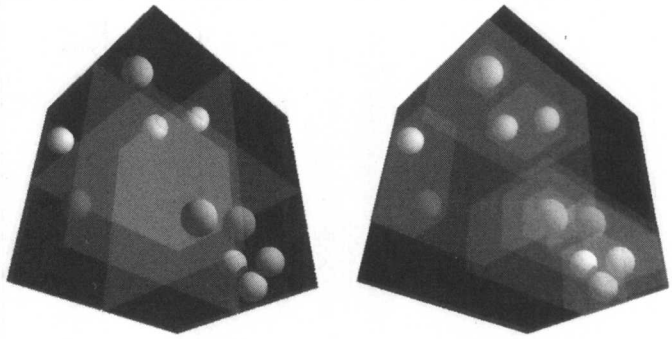


图10-14 左：均匀空间分割。右：自适应包围盒层次。图片经David DeMarle授权使用

10.9.1 包围盒

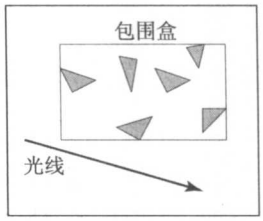


图10-15 只有在光线射中表面的包围盒时，才检测光线是否与表面相交

大部分相交检测加速机制中关键的操作是计算光线和包围盒的交（图10-15）。与传统的相交检测不同，它不需要知道光线与包围盒相交的位置，只需要知道它们是否相交。

为了构造光线与包围盒相交算法，首先考虑一条二维光线，其方向向量的x和y分量都为正数。以后可以把它扩展成任意的三维光线。二维包围盒定义为两条水平直线和两条垂直直线：

$$x = x_{\min}$$
$$x = x_{\max}$$
$$y = y_{\min}$$
$$y = y_{\max}$$

这些直线包围的点可以用区间表示法表示：

$$(x, y) \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$$

如图10-16，相交检测可以用这些区间来表达。首先计算光线射中直线$x = x_{\min}$时的参数：

$$t_{\text{xmin}} = \frac{x_{\min} - x_e}{x_d}$$

同理可以计算t_{xmax}、t_{ymin}和t_{ymax}。光线射中包围盒，当且仅当区间$[t_{\text{xmin}}, t_{\text{xmax}}]$和$[t_{\text{ymin}}, t_{\text{ymax}}]$重合，即它们的交为非空。算法的伪代码如下：

$t_{\text{xmin}} = (x_{\min} - x_e)/x_d$
$t_{\text{xmax}} = (x_{\max} - x_e)/x_d$
$t_{\text{ymin}} = (y_{\min} - y_e)/y_d$
$t_{\text{ymax}} = (y_{\max} - y_e)/y_d$
if $(t_{\text{xmin}} > t_{\text{ymax}})$ or $(t_{\text{ymin}} > t_{\text{xmax}})$ **then**
  return false
else
  return true

219

if语句看起来可能不明显。为了弄明白它的逻辑，注意如果第一个区间完全在第二个区间的左侧或者右侧，将不会产生重叠。

我们要处理的第一件事是x_d或者y_d为负的情况。如果x_d为负，光线将在射中$x_{\min}$之前射中$x_{\max}$。所以计算t_{xmin}和t_{xmax}的代码扩展为

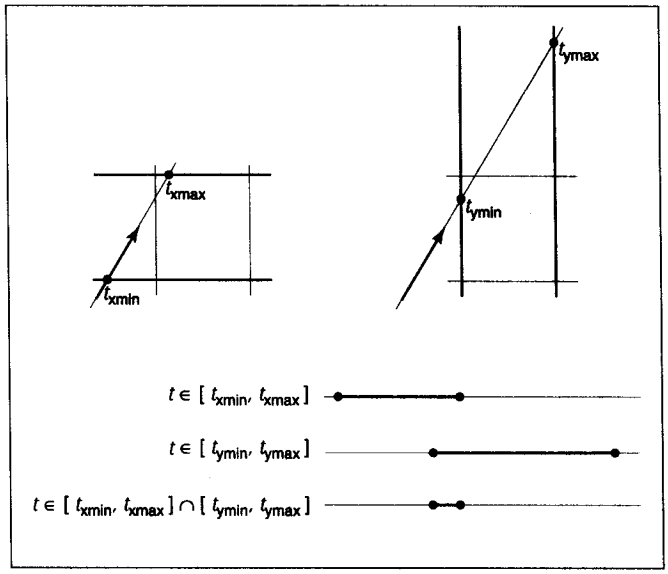


图10-16　对于某些在参数空间$t \in [t_{xmin}, t_{xmax}]$中的区间，光线将在区间$x \in [x_{min}, x_{max}]$中。同样的区间也存在于$y$区间中。如果光线同时在包围盒的$x$区间和$y$区间中，即两个一维区间的交为非空，那么光线与包围盒相交

if $(x_d \geqslant 0)$ **then**
$t_{xmin} = (x_{min} - x_e)/x_d$
$t_{xmax} = (x_{max} - x_e)/x_d$
else
$t_{xmin} = (x_{max} - x_e)/x_d$
$t_{xmax} = (x_{min} - x_e)/x_d$

对y的情况也必须作相似的代码扩展。一个严重的问题是水平和垂直光线x_d和y_d分别为零的情
220 况。这可能会引起除零问题。然而，在直接处理这个问题之前，看看IEEE浮点运算是否能很漂亮地处理这种情况。回想1.6节中除以零的规则是：对于任意正实数a，

$$+a/0 = +\infty$$
$$-a/0 = -\infty$$

考虑垂直光线$x_d = 0$且$y_d > 0$的情况。可以算出

$$t_{xmin} = \frac{x_{min} - x_e}{0}$$
$$t_{xmax} = \frac{x_{max} - x_e}{0}$$

可能有三种情况：

(1) $x_e \leqslant x_{min}$（未射中）
(2) $x_{min} < x_e < x_{max}$（射中）
(3) $x_{max} \leqslant x_e$（未射中）

对于第一种情况有

$$t_{xmin} = \frac{\text{正数}}{0}$$
$$t_{xmax} = \frac{\text{正数}}{0}$$

得到区间 $(t_{xmin}, t_{xmax}) = (\infty, \infty)$。该区间将不会同任何区间重叠，所以正如期望的那样没有射中。对于第二种情况，有

$$t_{xmin} = \frac{\text{负数}}{0}$$

$$t_{xmax} = \frac{\text{正数}}{0}$$

得到区间 $(t_{xmin}, t_{xmax}) = (-\infty, \infty)$，它将与所有的区间重叠，所以正如期望的那样被射中。第三种情况得到区间 $(-\infty, -\infty)$，正如期望的那样它也没有射中。因为在这些情况下的结果和期望的一样，所以不必进行特别的检测。像往常一样，IEEE浮点规范总是能为我们解决问题的。然而，这个方法还有一个问题。 221

考虑下列代码：

```
if (x_d ≥ 0) then
  t_min = (x_min − x_e)/x_d
  t_max = (x_max − x_e)/x_d
else
  t_min = (x_max − x_e)/x_d
  t_max = (x_min − x_e)/x_d
```

当 $x_d = -0$ 时代码中断了。通过检查x_d的倒数就可以克服这个问题（A. Williams, Barrus, Morley, & Shirley, 2005）：

```
a = 1/x_d
if (a ≥ 0) then
  t_min = a(x_min − x_e)
  t_max = a(x_max − x_e)
else
  t_min = a(x_max − x_e)
  t_max = a(x_min − x_e)
```

10.9.2 层次包围盒

层次包围盒的基本思想可以看作是通过普通的策略把一个轴向三维包围盒放在所有物体外面，如图10-17所示。判断光线与包围盒相交的计算实际上将比蛮力搜索计算的代价更大，因为与包围盒的相交检测是受限制的。然而，判断光线不与包围盒相交的检测代价要比蛮力搜索小。通过划分包围盒中的物体集合，并在每一个划分外面建立包围盒，可以建立包围盒的层次结构，如图10-18所示。图10-19所示的分层结构的数据结构可能是一棵树，大的包围盒是根结点，两个较小的包围盒是左右两个子树。这样每一个结点各自指向一个包含三个三角形的列表。检测光线和这棵特别编码树是否相交的代码如下：

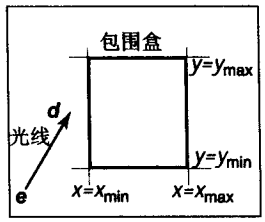


图10-17 二维光线*e+td*与一个二维包围盒进行相交检测

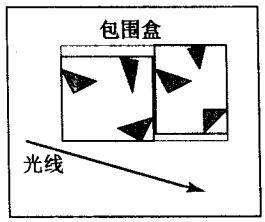


图10-18 包围盒可以嵌套，可以在模型的子集外面建立包围盒 222

```
if (ray hits root box) then
  if (ray hits left subtree box) then
    check three triangles for intersection
  if (ray intersects right subtree box) then
    check other three triangles for intersection
  if (an intersections returned from each subtree) then
    return the closest of the two hits
  else if (a intersection is returned from exactly one subtree) then
```

```
        return that intersection
      else
        return false
    else
      return false
```

观察上述算法可以看出，两个子树之间没有几何顺序，并且光线可能同时射中两个子树。事实上，两个子树没有理由不会重叠。

这个数据层次的关键是，一个包围盒保证包含层次结构中所有在它下方的物体，但是它并不能保证包含所有在空间上相互重叠的物体，如图10-19所示。这使得几何搜索比在严格排序的一维数据上传统的二分搜索复杂得多。读者可能注意到它本身有多种优化方式。这里等有了完整的层次算法后再进行优化。

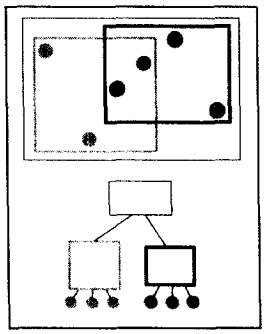


图10-19　灰色的包围盒是树的一个结点，它指向三个灰色的球，黑色的包围盒指向三个黑色的球。注意并不是所有被包围盒包含的球都一定包含在相应的树结点中

如果把树限制为二叉树，并要求树中的每个结点都有包围盒，那么遍历的代码可以很自然地扩展。进一步，假设所有结点要么是树的叶结点，它包含一个图元，要么包含一或两个子树。

bvh-node类应该是surface类的一种，所以应该实现函数surface::hit。它包含的数据应该很简单：

```
class bvh-node subclass of surface
virtual bool hit(ray e + td, real t0, real t1, hit-record rec)
virtual box bounding-box()
surface-pointer left
surface-pointer right
box bbox
```

遍历的程序可以采用面向对象的方式递归调用：

```
bool bvh-node::hit(ray a + tb, real t0, real t1, hit-record rec)
if (bbox.hitbox(a + tb, t0, t1)) then
  hit-record lrec, rrec
  left-hit = (left ≠ NULL) and (left → hit(a + tb, t0, t1, lrec))
  right-hit = (right ≠ NULL) and (right → hit(a + tb, t0, t1, rrec))
  if (left-hit and right-hit) then
    if (lrec.t < rrec.t) then
      rec = lrec
    else
      rec = rrec
    return true
  else if (left-hit) then
    rec = lrec
    return true
  else if (right-hit) then
    rec = rrec
    return true
  else
    return false
else
  return false
```


注意因为left和right指向surface而没有明确指向bvh-node，所以虚函数可以分辨内部结点和叶

结点；它将会调用适当的hit函数。如果能正确地建立这棵树，就不需要检测左子树是否为空（NULL）。如果还希望不检测右子树是否为空（NULL），可以把空的右指针替换为一个指向左子树的冗余指针。这将会对左子树进行两次检测，但会排除在整个树中的检测。是否值得这样做取决于构建树的详细过程。

构造包围体分层结构的方式有很多。为了便于处理，假设树是二叉的、粗平衡的，并且兄弟子树对应的包围盒重叠少。一种启发式方法是在把所有表面分割为两个列表之前，把它们沿着一个坐标轴进行排序。如果用整数定义坐标轴，$x=0$，$y=1$，$z=2$，有

```
bvh-node::bvh-node(object-array A, int AXIS)
N = A.length
if (N = 1) then
  left = A[0]
  right = NULL
  bbox = bounding-box(A[0])
else if (N=2) then
  left-node = A[0]
  right-node = A[1]
  bbox = combine(bounding-box(A[0]), bounding-box(A[1]))
else
  sort A by the object center along AXIS
  left = new bvh-node(A[0..N/2 – 1], (AXIS + 1) mod 3)
  right = new bvh-node(A[N/2..N– 1], (AXIS + 1) mod 3)
  bbox = combine(left-node → bbox, right-node → bbox)
```

224

如果每次都能仔细地选择AXIS，可以提高树的质量。选择坐标轴的一种方式是使两个子树包围盒的体积之和最小。与围绕坐标轴的旋转相比，这种变化将不会改变由均衡分布的小物体构成的场景，但是它可能对特性不好的场景有显著的帮助。如果只进行划分而不是全排序，代码的效率会更高。

另一种（可能是更好的）创建树的方式是使子树包含同样大的空间，而不是相同数量的物体。为此我们基于空间对列表进行划分：

```
bvh-node::bvh-node(object-array A, int AXIS)
N=A.length
if (N = 1) then
  left = A[0]
  right = NULL
  bbox = bounding-box(A[0])
else if (N = 2) then
  left = A[0]
  right = A[1]
  bbox = combine(bounding-box(A[0]), bounding-box(A[1]))
else
  find the midpoint m of the bounding box of A along AXIS
  partition A into lists with lengths k and (N-k) surrounding m
  left = new node(A[0..k], (AXIS + 1) mod 3)
  right = new node(A[k+1..N – 1], (AXIS + 1) mod 3)
  bbox = combine(left-node → bbox, right-node → bbox)
```

尽管这样得到的是一棵不平衡的树，但它易于遍历空的空间，并且因为划分比排序代价小，所以它的代价也比较小。

10.9.3　均匀空间子分法

225 另一种减少相交检测的策略是划分空间。这与前面的层次包围体有本质的区别：

- 在层次包围体中，每一个物体隶属于一或两个兄弟结点，而空间中的一点可能同时在两个兄弟结点中。
- 在空间子分法中，空间中的一点只隶属于一个结点，而物体可能隶属于多个结点。

场景被划分到轴向立方体中。所有盒子的大小相同，当然没有必要都是立方体。如图10-20所示，
226 光线穿过这些立方体，当它射中一个物体时，遍历结束。

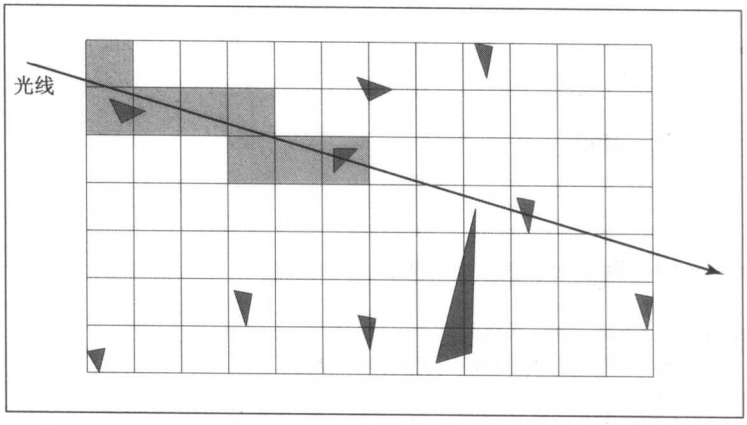


图10-20　在均匀空间子分法中，光线穿过一系列单元格，直到射中其中某个单元格中的一个物体。在本例中，只检测了灰色单元格中的物体

栅格本身应该是surface的子类，并且应该作为一个指向surface的三维数组。对于空的单元格指针为空（NULL）。对于包含一个物体的单元格，指针指向该物体。对于包含多个物体的单元格，指针指向一个队列、另一个栅格，或者其他数据结构，如包围体层次。

遍历按照递增的方式进行。如图10-21所示，光线射中每一组平行平面的模式是很规则的。为了解遍历是如何进行的，首先考虑二维的情况，其中光线方向的x和y分量都为正，并且光线的起点在栅格外面。假设栅格包含在点$(x_{\min}, y_{\min})$和$(x_{\max}, y_{\max})$中。网格有$n_x \times n_y$个单元格。

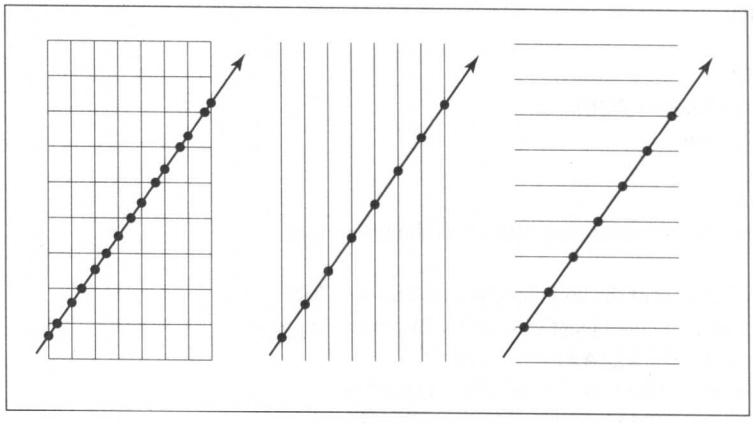


图10-21　尽管单元格上射点的样式看起来不规则（左），但是在一组平行平面上的射点是非常均匀的

第一件事是找到光线$\boldsymbol{e}+t\boldsymbol{d}$射中的第一个单元格的索引$(i, j)$。然后需要按照恰当的顺序遍历所有单元格。算法的关键部分是找到初始单元格(i, j)并确定对x还是y递增（图10-22）。注意当对一

个单元格中的物体进行相交检测时，要把t的范围限制在该单元格中（图10-23）。在算法的大部分实现中都采用三维数组作为“到surface的指针”。为了提高遍历的局部性，数组可以分成方格状，像13.4节讨论的那样。

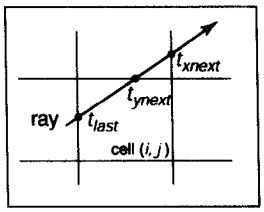


图10-22 为了确定向右还是向上步进，我们跟踪与单元格下一个垂直或者水平边界的交点

10.9.4 二叉空间划分法

我们也可以用层次数据结构划分空间，如二叉空间划分树（BSP树）。这与第8章画家算法中用的BSP树相似，但是对于简单的光线相交，通常用轴向裁剪平面。在这个结构中，一个结点可能包含一个裁剪平面和左右子树。这些子树包含位于裁剪平面一侧的所有物体。穿过裁剪平面的物体将在两个子树中。如果假设裁剪平面平行于yz平面，且位于$x = D$，则结点类为

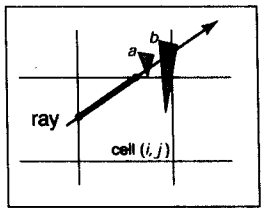


图10-23 只考虑在单元格内射点。否则上述情况将使我们认为射中的物体是b而不是a

```
class bsp-node subclass of surface
virtual bool hit (ray e + td, real t0, real t1, hit-record rec)
virtual box bounding-box()
surface-pointer left
surface -pointer right
real D
```

以后可以把它扩展到y和z裁剪平面上。那么相交的代码可以按照面向对象的方式被递归调用。代码考虑了四种情况，如图10-24所示。设光线的原点是参数为t_0的点：

$$\boldsymbol{p} = \boldsymbol{a} + t_0\boldsymbol{b}$$

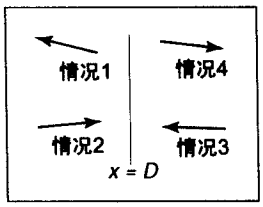


图10-24 光线如何与裁剪平面 $x = D$ 相关的四种情况

227

这4种情况是：

(1) 光线只和左子树相交，不需要检测它是否和裁剪平面相交。这发生在$x_p < D$且$x_b < 0$的情况下。

(2) 光线和左子树进行检测，如果没有射中，那么再和右子树进行检测。我们需要获得光线在$x = D$处的参数，这样可以保证只在子树范围内进行相交检测。这发生在$x_p < D$且$x_b > 0$的情况下。

(3) 这种情况与情况1相似，它发生在$x_p > D$且$x_b > 0$的情况下。

(4) 这种情况与情况2相似，它发生在$x_p > D$且$x_b < 0$的情况下。

按照顺序处理上述情况遍历代码如下：

```
bool bsp-node::hit(ray a + tb, real t0, real t1, hit-record rec)
x_p = x_a + t0 x_b
if (x_p < D) then
  if (x_b < 0) then
    return (left ≠ NULL) and (left → hit(a + tb,t0,t1,rec))
  t = (D − x_a)/x_b
  if (t > t1) then
    return (left ≠ NULL) and (left → hit(a + tb,t0,t1,rec))
  if (left ≠ NULL) and (left → hit(a + tb,t0,t1,rec)) then
    return true
  return (right ≠ NULL) and (right → hit(a + tb,t,t1,rec))
else
  analogous code for cases 3 and 4
```

这是一段非常简洁的代码。但是为了启动该程序，需要使光线射中某些含有包围盒的根物体，这样就可以初始化遍历过程，t_0和t_1。现在要解决的一个问题是裁剪平面可能会沿着任意的坐标

轴。可以在bsp-node类中添加一个整数索引axis作为指示。如果允许对点使用索引操作，只需要对上述代码进行简单的修改，例如，

$$x_p = x_a + t_0 x_b$$

将变成

228

$$u_p = a[\text{axis}] + t_0 b[\text{axis}]$$

这将产生某些附加的数组索引，但是不会产生更多的分支。

尽管处理单个bsp-node比处理bvh-node快，事实是一个平面可能存在于多个子树中，这意味着可能有多个结点，并且可能用更多的内存。树构建的质量取决于哪个更快。构造BSP树和构造BVH树是相似的。我们可以选择坐标轴循环进行划分，也可以每次分成两半，还可以尝试更复杂的划分方法。

10.10　构造实体几何

光线跟踪的一个好处是，任何几何元素，只要能够计算与三维直线的交点，都可以无缝地添加到光线跟踪程序中。在光线跟踪程序中添加构造实体几何（constructive solid geometry，CSG）也是很简单的（Roth, 1982）。CSG的基本思想是用集合运算合成实体形状。这些基本运算如图10-25所示。运算可以看作是集合运算。例如，可以把C看作圆中所有点的集合，把S看作正方形中所有点的集合。相交运算$C \cap S$是所有既是C的元素又是S的元素的点的集合。其他的运算也是类似的。

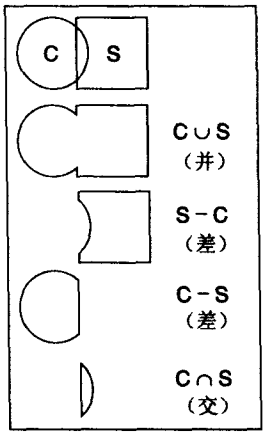


图10-25　在二维的圆和正方形上基本的CSG运算

尽管可以直接对模型进行CSG处理，但是如果要的只是一个图像，并不需要显式地对模型进行转换。相反，当光线与模型求交时，可以直接对光线进行集合运算。为了处理更自然，我们找出模型和光线的所有交点，而不是最近的一个。例如，光线$\boldsymbol{a} + t\boldsymbol{b}$可能在$t = 1$和$t = 2$处射中一个圆。在CSG环境中，对于$t \in [1, 2]$光线在圆内。可以计算出所有表面的“内部区间”，并在区间上进行集合运算（回想一下2.1.2节）。图10-26说明了这个过程，射中的区间处理后表明有两个区间分别在不同的物体内。对于$t > 0$的第一个射点是光线实际的交点。

实践中，CSG相交检测过程必须维护一个区间列表。当检测到第一个射点时，就把材质特性和表面法线与该射点关联起来。另外，必须要注意精度的问题，因为没有什么能防止用户让两个相邻的物体相交。通过删除所有宽度低于特定限度的区间可以提高程序的稳健性。

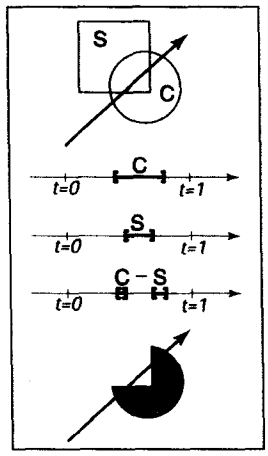


图10-26　处理后的区间表明光线如何射中了一个合成物体

10.11　分布式光线跟踪

对于某些应用而言，光线跟踪得到的图像太“整洁”了。用分布式光线跟踪可以降低这种效果（Cook et al., 1984）。传统的光线跟踪得到的图像看起来很整洁，因为一切都非常清晰；阴影也非常明显。反射没有模糊失真，一切都在完美的焦点上。有时我们可能希望阴影软一点（像它们在真实生活中那样），希望拉毛金属上的反射模糊一点，或者希望图像具有可变的焦点，就像在具有大光圈的照相机中。根据基本原理实现上述效果

229

比较困难（正如在第23章中实现的），我们对基本光线跟踪算法作一些简单的改变就可以获得大部分视觉效果。另外，这个结构提供了一种比较简单的图像反走样（见3.7节）方式。

10.11.1 反走样

回想一下，最简单的图像反走样方式是计算像素区域的平均颜色，而不是中心点颜色。在光线跟踪中，基本的计算是计算屏幕上一点的颜色。如果对像素中许多这样的点取平均，就接近实际的平均值了。如果包含该像素的屏幕坐标是$[i, i+1]\times[j, j+1]$，可以把代码

```
for each pixel (i, j) do
  c_ij = ray-color(i + 0.5, j + 0.5)
```

替换为下列代码，它在每个像素内的$n\times n$规则栅格上进行采样：

```
for each pixel (i, j) do
  c = 0
  for p = 0 to n − 1 do
    for q = 0 to n − 1 do
      c = c + ray-color(i + (p + 0.5)/n, j + (q + 0.5)/n)
  c_ij = c/n^2
```

这通常称为规则采样（regular sampling）。对于$n=4$，一个像素中有16个采样点，如图10-27所示。注意这种方法与用下面方法得到的图像是相同的：用传统的光线跟踪方法，在$n_x n\times n_y n$分辨率下，每个像素作为一个采样，然后对$n\times n$个像素区域取平均得到$n_x\times n_y$的图像。

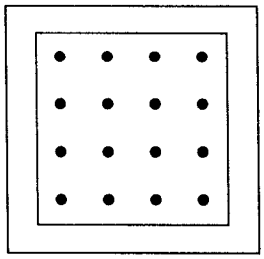


图10-27 一个像素的16个规则采样

像素内用规则模式采样潜在的问题是会出现规则的伪像，如网纹模式。在每一个像素内按照随机模式进行采样，可以把这些伪像变成噪声，如图10-28所示。这通常称为随机采样（random sampling），它只需要对代码进行一点修改：

230

```
for each pixel (i, j) do
  c = 0
  for p = 1 to n^2 do
    c = c + ray-color(i + ξ, j + ξ)
  c_ij = c/n^2
```

其中ξ是一个调用，它返回[0, 1)区间上的一个均匀随机数。不幸的是，噪声总是非常令人不快的，除非进行很多采样。一个折中的方法是采用混合策略，即对规则栅格进行随机扰动：

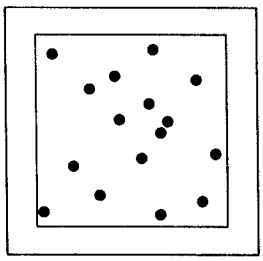


图10-28 一个像素的16个随机采样

```
for each pixel (i, j) do
  c = 0
  for p = 0 to n − 1 do
    for q = 0 to n − 1 do
      c = c + ray-color(i + (p + ξ)/n, j + (q + ξ)/n)
  c_ij = c/n^2
```

这种方法通常称为*抖动*（jittering）或者*分层采样*（stratified sampling）（图10-29）。

10.11.2 软阴影

阴影难以处理的原因是在标准光线跟踪中光源是无穷小的点或者方向，所以它要么完全可见要么完全不可见。在真实生活中，光源是有面积的区域，所以可能是部分可见的。这个观念在图10-30中以二维的形式表示出来。相对光源完全不可见的区域称为*本影*（umbra）。部分可见的区

域称为半影（penumbra）。不在阴影中的区域没有常用的术语，但是它有时被称为非阴影区（anti-umbra）。

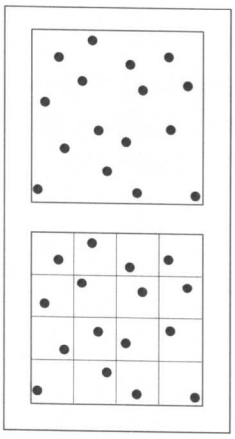


图10-29　单个像素内的16个分层（抖动）采样，用两种形式显示：有栅格线的和没有栅格线的。在每一个方格中恰好只有一个采样

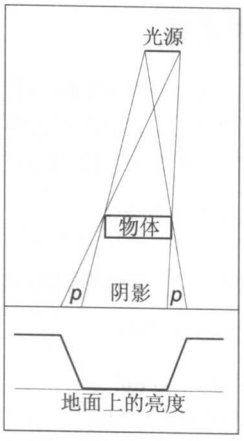


图10-30　软阴影从非阴影区向阴影区逐步过渡。图中用*p*标志的过渡区是“半影”

实现软阴影的关键是把光源作为一个区域而不是一个点。最简单的实现软阴影的方式是把光源近似为N个点光源的集合，每个点光源的强度是基本光源的$1/N$。图10-31左侧阐明了这个想法，其中用了9个光源。用标准光线跟踪程序中就可以实现软阴影，计算软阴影常用的策略是用现有的绘制器。这种技术有两个潜在的问题。第一，要获得平滑的视觉效果通常需要很多点光源，这会大大降低程序的速度。第二个问题是半影内的阴影过渡非常明显。

分布式光线跟踪在生成阴影的代码中引入一点改动。它不是把光源区域表示成离散数量的点
231 光源，而是把它表示为无穷多的光源，并随机地为每一条观察光线选择一个光源。这等价于为被照亮的表面的所有点在光源中选择一个随机点，如图10-31右侧所示。

如果光源是一个平行四边形，它由一个角点$\boldsymbol{c}$和两个边向量$\boldsymbol{a}$和$\boldsymbol{b}$确定（图10-32），那么选择一个随机点$\boldsymbol{r}$是很简单的：

$$\boldsymbol{r} = \boldsymbol{c} + \xi_1 \boldsymbol{a} + \xi_2 \boldsymbol{b}$$

其中ξ_1和ξ_2是[0, 1)区间上的均匀随机数。

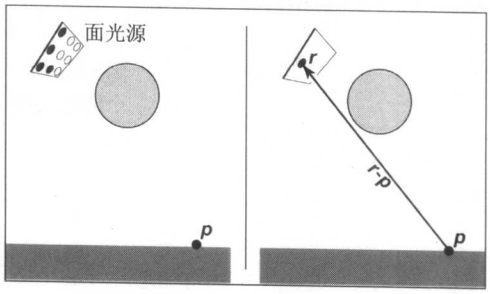


图10-31　左：面光源可以通过一定数量的点光源近似；9个点光源中有4个相对于$\boldsymbol{p}$是可见的，所以$\boldsymbol{p}$在半影中。右：在光源中为阴影光线选择一个随机点，它可能射中光源，也可能射不中

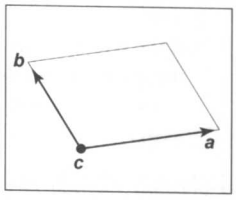


图10-32　平行四边形的光源由一个角点$\boldsymbol{c}$和两个边向量确定

然后我们向该点发出一条阴影光线，如图10-31右侧所示。注意阴影光线的方向向量不是单位长度，这可能需要对依赖于这个假设的基本光线跟踪程序进行一些修改。

可能需要对光线上的点进行抖动。然而，不加考虑地直接实现抖动是很危险的。我们并不总是希望像素左上角的光线产生一条指向光源左上角的阴影光线。相反，我们希望对采样进行干扰，这样对像素的采样和对光源的采样分别进行了扰动，但是这两者之间没有关系。一种实现抖动的好方法是生成两个独立的n^2个抖动采样的集合，然后把采样传递到光源程序中：

```
for each pixel (i, j) do
  c = 0
  generate N = n^2 jittered 2D points and store in array r[ ]
  generate N = n^2 jittered 2D points and store in array s[ ]
  shuffle the points in array s[ ]
  for p = 0 to N − 1 do
    c = c + ray-color(i + r[p].x(), j + r[p].y(), s[p])
  c_ij = c / N
```

232

混洗程序删除所有r和s一致的情况。计算阴影的程序将只使用保存在$s[p]$中的二维随机点，而不是调用随机数产生器。对于索引为从0到$N-1$的数组，混洗程序为

```
for i = N − 1 downto 1 do
  choose random integer j between 0 and i inclusive
  swap array elements i and j
```

10.11.3 景深

大部分照片中的软焦点效果可以这样模拟：在面积为非零的“透镜”上收集光线，而不是在一点上。这称为景深（depth of field）。透镜从一个方向圆锥中收集光线，圆锥的顶点在远处，这样所有的物体就都在焦点上（图10-33）。我们可以把采样的“窗口”放在所有物体都能聚焦的平面上（而不是前面用的平面$z=n$），并把镜头放在眼睛的位置。使所有物体聚焦的平面称为聚焦平面（focus plane），到该平面的距离可以由用户设置，就像真正的相机中到聚焦平面的距离是由用户或者测距仪来设置的一样。

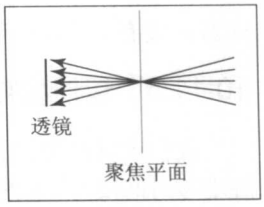


图10-33 透镜在一个方向圆锥中求平均值，这些方向与当前采样的像素相交

233

为了更忠实于真实的照相机，应该把镜片作为圆形。然而，这样得到的效果与使用正方形镜头的效果非常相似（图10-35）。所以我们要选择镜片的边长，并在其上进行随机采样。观察光线的起点将是经过扰动后的位置，而不是眼睛的位置。这里再次使用混洗程序防止与像素采样位置产生关联。图10-34的例子中每像素采用25个采样，并使用了一个大的圆形镜片。

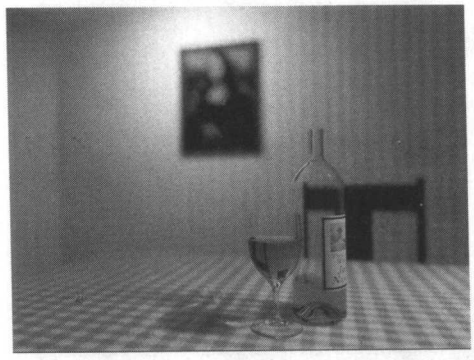


图10-34 一个景深的示例。酒瓶阴影中焦散现象是通过第23章中的粒子跟踪方法计算的（另见彩图VII）

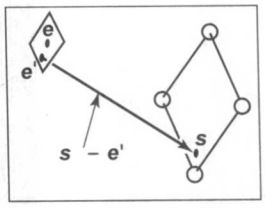


图10-35 为了创建景深效果，在一个正方形区域内随机选择眼睛位置

10.11.4　光泽反射

某些表面，如拉绒金属，一定程度上介于理想镜面和漫反射表面之间。反射图像中某些是可辨别的，但是变模糊了。我们可以对理想镜面反射光线进行扰动来模拟这种现象，如图10-36。

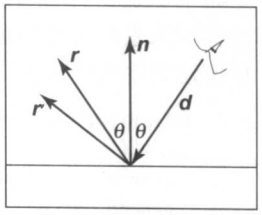


图10-36　反射光线经扰动后成为向量$\boldsymbol{r}'$

只需要确定两个细节：如何选择向量$\boldsymbol{r}'$，以及当扰动后的光线位于反射光线的平面下方时如何处理。当光线在表面下方时，往往通过返回零颜色来解决后一个细节问题。

为了选择$\boldsymbol{r}'$，可以在一个随机的正方形内采样。正方形垂直于$\boldsymbol{r}$，宽度a用来控制模糊的程度。利用2.4.6节中的技术，建立一个满足$\boldsymbol{w}=\boldsymbol{r}$的正交基，这样就可以确立正方形的朝向。然后在中心在原点、边长为a的二维正方形中确定一个随机点。如果二维采样点$(\xi,\xi')\in[0,1]^2$，那么在期望的正方形中的模拟点是

$$u=-\frac{a}{2}+\xi a$$

$$v=-\frac{a}{2}+\xi' a$$

因为要进行扰动的正方形平行于向量$\boldsymbol{u}$和$\boldsymbol{v}$，所以光线$\boldsymbol{r}'$为

$$\boldsymbol{r}'=\boldsymbol{r}+u\boldsymbol{u}+v\boldsymbol{v}$$

注意$\boldsymbol{r}'$没有必要是单位向量，但是如果程序要求光线的方向为单位向量，那么应该进行归一化。

10.11.5　运动模糊

我们可以在物体上添加模糊的效果，如图10-37所示。这称为*运动模糊*（motion blur），它是
234 在非零时间段内生成的图像。在真正的照相机中，光圈在一定时间内打开，在此期间物体产生了

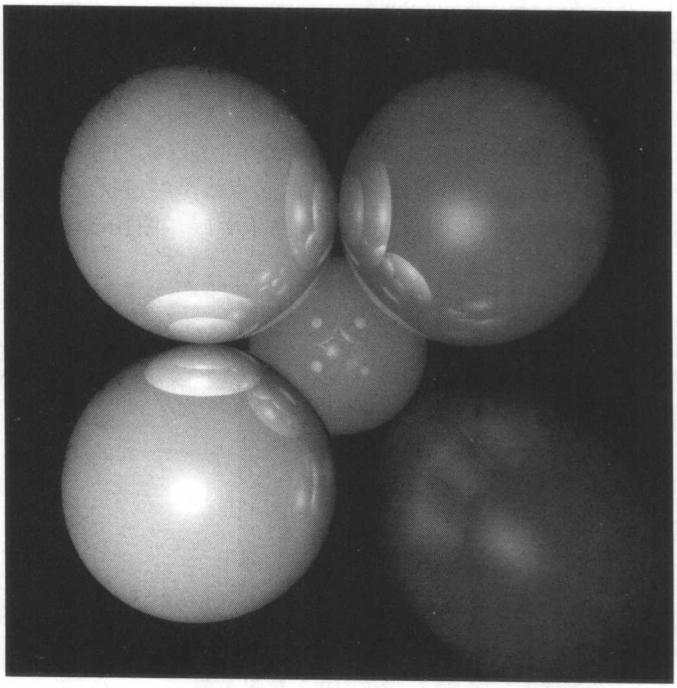


图10-37　右下方的球在运动，产生了模糊的外观。图像经Chad Barb授权使用

移动。可以设置一个时间变量模拟打开的光圈，变量的取值范围为T_0到T_1。对每一条观察光线，选择一个随机的时间段：

$$T = T_0 + \xi(T_1 - T_0)$$

可能还需要创建一些随时间移动的物体。例如有一个移动的球，球心在特定时间段内从c_0移动到c_1。给定T，可以计算球心的实际位置，并对球进行与光线的相交检测。因为每一条光线是在不同时间发出的，每一条光线将在不同的位置与球相交，所以最终的图像是模糊的。注意运动球体的包围盒应该包含其完整的路径，这样在整个时间段中就可以建立一个高效的数据结构（Glassner, 1988）。 235

常见问题

- **为什么光线跟踪中没有透视矩阵？**

z缓冲器中有透视矩阵，这样就可以把透视投影变成一个平行投影。在光线跟踪程序中这是不需要的，因为从眼睛开始在一个扇形区域内发出许多光线，就实现了隐式的透视投影。

- **什么是光线相交检测的最有效的结构？**

最常用的结构是BSP树、均匀子分栅格和包围体层次。哪一个最好没有明确的答案，但是实践中它们比蛮力搜索要好得多。如果只要实现一个，那么就用包围体层次，因为它简单而且稳健性好。

- **为什么人们使用包围盒立方体，而不是球体或者椭圆体？**

有时球体或者椭圆体会更好。然而许多模型具有多面体元素，这些元素可以被立方体紧密包围，但是要用椭圆体紧密包围这些元素是很困难的。

- **光线跟踪能够成为交互式的吗？**

对于足够小的模型和图像，任何现代PC机都足以使光线跟踪成为交互式的。在实践中，对基于全屏的实现，需要有共享帧缓冲器的多个CPU才行。因为计算机性能提高的速度比屏幕分辨率增加的速度快得多，所以传统PC机能够以屏幕分辨率实现复杂场景的光线跟踪只是时间的问题。

- **光线跟踪在硬件图形程序中有用吗？**

光线跟踪经常用于拾取（pick）。当用户鼠标点击三维图形程序中的一个像素时，程序需要确定该像素处哪个物体是可见的。光线跟踪是一个确定物体的理想方式。 236

习题

1. 光线$(1, 1, 1) + t(-1, -1, -1)$和以原点为圆心半径为1的球的交点的光线参数是多少？注意：这是用于调试的一个好例子。
2. 光线$(1, 1, 1) + t(-1, -1, -1)$和以$(1, 0, 0)$、$(0, 1, 0)$、$(0, 0, 1)$为顶点的三角形的交点的重心坐标和光线参数各是多少？注意：这是用于调试的一个好例子。
3. 实现物体包络线的计算，其复杂度与在“好”的模型上进行光线跟踪的复杂度近似。把分析过程分成两部分：预处理和计算图像，当计算静态模型的多帧图像时就可以预测光线跟踪程序的行为。 237

第11章

纹理映射

第9章中的明暗处理模型假设漫反射表面具有均匀的反射率c_r。这对于白纸或粉刷的墙是合适的，但是对于像一张已打印过的纸这样的物体是无效的。这类物体外观的复杂性是由反射率的变化引起的。虽然可以使用小的三角形，然后通过改变三角形的反射特性来获得这种变化，但效率太低。

常用的处理变反射率的技术是把反射率作为一个函数或者一幅基于像素的图像来存储，然后把它“映射”到表面上（Catmull, 1975）。该函数或者图像称为纹理图（texture map），控制反射率特性的过程称为纹理映射（texture mapping）。一旦理解了相关的坐标系统，这并不难实现。纹理映射可以按照不同的性质分类：

(1) 纹理函数的维数。

(2) 表面上的点和纹理函数中的点之间的一致性。

(3) 本质上纹理函数是一个程序还是一个查找表。

这些因素通常是紧密相关的，所以现在随意地按照维数对纹理进行分类。首先考虑三维纹理，
239 它常被称为实体纹理（solid texture）或者体纹理（volume texture）。然后考虑二维纹理，它有时被称为图像纹理（image texture）。当图形程序员谈到纹理而没有指定维数的时候，他们通常指的是二维纹理。但我们先从三维纹理开始，因为在很多方面它更易于理解和实现。本章最后将讨论凹凸映射和置换映射，它们分别用纹理改变表面法线和位置。尽管它们修改的性质不是反射率，但它们用到的图像或函数仍然称为纹理。这和通常的用法是一致的，任何用来修改物体外观的图像都称为纹理。

11.1 三维纹理映射

在前面的章节中，用c_r作为物体上某点的反射率。对于不具有一致颜色的物体，可以用一个函数$c_r(\boldsymbol{p})$替换它，该函数是一个三维点到RGB颜色的映射（Peachey, 1985; Perlin, 1985）。它可能只返回包含$\boldsymbol{p}$的物体的反射率。但是对于具有纹理的物体，我们希望$c_r(\boldsymbol{p})$随$\boldsymbol{p}$在物体表面的移动而改变。一种方式是创建三维纹理，它在三维空间中所有点都定义一个RGB值。我们将只调用表面上点$\boldsymbol{p}$处的值，但是为所有的三维点定义纹理值要比为二维表面点集定义纹理简单得多，因为任意表面上点的二维子集形状可能很奇怪。显然这种策略对于由实体介质雕刻出的表面是适用的，例如大理石雕塑。

注意在光线跟踪程序中，可以直接访问从一个像素看到的点$\boldsymbol{p}$。然而，对于z缓冲器或者BSP树程序，只有投影到设备坐标系中才能知道该点。我们将在11.4.1节中解决这个问题。

11.1.1 三维条纹纹理

建立条纹纹理的方式非常多。假设有两种颜色c_0和c_1，把它们作为条纹的颜色。现在需要某

个振荡函数在两种颜色间切换。一个简单的函数是正弦函数：

```
RGB stripe ( point p )
if (sin(x_p) > 0) then
  return c_0
else
  return c_1
```

还可以控制条纹的宽度ω：

```
RGB stripe ( point p, real w)
if (sin(πx_p /w) > 0) then
  return c_0
else
  return c_1
```

如果希望在条纹颜色间进行平滑插值，可以使用参数t线性地改变颜色：

```
RGB stripe ( point p, real w)
t = (1 + sin(πp_x/w)) /2
return (1 – t)c_0 + tc_1
```

这3种可能性显示在图11-1中。

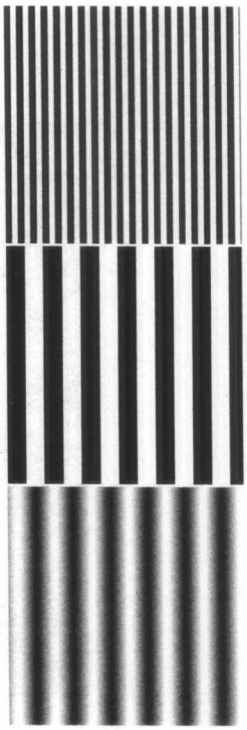


图11-1 不同的条纹纹理，它们是在保持z不变的情况下画出xy点的规则阵列而得出

11.1.2 纹理数组

用来指定空间中纹理的另一种方式是用一个三维数组存储颜色值，并把空间位置与这些值关联起来。首先讨论二维空间中的二维数组。这样的纹理就可以应用到三维中，方法是用两个维度（如x和y）来确定用什么纹理值。然后我们把二维的结果推广到三维中。

假设要映射的两个维度称为u和v。再假设有一个$n_x \times n_y$的图像作为纹理。我们需要每一个(u, v)在该图像中具有一个相关的颜色。要使纹理映能够映射在(u, v)上，比较标准的方式是首先删除(u, v)的整数部分，使它位于一个单位正方形中。这样的效果是把现在的正方形纹理平铺在整个uv平面上（图11-2）。然后用三种插值策略之一计算该坐标处的图像颜色。最简单的策略是把每一个图像像素作为具有固定颜色的矩形方格（图11-3a）。为了计算颜色，应用$c(u,v) = c_{ij}$，这里$c(u,v)$是(u, v)位置的颜色，c_{ij}是像素索引处的像素颜色：

$$\begin{aligned} i &= \lfloor un_x \rfloor \\ j &= \lfloor un_y \rfloor \end{aligned} \tag{11-1}$$

其中$\lfloor x \rfloor$是对x向下取整，(n_x, n_y)是纹理图的大小，索引从$(i, j) = (0, 0)$开始。对简单图像应用该方法的结果如图11-3b所示。

对于比较平滑的纹理，可以使用双线性插值，如图11-3c所示。这里使用公式

$$\begin{aligned} c(u,v) = {} & (1-u')(1-v')c_{ij} \\ & +u'(1-v')c_{(i+1)j} \\ & +(1-u')v'c_{i(j+1)} \\ & +u'v'c_{(i+1)(j+1)} \end{aligned}$$

其中

$$u' = n_x u - \lfloor n_x u \rfloor$$
$$v' = n_y v - \lfloor n_y v \rfloor$$

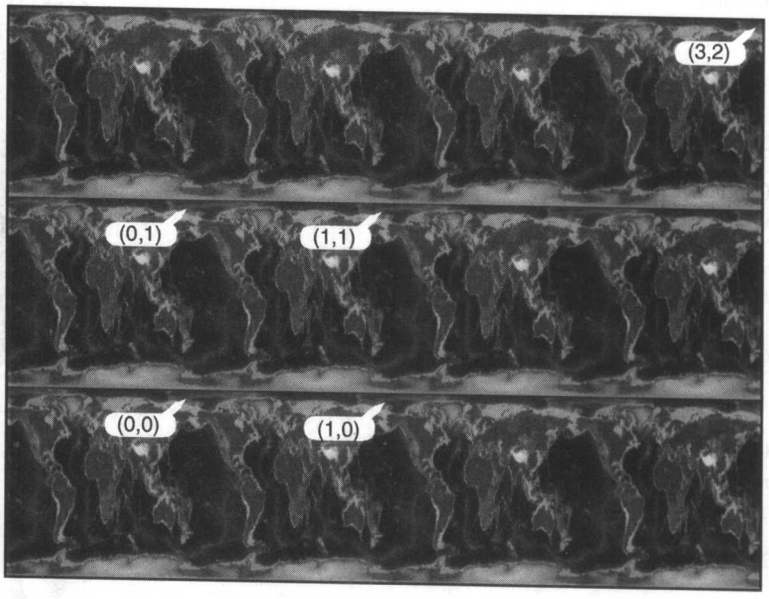


图11-2　一幅图像在(u, v)平面上平铺的效果。注意输入图像是矩形的，该矩形映射到(u, v)平面中的单位正方形上

亮度导数的不连续会引起明显的马赫条带，所以可以使用平滑的埃尔米特（Hermit）插值，得到的结果如图11-3d所示：

$$\begin{aligned} c(u,v) = {} & (1-u'')(1-v'')c_{ij} \\ & +u''(1-v'')c_{(i+1)j} \\ & +(1-u'')v''c_{i(j+1)} \\ & +u''v''c_{(i+1)(j+1)} \end{aligned}$$

242

其中

$$u'' = 3(u')^2 - 2(u')^3$$
$$v'' = 3(v')^2 - 2(v')^3$$

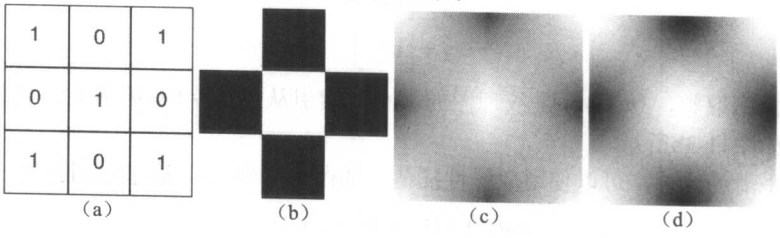


图11-3　（a）左边的图像具有9个像素，都是黑色或者白色。3种插值策略是（b）最近邻法、（c）双线性插值和（d）埃尔米特插值

在三维情况下，使用一个三维数组。所有这些思想可以自然地从二维扩展得来。例如，假设要在数值间进行三线性插值。首先计算纹理坐标(u', v', w')和进行插值的数组元素的索引下限(i, j, w)：

$$
\begin{aligned}
c(u,v,w) = {} & (1-u')(1-v')(1-w')c_{ijk} \\
& +u'(1-v')(1-w')c_{(i+1)jk} \\
& +(1-u')v'(1-w')c_{i(j+1)k} \\
& +(1-u')(1-v')w'c_{ij(k+1)} \\
& +u'v'(1-w')c_{(i+1)(j+1)k} \\
& +u'(1-v')w'c_{(i+1)j(k+1)} \\
& +(1-u')v'w'c_{i(j+1)(k+1)} \\
& +u'v'w'c_{(i+1)(j+1)(k+1)}
\end{aligned}
\tag{11-2}
$$

其中

$$
\begin{aligned}
u' &= n_x u - \lfloor n_x u \rfloor \\
v' &= n_y v - \lfloor n_y v \rfloor \\
w' &= n_z w - \lfloor n_z w \rfloor
\end{aligned}
\tag{11-3}
$$

11.1.3 实体噪声

尽管普通的纹理（如条纹）常常是很有用的，我们还希望能建立“有斑点的”纹理，就像在 243
鸟蛋上见到的。这通常可以用一种“实体噪声”来实现，在它发明以后通常被称为Perlin噪声（Perlin noise），Perlin为在电影工业中的影响而荣获技术类奥斯卡奖（Perlin, 1985）。

为了得到有噪声的外观，为每一个点调用一个随机数将是不合适的，因为它就像电视静电中的“白噪声”。我们希望使它更平滑，而又不失随机性。一种可能是对白噪声进行模糊化，但是这至今还没有实现。另一种可能是建立一个大的点阵，每一点上有一个随机数，这就是上一节中描述的三维纹理数组，数组中是随机数。这种技术使每一个点都太明显了。Perlin使用了一系列小技巧改善了基本的点阵技术，使点阵看起来不太明显。这样就得到一系列巴洛克式的步骤，但本质上这只对三维随机数数组的线性插值作了3个改动。第一个改动是用埃尔米特插值避免马赫条带，就像处理规则纹理一样。第二个改动是使用随机向量，而不是数字，通过点乘得到一个随机数；这样能把局部最小和局部最大点从栅格点上移开，使得基本的栅格结构更不明显。第三个改动是使用一维数组和散列表创建虚拟的随机向量三维数组。这增加了计算量而降低了内存使用。下面是他的基本方法：

$$
n(x,y,z) = \sum_{i=\lfloor x \rfloor}^{\lfloor x \rfloor+1} \sum_{j=\lfloor y \rfloor}^{\lfloor y \rfloor+1} \sum_{k=\lfloor z \rfloor}^{\lfloor z \rfloor+1} \Omega_{ijk}(x-i, y-j, z-k)
$$

其中(x,y,z)是$\boldsymbol{x}$的笛卡儿坐标，并且

$$
\Omega_{ijk}(u,v,w) = \omega(u)\omega(v)\omega(w)(\Gamma_{ijk} \cdot (u,v,w))
$$

$\omega(t)$是三次权重函数：

$$
\omega(t) = \begin{cases} 2|t|^3 - 3|t|^2 + 1 & \text{如果} |t| < 1 \\ 0 & \text{其他} \end{cases}
$$

最后一部分Γ_{ijk}是栅格点$(x,y,z)=(i,j,k)$的一个随机单位向量。因为要得到任意的ijk，所以使用一个伪随机表：

$$
\Gamma_{ijk} = \boldsymbol{G}(\phi(i+\phi(j+\phi(k))))
$$

其中$\boldsymbol{G}$是预先计算出的数组，其中包含n个随机单位向量。$\phi(i) = P[i \bmod n]$，其中P是长度为n 244

的数组，它包含从整数0到$n-1$的排列。实践中，Perlin认为$n=256$效果较好。为了选择一个随机的单位向量(v_x, v_y, v_z)，首先设

$$v_x = 2\xi - 1$$
$$v_y = 2\xi' - 1$$
$$v_z = 2\xi'' - 1$$

其中ξ、ξ'、ξ''是规范的随机数（归一到区间[0, 1)上）。那么如果$(v_x^2 + v_y^2 + v_z^2) < 1$，把向量变换为单位向量。否则随机设置该向量直到它的长度小于1，然后把它变换为单位向量。这是*拒绝方法*的一个例子，第14章中将详细讨论该方法。本质上，“小于”检测得到的是单位球中一个随机点，从原点到该点的向量是随机均匀分布的。但是这对立方体中的随机点就不成立了，所以可以用这个检测“删除”立方体的角。

因为实体噪声可以为正或者为负，所以把它转换为颜色之前必须先进行变换。图11-4是在一个10×10正方形中的噪声绝对值，另外还有拉伸后的变型。这些变型是通过对噪声函数的输入点进行缩放实现的。

深色的曲线是原来的噪声函数由正转负的地方。噪声从-1到1间变化，所以用(noise+1)/2作为颜色可以得到更平滑的图像。然而噪声值很少接近1或-1，这将是一个相当平滑的图像。更大的缩放系数可以提高对比度（图11-5）。

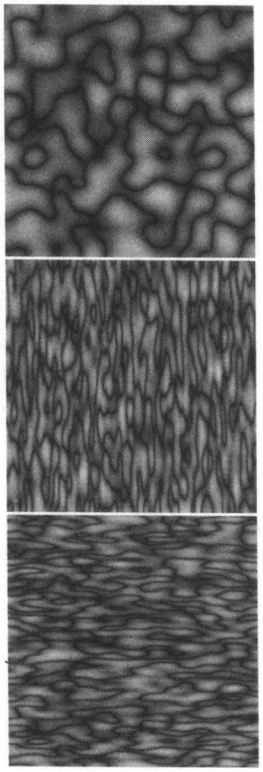


图11-4　实体噪声的绝对值、x缩放后的噪声和y缩放后的噪声

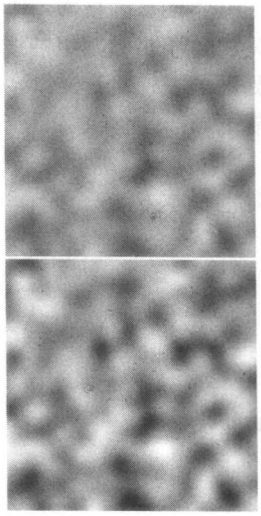


图11-5　（上）使用0.5(noise + 1)作为亮度，（下）使用0.8(noise + 1)作为亮度

11.1.4　扰动

许多自然纹理在同一个纹理中包含着不同尺寸的特征。Perlin用了一个伪分形“扰动”函数：

$$n_t(\mathbf{x}) = \sum_i \frac{|n(2^i \mathbf{x})|}{2^i}$$

它能有效地在自身上反复添加经过缩放的噪声函数，如图11-6所示。

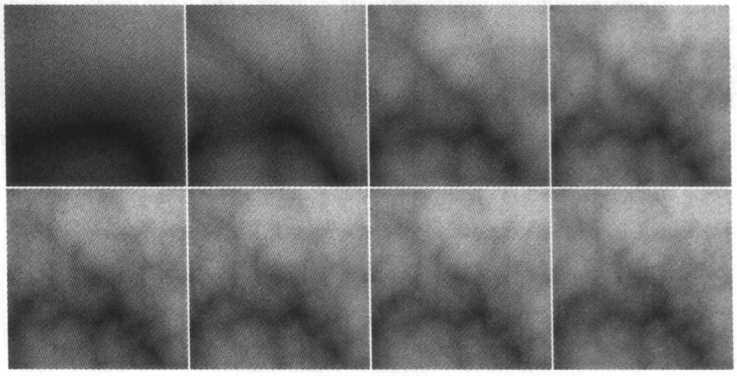


图11-6 分别采用1项到8项之和的扰动函数（从左上到右下）

可以用扰动函数对条纹函数进行变形：

```
RGB turbstripe ( point p, double w)
double t = (1 + sin(k1zp + turbulence(k2p)))/w)/2
return t * s0 + (1 – t) * s1
```

用不同的k_1和k_2就可以得到图11-7。 245

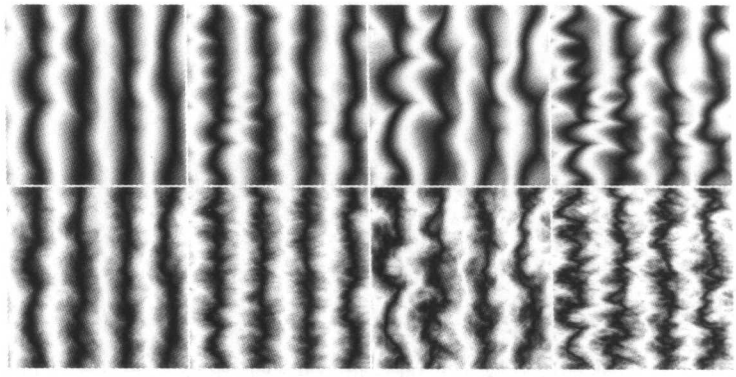


图11-7 具有不同的k_1和k_2的扰动条纹纹理。上面一行只用了扰动系列的第一项

11.2 二维纹理映射

对于二维纹理映射，我们使用二维坐标，它常被称为uv，用来建立反射率$R(u, v)$。关键是把图像和一个(u, v)坐标系统建立关联，然后就能和三维表面上的点建立关联。例如，如果把世界地图上的经度和维度与球体上的极坐标系建立关联，就可以得到一个地球仪（图11-8）。

图像上的坐标和物体以“正确的方式”进行匹配是至关重要的。作为惯例，图像上的坐标系统设成单位正方形$(u,v) \in [0, 1]^2$。对于正方形外面的(u, v)，只用坐标的小数部分，这样得到平面平铺的效果（图11-2）。注意图像水平和垂直方向的像素数不同，所以图像像素在(u, v)空间中的 246
纵横比是不均匀的。

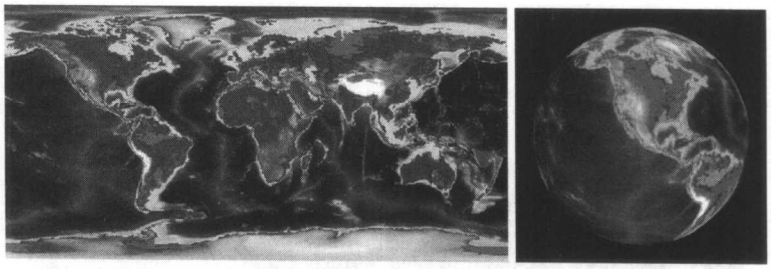


图11-8　用一幅米勒圆柱体投影图得到的世界地图，和把它放在球体上的效果。纹理地图的变形（如格陵兰岛非常大）恰好与把地图映射到球体上产生的收缩对应

为了把$(u,v)\in[0,1]^2$的图像映射到球体上，首先计算极坐标。回忆一下式（2-24）中描述的球坐标系。对于一个半径为R圆心为(c_x,c_y,c_z)的球，其参数方程为

$$x = x_c + R\cos\phi\sin\theta$$
$$y = y_c + R\sin\phi\sin\theta$$
$$z = z_c + R\cos\theta$$

由此得到$(\theta,\ \phi)$：

$$\theta = \arccos\left(\frac{z-z_c}{R}\right)$$
$$\phi = \arctan 2(y-y_c, x-x_c)$$

其中arctan2$(a,\ b)$是大多数数学库中的*atan2*，它返回a/b的反正切。因为$(\theta,\ \phi)\in[0,\ \pi]\times[-\pi,\ \pi]$，我们按照下列方法转换到$(u,\ v)$，注意如果$\phi$为负则先加$2\pi$：

$$u = \frac{\phi}{2\pi}$$
$$v = \frac{\pi-\theta}{\pi}$$

这种映射如图11-8所示。有一种类似的方式（尽管可能更复杂），可以为大多数三维形体建立坐标。

247

11.3　镶嵌模型

真实世界中大多数模型是由共享顶点的三角形集合体组成。就是大家知道的三角形网格（triangular mesh）或者不规则三角形网络（triangular irregular network, TIN）。大多数图形程序要求模型不能占用太多内存并且具有处理纹理图的能力。

图11-9中是一个简单的三角形网格。可以把3个三角形作为独立的元素存储，这样点$\boldsymbol{p}_1$要存3次，另外两点分别存2次，总共存9个点（每一个三角形存3个点），或者也可以以某种方式共享公用的点，只存4个点。所以这里不用下面的类：

```
class triangle
material m
vector3 p0, p1, p2
```

我们可以使用两个类：

```
class mesh
material m
```

array of vector3 vertices

和

class meshtriangle
pointer to mesh meshptr
int i_0, i_1, i_2

其中i_0、i_1、i_2是*vertices*数组中的索引。用triangle类或者mesh类都可以工作。mesh类有空间上的优势吗？通常大型的网格中，每一个顶点大概存储在6个三角形中，尽管在极端情况下可能是任意的个数。这意味着一个共享顶点大约对应两个三角形。如果有n个三角形，在共享模式下大概有$n/2$个顶点，在非共享模式下有$3n$个顶点。但是，当共享顶点时，另外还需要$3n$个整数和n个指针。既然不需要在每一个网格三角形中都存储材质，这就节省了n个指针，可以删除指针*meshptr*。如果假设浮点、指针和整型数据需要相同的内存（一个可疑的假设），所有三角形将占用$10n$个存储单位，而整个网格将占用$5.5n$个存储单位。所以网格减少了将近一半的存储量；这好像对大部分实现都是有效的。为这一半的存储量值得采样这样复杂的数据结构吗？当开始添加顶点的性质时，我想回答是肯定的。 248

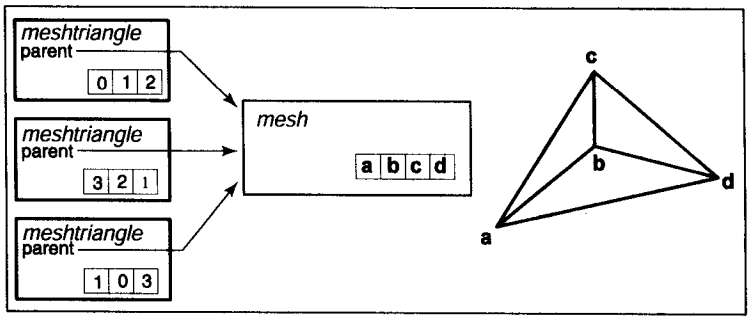


图11-9 4个顶点、3个三角形的网格

每个顶点都有材质参数、纹理坐标、辐射度和绘制程序可能用到的任何重要参数。实际上要在三角形中对这些参数进行双线性插值。所以如果三角形在重心坐标(β, γ)上分割，可以按照对点进行插值的方式在(u, v)坐标中进行插值。回忆一下在重心坐标(β, γ)上的点是

$$\boldsymbol{p}(\beta,\gamma)=\boldsymbol{a}+\beta(\boldsymbol{b}-\boldsymbol{a})+\gamma(\boldsymbol{c}-\boldsymbol{a})$$

类似的公式应用到(u, v)上，得

$$u(\beta,\gamma)=u_a+\beta(u_b-u_a)+\gamma(u_c-u_a)$$
$$v(\beta,\gamma)=v_a+\beta(v_b-v_a)+\gamma(v_c-v_a)$$ 249

改变三角形顶点的(u, v)坐标，可以按照多种方式应用纹理，如图11-10所示。在调试的时候，这种校准纹理图使我们更易于理解物体的纹理坐标（图11-11）。

11.4 光栅化三角形的纹理映射

无论用光线跟踪程序还是光栅化方法，如z缓冲器方法，都希望用同样的纹理图。为了获得正确的透视效果，这里有几个微妙的问题，但它们都可以在光栅化阶段解决。事情并不简单，原因是只在屏幕空间进行纹理坐标插值得到的是错误的图像，就像图11-12中的网格纹理。因为物体在透视投影下会随着到观察者距离的增加而变小，在三维空间中均匀分布的直线在二维图像空间中将会被压缩。只有对纹理坐标进行更细致的插值才能实现这种效果。

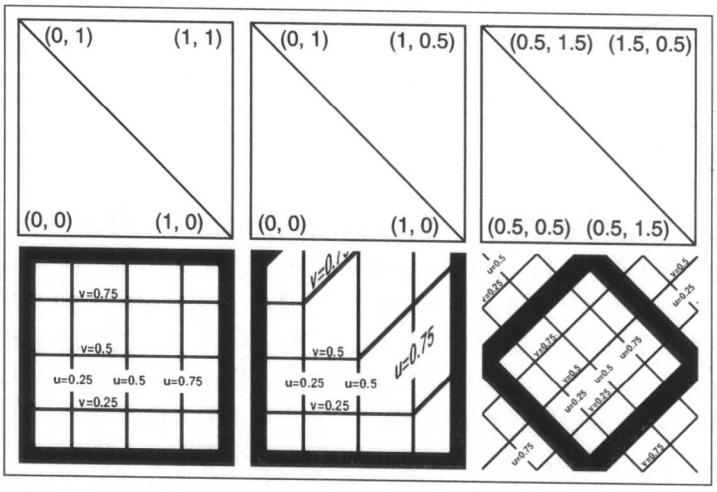


图11-10　通过改变存储在顶点中的(u, v)坐标而得到的几种网格纹理

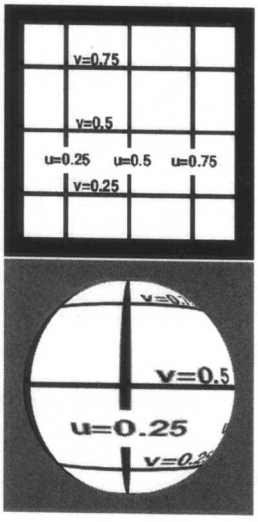


图11-11　上：校准纹理图。下：沿y轴观察球体

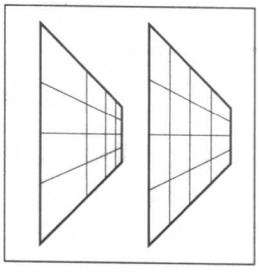


图11-12　左：正确的透视。右：在屏幕空间中的插值

透视的正确纹理

要实现三角形中的纹理映射，可以对(u, v)坐标进行插值并修正3.6节中的光栅化方法，但是这会出现问题，如图11-12右侧所示。如果在下列光栅化代码中使用屏幕空间的重心坐标，三角形将会出现类似的问题：

```
for all x do
  for all y do
    compute (α, β, γ) for (x, y)
    if α ∈ (0,1) and β ∈ (0,1) and γ ∈ (0,1) then
      t = αt0 + βt1 + γt2
      drawpixel (x, y) with color texture(t) for a solid texture
      or with texture(β, γ) for a 2D texture
```

上述代码将生成图像，但是会有问题。为了弄清根本的问题，考虑从世界空间$\boldsymbol{q}$到齐次点$\boldsymbol{r}$

再到齐次化的点s这个连续变换过程：

$$\begin{bmatrix} x_q \\ y_q \\ z_q \\ 1 \end{bmatrix} \xrightarrow{\text{transform}} \begin{bmatrix} x_r \\ y_r \\ z_r \\ h_r \end{bmatrix} \xrightarrow{\text{homogenize}} \begin{bmatrix} x_r/h_r \\ y_r/h_r \\ z_r/h_r \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix}$$

250

如果使用屏幕空间，那么是在s中进行插值。然而我们希望在空间q或r中进行插值，在这个空间中齐次除法还没有对三角形重心坐标进行非线性变形。

关键是对$1/h_r$插值没有产生变形，对u/h_r和v/h_r也一样。事实上，对k/h_r也一样，k是对三角形进行线性变换的量。回想7.4节中，如果沿着点q和Q间的线段对所有点进行变换并齐次化，得

$$\mathbf{s}+\frac{h_R t}{h_r+t(h_R-h_r)}(\boldsymbol{S}-\boldsymbol{s})$$

但是如果在齐次空间中进行线性变换，得

$$\mathbf{s}+a(\boldsymbol{S}-\boldsymbol{s})$$

尽管这些线段扫描过相同的点，对于线段上相同的点通常$a \neq t$。然而如果对$1/h$插值，那么无论在哪个空间中进行插值，确实都能得到相同的结果。为了说明这一点，请证明（习题2）：

$$\frac{1}{h_r}+\frac{h_R t}{h_r+t(h_R-h_r)}\left(\frac{1}{h_R}-\frac{1}{h_r}\right)=\frac{1}{h_r}+t\left(\frac{1}{h_R}-\frac{1}{h_r}\right) \tag{11-4}$$

在变换后的空间中线性地对$1/h$插值不会产生错误，这种特性允许我们对三角形进行正确的纹理计算。处理变形的最不令人混淆的方式可能是根据屏幕空间坐标(β, γ)计算三角形世界空间重心坐标(β_w, γ_w)。注意β_s/h和γ_s/h可以在屏幕空间中进行线性插值。例如，在屏幕空间中与屏幕空间重心坐标(β, γ)相关的位置，可以不变形地进行插值β_w/h。因为在顶点0和顶点2处$\beta_w=0$，在顶点1处$\beta_w=1$，有

$$\frac{\beta_s}{h}=\frac{0}{h_0}+\beta\left(\frac{1}{h_1}-\frac{0}{h_0}\right)+\gamma\left(\frac{0}{h_2}-\frac{0}{h_0}\right) \tag{11-5}$$

公式（11-5）是相当简单的，因为其中有很多零项。然而为了从中得到β_w，必须要知道h。因为已知$1/h$在屏幕空间中是线性的，有

$$\frac{1}{h}=\frac{1}{h_0}+\beta\left(\frac{1}{h_1}-\frac{1}{h_0}\right)+\gamma\left(\frac{1}{h_2}-\frac{1}{h_0}\right) \tag{11-6}$$

公式（11-5）除以公式（11-6），得

$$\beta_w=\frac{\dfrac{\beta}{h_1}}{\dfrac{1}{h_0}+\beta\left(\dfrac{1}{h_1}-\dfrac{1}{h_0}\right)+\gamma\left(\dfrac{1}{h_2}-\dfrac{1}{h_0}\right)}$$

251

分子分母分别乘以$h_0h_1h_2$，然后对γ_w中类似的公式进行一系列同样的运算得

$$\begin{aligned}\beta_w&=\frac{h_0h_2\beta}{h_1h_2+h_2\beta(h_0-h_1)+h_1\gamma(h_0-h_2)}\\ \gamma_w&=\frac{h_0h_1\gamma}{h_1h_2+h_2\beta(h_0-h_1)+h_1\gamma(h_0-h_2)}\end{aligned} \tag{11-7}$$

注意两个分母是相同的。

对于使用第7章中透视矩阵的三角形，回忆一下 $w = z/n$，其中z是观察者到屏幕的垂直距离。这样，矩阵1/z也是线性变化。可以利用这个事实修改对三维点 $t_i = (x_i, y_i, z_i, h_i)$ 进行扫描转换的代码，这些点传给观察矩阵但是并没有齐次化：

```
Compute bounds for x = x_i / h_i and y = y_i / h_i
for all x do
  for all y do
    compute (α, β, γ) for (x, y)
    if (α ∈ [0,1] and β ∈ [0,1] and γ ∈ [0,1] then
      d = h1h2 + h2β(h0 − h1) + h1γ(h0 − h2)
      β_w = h0h2β / d
      γ_w = h0h1γ / d
      α_w = 1 − β_w − γ_w
      u = α_w u0 + β_w u1 + γ_w u2
      v = α_w v0 + β_w v1 + γ_w v2
      drawpixel (x, y) with color texture(u, v)
```

对于实体纹理，回忆一下重心坐标的定义：

$$\boldsymbol{p} = (1-\beta_w-\gamma_w)\boldsymbol{p}_0 + \beta_w\boldsymbol{p}_1 + \gamma_w\boldsymbol{p}_2$$

其中$\boldsymbol{p}_i$是世界空间中的顶点。所以只需要对点$\boldsymbol{p}$调用实体纹理映射程序。

11.5　凹凸纹理

目前为止，尽管只讨论了用纹理改变反射率，但是也可以用它改变表面法线，这样就会得到一个假象，即表面具有精致几何形状。可以用*凹凸映射*对表面法线进行扰动（J. F. Blinn, 1978）。

252

实现凹凸纹理的一种方式是

```
vector3 n = surfaceNormal(x)
n += k1 * vectorTurbulence(k2 * x)
return t * s0 + (1 − t) * s1
```

如图11-13所示。

为了实现vectorTurbulence，首先需要知道vectorNoise，它生成一个简单的在空间中变化的三维向量：

$$n_v(x,y,z) = \sum_{i=\lfloor x\rfloor}^{\lfloor x\rfloor+1}\sum_{j=\lfloor y\rfloor}^{\lfloor y\rfloor+1}\sum_{k=\lfloor z\rfloor}^{\lfloor z\rfloor+1}\Gamma_{ijk}\omega(x)\omega(y)\omega(z)$$

然后vectorTurbulence可以直接模拟扰动：即把一系列经过缩放的vectorNoise相加。

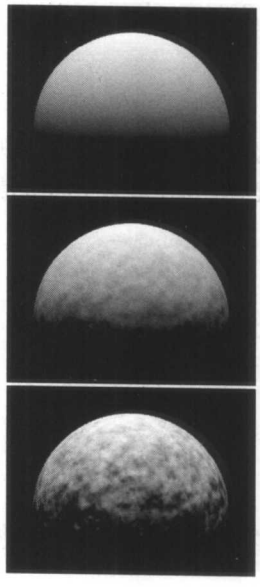


图11-13　在半径为1.6的球体上进行向量扰动。光照来自正上方。上：$k_1 = 0$。中：$k_1 = 0.08$，$k_2 = 8$。下：$k_1 = 0.24$，$k_2 = 8$

11.6　置换映射

图11-13中有一个问题，表面的扰动既没有阴影也没有影响物体的轮廓。因为它并没有真正改变任何几何形状，所以产生了这些局限。如果想得到更强的真实感，可以采用*置换映射*（Cook et al., 1987）。置换映射实际上是用纹理改变几何形状。常用的简化是沿着表面法线方向进

行置换。

如果把表面上所有的点p与表面法线向量n相关联，就可以用一个三维纹理$d(p)$构造一个新的表面：

$$p' = p + f(p)n$$

这种思想显示在图11-14中。

置换映射可以在z缓冲器程序中直接实现，方法是把要置换的表面作为一个由许多三角形组成的精致的网格来存储。网格中每一个顶点沿着法线向量方向进行置换。这样得到的模型很大，但是它的稳健性很好。

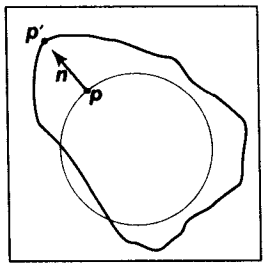


图11-14 函数$f(p)$把圆上所有的点p分别沿着n的方向进行置换。如果f是连续的，最终的点p'形成一个连续的表面

11.7 环境映照

我们常常希望背景是通过纹理映射得到的，并且希望物体能反射出这个背景。应用环境映照（environment map）（J.F. Blinn, 1976）就可以实现这样的效果。环境映照可以作为背景函数来实现，其输入是一个观察方向b，返回纹理图中一个RGB值。存储环境映照的方式有很多。例如，可以用一个索引为球坐标的球形表。本节中，将讨论基于立方体的表，它有6个正方形的纹理图，通常称为立方体图（cube map）。

253

立方体图的基本思想是假设有一个无限大的立方体，其每一个面上有一个纹理。因为立方体很大，所以光线的起点并不改变光线“看到”的东西。这等价于一个任意大小的立方体，用起点在笛卡儿坐标原点的光线对它进行查询。如何把给定的方向b转化为(u, v)坐标呢？作为一个例子，考虑图11-15中右边的面。这里x_b作为最大的绝对值项。在这种情况下，可以计算纹理的(u, v)，得

$$u = \frac{y + x}{2x}$$

$$v = \frac{z + x}{2x}$$

其他5个面也有类似的公式。

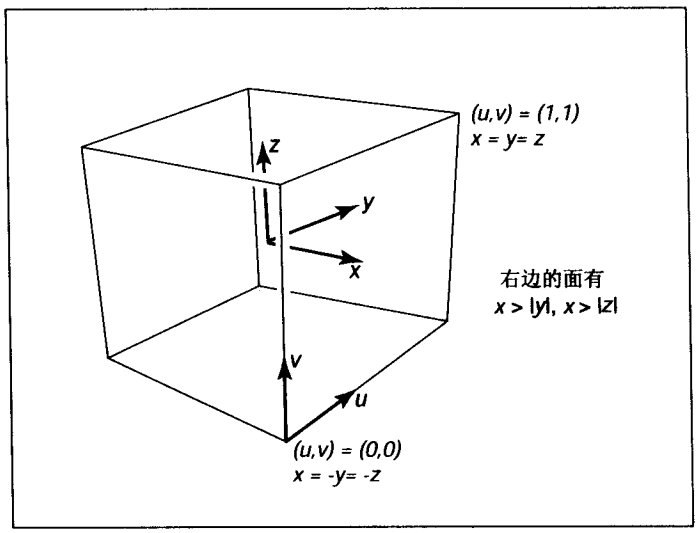


图11-15 具有6个平行于坐标轴的纹理的立方体图，它存储了背景。右边的面包含一个纹理

254 这样对于任意反射光线$\boldsymbol{a}+t\boldsymbol{b}$，返回$cubemap(\boldsymbol{b})$作为背景颜色。在z缓冲器的程序中，需要逐个像素地执行这个计算。如果在一个给定的像素上，知道观察方向$\boldsymbol{c}$和表面法线向量$\boldsymbol{n}$，就可以计算反射方向$\boldsymbol{b}$（图11-16）。我们可以通过修改公式（9-6）来计算

$$\boldsymbol{b}=-\boldsymbol{c}+\frac{2(\boldsymbol{c}\cdot\boldsymbol{n})\boldsymbol{n}}{\|\boldsymbol{c}\|^2} \tag{11-8}$$

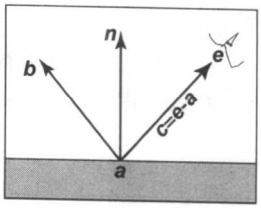


图11-16　向量$\boldsymbol{b}$是向量$\boldsymbol{c}$相对于表面法线向量$\boldsymbol{n}$的反射

其中分数的分母说明c可能不是单位向量。因为要知道每一个像素处的$\boldsymbol{b}$，我们可以在每一个三角形的顶点计算$\boldsymbol{b}$，然后按照正确的透视方式对$\boldsymbol{b}$进行插值。也可以先对$\boldsymbol{n}$进行插值，然后计算每一个像素的$\boldsymbol{b}$。这允许我们在每一个像素处调用$cubemap(\boldsymbol{b})$。

11.8　阴影图

如果把光源位置作为眼睛的位置来绘制场景，可见面都将被照亮，隐藏面将都在阴影中。这是关于阴影图一个基本事实，它可以用来确定光栅化的点是否在阴影中（L. Williams, 1978）。首先，从光源位置用矩阵$\boldsymbol{M}_s$对场景进行光栅化。该矩阵与7.3节中用来进行观察的全变换矩阵$\boldsymbol{M}$相同，但是它把光源的位置作为眼睛的位置，并把光线的主方向作为观察平面的法线。

回忆一下矩阵$\boldsymbol{M}$把世界坐标系中的(x, y, z)转换成关于屏幕的(x', y', z')。当按照正确的透视方式进行光栅化时，可以得到通过每一个像素中心看到的(x, y, z)。如果再用$\boldsymbol{M}_s$对点进行光栅化，并对得到的x和y坐标四舍五入，将得到

$$(i, j,\ \text{深度})$$

把这个深度与阴影深度图中像素（i, j）的z值比较一下。如果相同，则该点是被照亮的，否则它在阴影中。因为计算精度的问题，实际上是要检测在一个小的常数范围内点是否相同。

因为通常我们不希望光线只在一个正方形窗口中，所以常常用点光源。光源的亮度随着到阴
255 影缓冲器的距离而衰减。例如，如果阴影缓冲器有$n\times n$个像素，对于阴影缓冲器中的像素（i, j），可以使用基于小数半径r的衰减因子：

$$r=\sqrt{\left(\frac{2i-n}{n}\right)^2+\left(\frac{2j-n}{n}\right)^2}$$

任何放射状的衰减函数都呈现出聚光灯的样子。

常见问题

- **在光线跟踪中如何实现置换映射？**

这没有理想的方式。生成所有的三角形并在必要的时候对几何形体进行缓存可以防止内存超载（Pharr & Hanrahan, 1996; Pharr, Kolb, Gershbein, & Hanrahan, 1997）。当对置换函数进行限制后，直接与置换后的表面求交也是可能的（Patterson, Hoggar, & Logie, 1991; Heidrich & Seidel, 1998; Smits, Shirley, & Stark, 2000）。

- **为什么用纹理得到的图像看起来不真实？**

人类善于看到表面上小的瑕疵。计算机用纹理图生成的图像中通常没有几何瑕疵，所以它们看起来“太光滑”了。

- **如果在一个屏幕像素中有多个纹理像素是可见的，这时模拟的纹理映射看起来很糟糕。这应该怎么办？**

这里的问题是图像的纹理分辨率太高了。如果用一个更小的向下采样得到的纹理会更好。然而，如果我们靠得更近，向下采样的纹理看起来也将是模糊的。我们真正需要的是能够根据观察条件自动地选择纹理分辨率，这样从一个屏幕像素只能看到一个纹理像素。常用的方式是使用MIP映射（L. Williams, 1983）。这种技术建立一个多分辨率纹理集合，并为每一个多边形或者像素选择一个纹理。通常分辨率按照二次方变化，如512^2、256^2、128^2等。

文献注记

关于正确的透视纹理的讨论来自论文*Fast Shadows and Lighting Effects Using Texture Mapping* (Segal, Korobkin, Widenfelt, Foran, & Haeberli, 1992) 和*3D Game Engine Design* (Eberly, 2000)。 256

习题

1. 列出用表面和实体技术实现二维无限棋盘图案的几种方式。哪种最好？
2. 使用蛮力代数法证明式（11-4）是正确的。
3. 如何使用z缓冲器深度和矩阵变换实现实体纹理映射？ 257

第12章 完整的图形流水线

目前我们已经学习了如何对三角形进行光栅化，如何使用变换矩阵和z缓冲器/BSP树创建三维三角形的透视视图。尽管这是大多数现代图形系统的核心，但是在系统完成之前还有许多的细节要处理。目前还没有处理某些或者所有的三角形在视体外的情况；这是通过称为“裁剪”的过程完成的；三角形在视体外面的部分被剪掉或者被“裁剪”(Sutherland et al., 1974; Cyrus & Beck, 1978; J. Blinn & Newell, 1978; Liang & Barsky, 1984)。本章中其他重要的细节是关于如何提高图形流水线的效率和绘制效果。

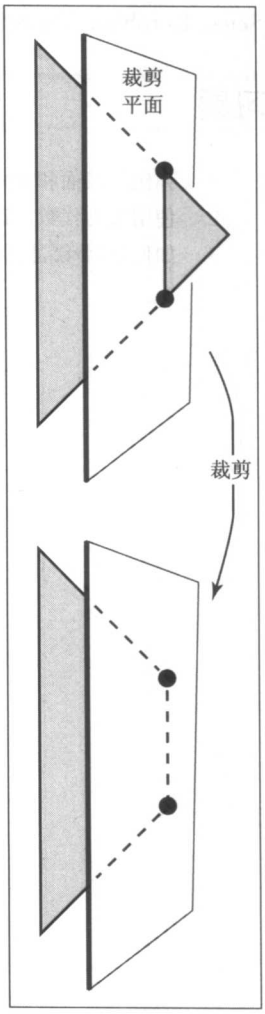


图12-1 多边形被裁剪平面裁剪。在平面“内”的部分保留下来

12.1 裁剪

在图形学中一个常用的操作就是裁剪（clipping），即用一个集合元素“切割”另一个。例如，如果三角形被平面$x = 0$裁剪，即该平面切割三角形。在大部分裁剪的应用中，三角形在平面“错”的一边的部分被删除。这里错误的一边是由应用细节指定的。对单平面的操作如图12-1所示。

本节讨论裁剪模块的基本实现。对达到工业速度的裁剪程序的实现感兴趣的读者应该参考Blinn的书，列在本章后面的文献注记中。

259

12.2 流水线中裁剪部分的位置

基本的图形流水线都采用三角形，其顶点用世界坐标系表示，并且

- 对每一个顶点乘以一个变换矩阵。
- 顶点的每一项除以它的齐次坐标。
- 对三角形进行光栅化。

260

裁剪的最大问题是在流水线的什么地方完成。图12-2中显示了所有可能的地方：

(1) 在使用6个围成平截面的观察金字塔的平面的世界坐标系中。

(2) 在用齐次坐标除之前的四维变换空间中。

(3) 在变换后三维空间（用平行于坐标轴的平面）中。

所以这些方式都可以高效地实现(J. Blinn, 1996)。对于所有的方式，基于三角形的实现中对单个三角形的处理是

```
for each of six planes do
  if (triangle entirely outside of plane) then
```

```
    break (triangle is not visible)
  else if triangle spans plane then
    clip triangle
    if (quadrilateral is left) then
      break into two triangles
```

唯一的问题是在流水线的什么阶段用哪6个平面。

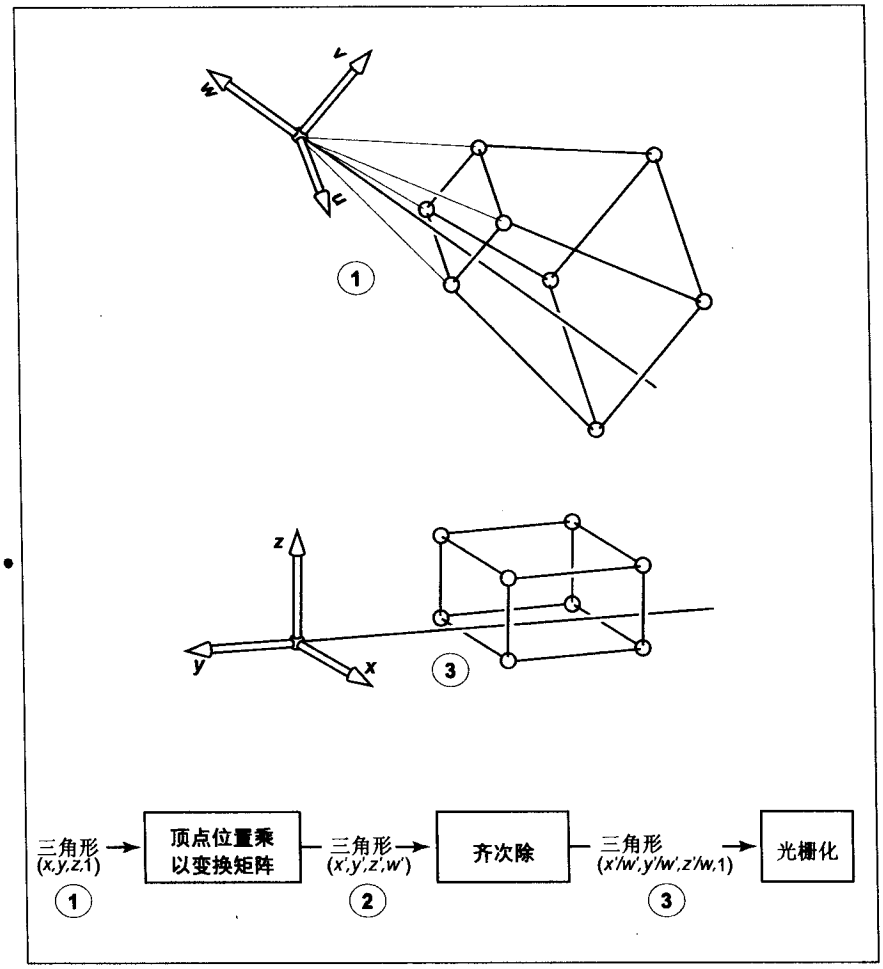


图12-2 一个基本的图形流水线，其中进行裁剪的位置有三种可能。与选项1和选项3相关的图示在流水线上方。与选项2相关的图示是四维的，所以很难描述

12.2.1 在透视除法后的裁剪（选项 3）

乍看起来，选项3是最易于实现也是效率最高的。6个平面方程很简单，计算的效率也很高：

$$-x+l=0$$
$$x-r=0$$
$$-y+b=0$$
$$y-t=0$$
$$-z+n=0$$
$$z-f=0$$

这些平面方程对任意位于视体外面的点设为正。然而，由于一个微妙的原因，选项3实际上是有问题的（见图12-3）。尽管深度大于零时，透视变换的确保持了深度顺序，但它在零深度处是不连续的。回忆一下齐次化除法后实际的变换：

261

$$z' = n + f - \frac{fn}{z}$$

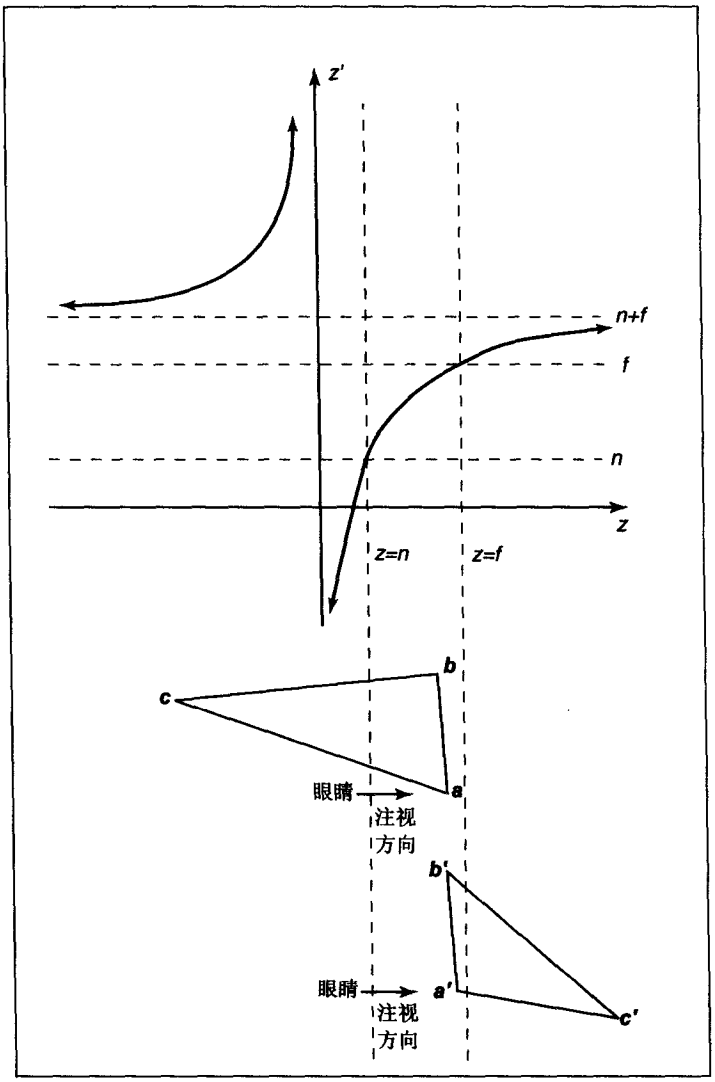


图12-3　透视变换把深度z变换为深度z'。注意当z从正移动到负时，z'从负变成正。这样眼睛后面的顶点越过$z' = n + f$移动到眼睛前面。这使裁剪（如对图中三角形的裁剪）变得很复杂

12.2.2　在变换前裁剪（选项1）

选项1的实现很简单。唯一的问题是，“6个平面方程是什么？”因为这些方程对于所有在同一图像中绘制的三角形是相同的，所以不需要进行高效的计算。为此，可以逆转图5-11中的变换，并把它应用到变换后视体的8个顶点上：

$$(x,y,z)=(l,b,n)$$
$$(r,b,n)$$
$$(l,t,n)$$
$$(r,t,n)$$
$$(l,b,f)$$
$$(r,b,f)$$
$$(l,t,f)$$
$$(r,t,f)$$

由此可以推出平面方程。还可以使用向量几何直接从观察参数中得到平面方程。

12.2.3 在齐次坐标系中裁剪（选项2）

令人惊讶的是，最常用的是在除法之前在齐次坐标系中进行裁剪。这里视体是四维的，它被几个三维体（超平面）包围。它们是：

$$-x+lw=0$$
$$x-rw=0$$
$$-y+bw=0$$
$$y-tw=0$$
$$-z+nw=0$$
$$z-fw=0$$

这些平面非常简单，所以效率比选项1要好。把视体$[l,r]\times[b,t]\times[n,f]$变换到$[0,1]^3$中，效率还可以进一步提高。这说明在四维中对三角形的裁剪并不比在三维中复杂多少。

262 ~ 263

12.2.4 用平面裁剪

无论我们选择哪个选项，都必须用平面进行裁剪。回想2.7.2节中，通过点$\boldsymbol{q}$、法线为$\boldsymbol{n}$的平面的隐式方程为

$$f(\boldsymbol{p})=\boldsymbol{n}\cdot(\boldsymbol{p}-\boldsymbol{q})=0$$

它常常写成下列形式：

$$f(\boldsymbol{p})=\boldsymbol{n}\cdot\boldsymbol{p}+D=0 \qquad (12\text{-}1)$$

有趣的是，这个方程不仅描述了一个三维平面，它还描述了一条二维直线和类似四维平面的一个体。通常所有这些元素在其相应的维度中都被称为平面。

假设有一条线段，其端点为$\boldsymbol{a}$和$\boldsymbol{b}$，可以用8.1.3节中描述的BSP树程序中裁剪三维三角形边的技术，用平面对该线段进行“裁剪”。这里要检测点$\boldsymbol{a}$和$\boldsymbol{b}$是否在平面$f(\boldsymbol{p})=0$的两侧，方法是检查$f(\boldsymbol{a})$和$f(\boldsymbol{b})$是否异号。通常$f(\boldsymbol{p})<0$定义为在平面“内”，$f(\boldsymbol{p})>0$定义为在平面“外”。如果平面确实分割该线段，可以按照下面的方法求交点：把参数直线方程，

$$\boldsymbol{p}=\boldsymbol{a}+t(\boldsymbol{b}-\boldsymbol{a})$$

代入公式（12-1）中的平面$f(\boldsymbol{p})=0$，得

$$\boldsymbol{n}\cdot(\boldsymbol{a}+t(\boldsymbol{b}-\boldsymbol{a}))+D=0$$

求解t得到

$$t=\frac{\boldsymbol{n}\cdot\boldsymbol{a}+D}{\boldsymbol{n}\cdot(\boldsymbol{a}-\boldsymbol{b})}$$

这样就可以得到交点，并把线段“缩短”。

为了对三角形进行裁剪，可以再次按照8.1.3节中的方法生成一个或两个三角形。

12.3　扩展的图形流水线

为了形成一个完整的系统，通常完整的图形流水线要添加阴影计算和一些提高效率的模式。
264 通常要对流水线进行优化，以绘制大量的小三角形。本节讨论在流水线中必须要处理的几个问题。

12.3.1　剔除

当整个三角形都位于视体外面时，可以把它剔除（culling），即从流水线中删掉。实际上，完美的剔除比裁剪模块删除物体的代价更高。当物体由许多三角形组成并有相应的包围体时，剔除是特别有用的。如果包围体位于视体外，那么所有组成该物体的三角形都在外面。例如，如果有1000个三角形，被一个中心为c半径为r的球包围，可以检查球是否在下列裁剪平面外面：

$$(p-a)\cdot n=0$$

其中a是平面上的点，p是一个变量。这等价于检查从球体中心c到平面的有向距离是否大于$+r$。这实际就是下列的检测：

$$\frac{(c-a)\cdot n}{\|n\|}>r$$

注意即使所有的三角形都位于平面外面，球体也可能与平面相交。所以这是一个保守的检测。检测的保守程度依赖于球体包围物体的紧密度。

12.3.2　光照和明暗处理

光照和明暗处理必须放在流水线中的某个位置。按照惯例，在流水线的早期，在顶点上进行光照计算，顶点间的像素在光栅化阶段使用重心插值设置颜色。然而，如果在光栅化阶段使用法线插值，那么把光照推迟到光栅化阶段就更有意义了。这是大多数现代流水线的趋势。

12.4　背向面删除

如果多面体是闭合的，即它们包围了一个封闭的空间并且没有洞，通常假设它们具有朝外的法线向量，正如在第9章讨论的。对于这样的模型，背离眼睛的多边形显然会被朝向眼睛的多边形遮挡。所以在流水线开始之前，就可以剔除这些多边形。检测这种情况的方法和9.3.1节中画轮
265 廓线的方法相同。

12.5　三角形条带和三角形扇区

现代图形流水线中两个基本的要素是三角形条带（triangle strip）和三角形扇区（triangle fan）。图12-4显示的是一个三角形扇区。在一个简单的绘制三角形的流水线中，需要的调用是：

draw (p_0, p_1, p_2)
draw (p_0, p_2, p_3)
draw (p_0, p_3, p_4)
draw (p_0, p_4, p_5)

注意尽管只有6个不同的顶点，但是这需要向流水线中传递12个顶点。三角形扇区假设有一个轴心顶点，一系列其他顶点就像折扇的顶端一样扫描过去。函数调用大概是

$$\text{triangle-fan}(p_0, p_1, p_2, p_3, p_4, p_5)$$

假设第一个顶点是轴心。API常常对每一个顶点都有一个函数调用，这样调用中就不需要参数的个数。

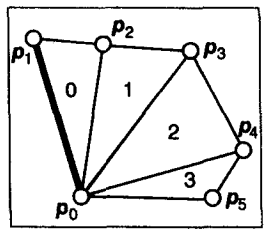


图12-4 三角形扇区

三角形条带也是类似的思想，但是它是为更传统的网格设计的。这里，在一个直线形条带中的顶点是上下交替地添加进来的，如图12-5所示。如果程序是基于顶点的，较长的三角形条带将节省大约三倍的时间。

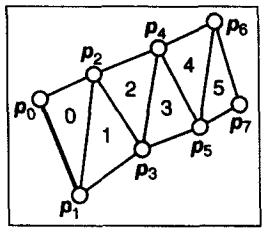


图12-5 三角形条带

看起来只有当条带很长时三角形条带才是有用的，然而事实并不是这样。如果运行时间与传递的顶点数目成比例，节省的时间如下：

条带长度	1	2	3	4	5	6	7	8	16	100	∞
相对时间	1.00	0.67	0.56	0.50	0.47	0.44	0.43	0.42	0.38	0.34	0.33

实际上随着条带增长，时间会迅速降低，但实践中最优的条带大小不应该超过硬件中缓冲器的大小。因而，即使对于非结构化的网格，使用某些贪心算法把它们组合成短的条带也是值得的。

12.6 保留状态

当为了扫描转换和光照进行顶点处理时，有必要知道明暗处理的模式，例如，光照打开还是关闭。另外有必要知道每个顶点的颜色和/或材质性质。许多API支持在三角形间共享状态信息。 266
例如，不共享状态信息的实现可能有下列调用：

```
set-triangle-attributes T1
draw-triangle T1
set-triangle-attributes T2
draw-triangle T2
```

这里图形流水线对T_1和T_2都要进行处理，处理时分别用各自的属性集合，如顶点颜色。相反，在支持共享特征的API或者硬件中，可以为通过流水线的所有三角形设置状态：

```
set-state-triangle-attributes
draw-triangle T1
draw-triangle T2
```

这里两个三角形用同样的属性绘制。在某些硬件上可以显著提高效率。

另一要保存的状态是几何模型本身，它在某些应用（应用中大部分几何模型在相邻帧间不发生变化）中是非常有用的。几何模型保存在显示列表中。由于下列两个原因，它可以显著地提高效率。第一，如果硬件允许，显示列表可以存在于图形布告板上，而不需要在每一帧都通过总线从主内存传递到图形布告板上。第二，可以对列表进行优化以提高性能。例如，共享顶点性质或明暗处理性质的三角形可以分为一组，这样它们就可以出现在共享属性的一组调用中。

12.7 完整的图形流水线

在裁剪部分曾提到，流水线可以按照很多方式安排。为了获得更高质量的视觉效果，现代的

趋势似乎是把明暗处理移到光栅化阶段。这样的流水线显示在图12-6中。

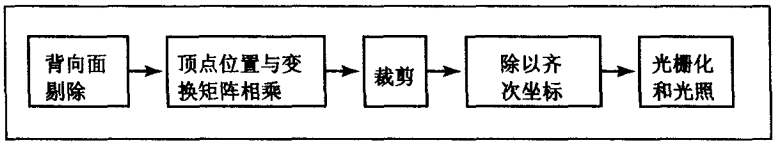


图12-6　一个完整的图形流水线。多边形从左边输入，逐次通过流水线的各个阶段

完整的流水线中还有很多问题必须要处理，程序员应该知道什么时候效率下降了。否则，它
267 们应该主要是在硬件设计者的领域。这包括：

- 多边形直接在流水线中进行光栅化还是在流水线开始时先进行三角形化？
- 纹理是如何处理的？它们是否保存在多个细节层次中（如MIP-maps（L. Williams, 1983））？是否有固定大小的纹理内存？

常见问题

- **由我经常看到关于裁剪的详细讨论，而且它是比本章中描述的复杂得多的过程。这是怎么回事？**

本章中描述的裁剪也起作用，但是缺少优化，而优化是产业级的裁剪程序中必须有的。这些优化在本章文献注记中列出的Blinn的权威著作中有详细的讨论。

- **我的API的文档中谈到了“场景图”和“矩阵栈”。它们是图形流水线的一部分吗？**

图形流水线在设计时当然考虑了这些，而我们是否把它们定义为流水线的一部分是个人喜好的问题。本书中将相关的讨论放在下一章。

文献注记

一本关于图形流水线设计的好书是*Jim Blinn's Corner: A Trip Down the Graphics Pipeline*（J. Blinn, 1996）。许多关于流水线和剔除的细节在*3D Game Engine Design*（Eberly, 2000）和*Real-Time Rendering*（Akenine-Möller & Haines, 2002）中。

习题

1. 假设在透视变换中有$n=1$，$f=2$。在什么样的条件下将会出现一个“反转”，即顶点在透视变换前后从眼睛前面跳到了眼睛后面，反之亦然？

268 2. 有没有什么理由不在透视除法后对x和y进行裁剪（见图11-2中的第三阶段）？

第13章

图形学的数据结构

在图形学应用中似乎总会反复出现很多种数据结构。本章讨论三种基本的且不相关的数据结构，它们是最常见、最有用的数据结构。数据结构有很多，但是其背后的基本思想可以用一个例子来说明。

首先讨论用于存储镶嵌几何模型的翼边数据结构（Baumgart, 1974）。翼边数据结构对于管理镶嵌格发生变化的模型是很有用的，如在子分或者简化过程中发生变化。

其次，介绍场景图数据结构。因为它们在管理物体和变换中非常有用，所以它们迅速成为所有新的图形API完全支持的特征。

最后，介绍平铺多维数组。最初是为提高分页性能而设计的，现在这种数据结构对于机器上内存的定位方式是至关重要的，无论数组是否与内存相适合。

13.1 三角形网格

最常用的模型表示方法之一是多边形网格，正如11.3节中讨论的。当网格在程序中保持不变时，11.3节中描述的简单数据结构通常是足够的。然而，当要修改网格时，就需要更复杂的数据结构提高查询的效率，查询包括：

269

- 给定一个三角形，三个相邻的三角形是什么？
- 给定一条边，哪两个三角形共享这条边？
- 给定一个顶点，哪些面共享该顶点？
- 给定一个顶点，哪些边共享该顶点？

有许多数据结构可以表示三角形网格、多边形网格和带洞的多边形网格（参考资料见本章最后的文献注记）。在很多应用中网格是非常大的，所以一个高效的表示方法是非常重要的。

最简单的，尽管是臃肿的，实现只有三种特征：顶点、边和三角形。有很多方式可以把数据分成这样的特征。如果只想保存所有的关系，将产生一个变长的数据结构，这实在是没有必要的。例如，一个顶点可能有任意多的边与之关联。

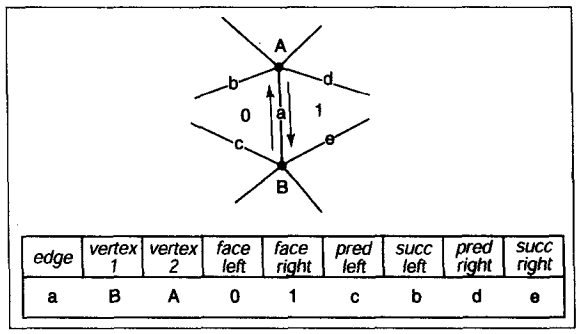
所以，最好把实现隐藏在类的接口里面。

13.2 翼边数据结构

我们可以使用*winged-edge*类这样的数据结构。这种数据结构把边作为数据结构中最重要的元素。图13-1和图13-2说明了这种数据结构和一个效率更高的实现机制。

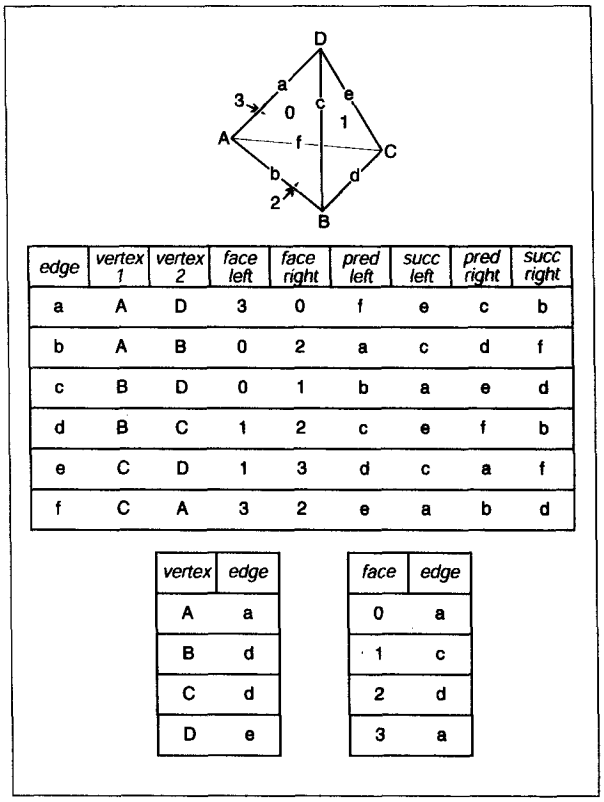
270

注意翼边数据结构可以在常数时间内实现指定的查询。例如，一个面可以访问它的一条边，并可以沿着遍历指针找到其所有的边。那些边又存储了相邻的面。



edge	vertex 1	vertex 2	face left	face right	pred left	succ left	pred right	succ right
a	B	A	0	1	c	b	d	e

图13-1　翼边数据结构中的一条边。每一条边存储其左边和右边的面（多边形）和遍历这两个面时对应的前驱边和后继边



edge	vertex 1	vertex 2	face left	face right	pred left	succ left	pred right	succ right
a	A	D	3	0	f	e	c	b
b	A	B	0	2	a	c	d	f
c	B	D	0	1	b	a	e	d
d	B	C	1	2	c	e	f	b
e	C	D	1	3	d	c	a	f
f	C	A	3	2	e	a	b	d

vertex	edge
A	a
B	d
C	d
D	e

face	edge
0	a
1	c
2	d
3	a

图13-2　一个四面体和其翼边数据结构中的相关元素。两个小的表并不是唯一的；每个顶点和面存储了所有与之关联的边

与其他数据结构相比，翼边数据结构从时间和空间上进行了几种折衷。例如，我们可以删除*prev*索引。当需要知道前驱边时，可以沿着后继边走一圈，直到回到起始的边。这将会节省空间，但是计算前驱边的时间会增长。这类问题在网格数据结构中将会产生扩散（关于那些数据结构的更多信息，见本章的文献注记）。

271

13.3　场景图

为了引出场景图数据结构，我们将用一个铰接的钟摆作为例子，如图13-3所示。考虑一下如

何画钟摆的上一部分：

M_1 = rotate(θ)
M_2 = translate(p)
$M_3 = M_2M_1$
Apply M_3 to all points in upper pendulum

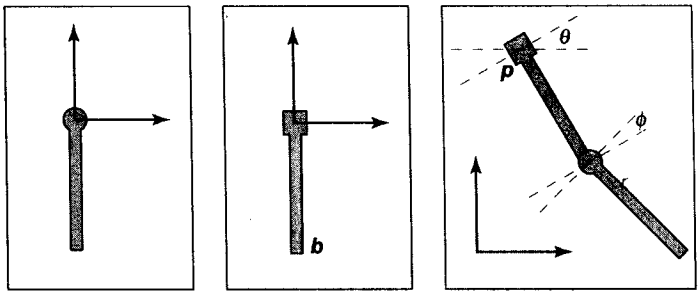


图13-3 一个铰接的钟摆。左边是在“局部”坐标系中的两部分。上一部分的铰链在点b，与下一部分的连接点在其局部原点。组合物体的自由度是角度(θ,ϕ)和上端铰链的位置p

下面一部分更复杂，但是我们可以利用这个事实，即它与钟摆上部的底端相连，连接点为局部坐标系中的b点。首先旋转钟摆下部，使之相对于初始位置的角度为ϕ。然后移动钟摆下部，使其上端铰链位于点b。现在它在钟摆上部的局部坐标中适当位置，并且可以随着该坐标系统移动。对钟摆下部的组合变换如下：

M_a = rotate(ϕ)
M_b = translate(b)
$M_c = M_bM_a$
$M_d = M_3M_c$
Apply M_d to all points in lower pendulum

如此，我们看到钟摆下部不仅在其自己的局部坐标系中，该坐标系本身也随着钟摆上部移动。 272

我们可以用一个数据结构对钟摆进行编码，该结构可以使管理坐标系的问题变得更简单，如图13-4所示。应用在物体上正确的矩阵就是从物体到数据结构中根结点的链上所有矩阵的乘积。例如，假设有一个渡轮模型，它包含一辆在渡轮甲板上自由移动的汽车，每一个车轮相对于汽车移动，如图13-5所示。

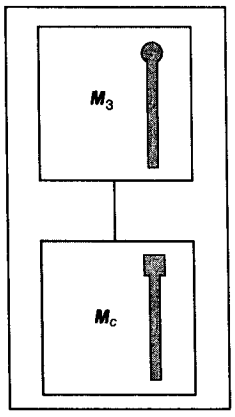


图13-4 图13-3中铰接钟摆的场景图

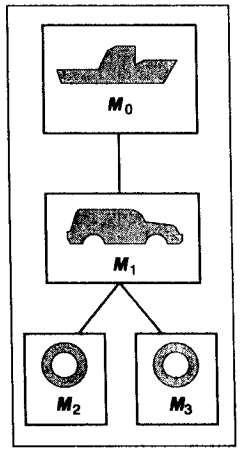


图13-5 一个渡轮、渡轮上的一辆汽车和汽车的车轮（只显示了两个）存储在一个场景图中

像钟摆一样，每一个物体都应该用从根结点到物体的路径上所有矩阵的乘积进行变换：

ferry transform using M_0

car body transform using M_0M_1

left wheel transform using $M_0M_1M_2$

left wheel transform using $M_0M_1M_3$

使用矩阵栈，实现的效率会更高，许多API都支持这种数据结构。矩阵栈是通过使用push和pop操作来控制的，这些操作从矩阵乘积的右边添加或删除矩阵。例如，调用

```
push(M0)
push(M1)
push(M2)
```

创建了当前矩阵$M = M_0M_1M_2$。随后的调用*pop*()去掉了最后添加的矩阵，所以当前矩阵变成了：$M = M_0M_1$。把矩阵栈和对场景图的递归遍历结合在一起得到

```
function traverse(node)
push(Mlocal)
draw object using composite matrix from stack
traverse(left child)
traverse(right child)
pop()
```

273

场景图有许多变型，但是都遵守上述基本思想。

13.4　平铺多维数组

在设计现代体系结构下的算法时，有效地利用缓存层次是非常重要的任务。要使多维数组的数据有非常“漂亮”的排列，可以通过平铺实现，有时也称为分块。传统的二维数组是用一维数组存储的，同时它具有一个索引机制；例如一个$N_x \times N_y$的数组存储在长度为N_xN_y的一维数组中，二维索引(x, y)（索引是从$(0, 0)$到(N_x-1, N_y-1)）可以用下列公式映射到一维索引（从0到N_xN_y-1）：

$$索引 = x + N_x y$$

图13-6是内存如何排列的一个例子。这种排列存在的问题是，尽管在同一行的相邻元素在内存中是相邻的，但在同一列中的相邻元素在内存中将间隔N_x个元素。对于较大的N_x，这将使内存定位的效率很低。标准的解决方法是利用平铺使内存对行和列的定位方式更一致。图13-7是一个例子，其中使用了2×2的平铺单元。这种数组索引机制的细节将在下一节中讨论。之后将讨论一个更复杂的例子，即三维数组的两层平铺。

关键的问题使如何确定平铺单元的大小。实际中，它应该和机器上内存单元的大小相同。例如，在有128字节缓存、使用16位数据的机器上，n恰好是8。然而，如果使用浮点（32位）数据集的话，n接近5。因为某些内存单元的大小是近似的，如页，具有类似逻辑的分层平铺可能是很有用的。

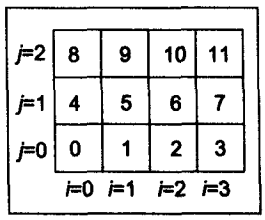


图13-6　非平铺的二维数组的内存排列，$N_x = 4$，$N_y = 3$

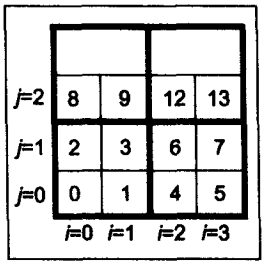


图13-7　平铺的二维数组的内存排列和2×2的平铺单元，$N_x = 4$，$N_y = 3$。注意因为y不是单元大小2的整数倍，所以需要在数组上面进行填补

13.4.1 二维数组的一层平铺

如果假设一个$N_x \times N_y$的数组分解成$n \times n$的正方形平铺单元（图13-8），那么所需的平铺单元数是

$$B_x = N_x / n$$
$$B_y = N_y / n$$

这里假设n恰好能整除N_x和N_y，否则需要对数组进行填补。例如，如果$N_x = 15$、$n = 4$，那么N_x应该变成16。为了得到数组的索引公式，首先找到平铺单元的索引(b_x, b_y)，它给出了平铺单元的行和列（平铺单元本身形成一个二维数组）：

$$b_x = x \div n$$
$$b_y = y \div n$$

274

其中÷表示整数除法，例如$12 \div 5 = 2$。如果按照图13-6中的方式按行排列平铺单元，那么单元(b_x, b_y)第一个元素的索引是

$$索引 = n^2(B_x b_y + b_x)$$

单元内的内存按照传统的二维数组排列，如图13-7。单元中的局部偏移(x', y')是

$$x' = x \bmod n$$
$$y' = y \bmod n$$

其中mod是求余操作，例如$12 \bmod 5 = 2$。所以单元内的偏移是

$$偏移 = y'n + x'$$

所以，在具有$n \times n$平铺单元的$N_x \times N_y$数组中，查找一维索引元素(x, y)的完整公式是

$$\begin{aligned} 索引 &= n^2(B_x b_y + b_x) + y'n + x' \\ &= n^2((N_x \div n)(y \div n) + x \div n) + (y \bmod n)n + (x \bmod n) \end{aligned}$$

这个表达式中包含很多整数乘法、除法和求余运算。在现代处理器中，这些运算是非常费时的。如果n是2的幂，这些运算可以转换成移位运算和按位逻辑运算。然而，就像上面提到的，理想的大小并不总是2的幂。某些乘法可以转换成移位或加法运算，但是除法和求余运算就更有问题了。应该以递增的方式计算索引，但是这需要跟踪计数器，需要大量的比较，并将降低分支预测的性能。

275

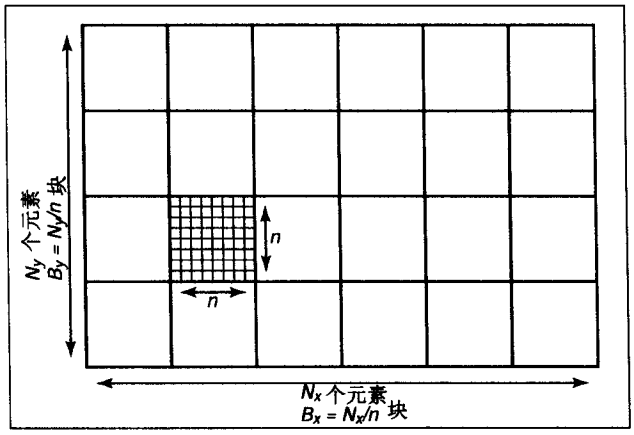


图13-8 由$B_x \times B_y$个单元组成的平铺二维数组，每一个单元的大小是$n \times n$

然而，有一个简单的解决方法。注意索引表达式可以写成

$$索引 = F_x(x) + F_y(y)$$

其中

$$F_x(x) = n^2(x \div n) + (x \bmod n)$$
$$F_y(y) = n^2(N_x \div n)(y \div n) + (y \bmod n)n$$

我们把F_x和F_y作成表格，利用x和y找到数据数组中的索引。表格全部的大小应该和处理器的基本数据缓存相符合，即使是对于巨大的数据集也应该这样。

13.4.2　示例：三维数组的两层平铺

在算法性能上，高效地使用TLB也变成一个非常重要的因素。通过创建具有$n \times n \times n$个单元的$m \times m \times m$个分块，同样的技术可以用来提高TLB在三维数组中的命中率。例如，一个40×20×19的体可以分解为4×2×2个大块，每个大块有2×2×2个小块，每个小块有5×5×5个单元。这对应于$m=2$，$n=5$。因为19不能分解为因子$mn=10$，所以需要对一层进行填补。经验大小是，对于16位的数据集，$m=5$，对于浮点数据集，$m=6$。

对于任意的三元组(x, y, z)，可以用下面的公式计算数据数组中的索引：

$$\begin{aligned}索引 = {} & ((x \div n) \div m)n^3m^3((N_z \div n) \div m)((N_y \div n) \div m) \\ & +((y \div n) \div m)n^3m^3((N_z \div n) \div m) \\ & +((z \div n) \div m)n^3m^3 \\ & +((x \div n) \bmod m)n^3m^2 \\ & +((y \div n) \bmod m)n^3m \\ & +((z \div n) \bmod m)n^3 \\ & +(x \bmod (n^2))n^2 \\ & +(y \bmod n)n \\ & +(z \bmod n)\end{aligned}$$

276 其中N_x、N_y和N_z是数据集各自的大小。

注意，在简单的二维一层的情况下，这个公式可以写成

$$索引 = F_x(x) + F_y(y) + F_z(z)$$

其中

$$\begin{aligned}F_x(x) = {} & ((x \div n) \div m)n^3m^3((N_z \div n) \div m)((N_y \div n) \div m) \\ & +((x \div n) \bmod m)n^3m^2 \\ & +(x \bmod n)n^2 \\ F_y(y) = {} & ((y \div n) \div m)n^3m^3((N_z \div n) \div m) \\ & +((y \div n) \bmod m)n^3m \\ & +(y \bmod n)n \\ F_z(z) = {} & ((z \div n) \div m)n^3m^3 \\ & +((z \div n) \bmod m)n^3 \\ & +(z \bmod n)\end{aligned}$$

常见问题

- **平铺真的能在性能上有这么大的区别吗？**

在某些体绘制应用中，两层的平铺策略可以使性能上有10倍的差异。而在某些应用中，如图像编辑系统，当数组与主存不相匹配时，它可以有效地防止系统失败。

- **如何存储翼边结构中的列表？**

对于大多数应用而言，使用数组和索引是很方便的。然而，如果要执行很多删除操作，明智的方式是使用链表和指针。

文献注记

关于翼边数据结构的论述是以*Ching-Kuang Shene*（Shene, 2003）的课程笔记为基础的。除了翼边结构还有更小的网格数据结构。在*Directed Edges—A Scalable Representation for Triangle Meshes*（Campagna, Kobbelt, & Seidel, 1998）中讨论了如何折衷使用这些数据结构。关于平铺数组的论述是以*Interactive Ray Tracing for Volume Visualization*（Parker, Martin, et al., 1999）为基础的。

277

习题

1. 对于一个简单的四面体，存为4个独立的三角形和存为翼边数据结构，内存有多大差异？
2. 画出一辆自行车的场景图。
3. 对于一个n维数组的单层平铺需要多少查找表？

278

第14章 采 样

许多图形学应用要求在特定的空间中进行“合理的”采样，如在所有可能的直线空间中。例如，我们可能需要在一个像素内生成随机边，或在一个像素上生成随机采样点，这些点根据某些密度函数拥有不同的密度。本章提供了一些概率运算的机制。这些方法同时被证明对于使用本章中提到的蒙特卡罗积分法计算复杂积分的值是有效的。

14.1 积分

尽管“积分”和“度量”通常认为是十分困难的，但它们涉及的是数学中最基本的概念，所以是不值得害怕的。对于不要求精度的用途，度量仅仅是将子集映射到$\mathbb{R}^+$上的函数，在某种意义上和长度、面积和体积的直观概念一致。例如，在二维实平面$\mathbb{R}^2$上，面积度量A就是给平面上的一个点集赋值。注意，A仅仅是一个函数，输入平面的一部分并返回平面面积。A的定义域是
279 集合$\mathbb{R}^2$所有可能的子集，定义为幂集$\mathcal{P}(\mathbb{R}^2)$。所以可以用箭头表示法描述$A$的特性：

$$A:\mathcal{P}(\mathbb{R}^2)\rightarrow\mathbb{R}^+$$

应用面积度量的例子如下，它表示边长为1的正方形的面积为1：

$$A([a,a+1]\times[b,b+1])=1$$

其中(a,b)是正方形左下角顶点的坐标。注意，单一的点例如（3, 7）是$\mathbb{R}^2$一个正确的子集，它的面积是0，即$A((3,7))=0$。同样结论对于x轴上点集S也成立，即$S=(x,y)$，$(x,y)\in\mathbb{R}^2$的面积$A(S)=0$。这样的集合称为零度量集合（zero measure set）。

函数必须服从一些与面积类似的性质，才能称作度量。例如，定义一个函数$\mu:\mathcal{P}(\mathrm{S})\rightarrow\mathbb{R}^+$。为使$\mu$成为一个度量，必须满足下列几个条件：

(1) 空集的度量值为0：$\mu(\phi)=0$

(2) 两个不相交的集合的并集的度量值等于它们各自度量值之和。如果两个集合可能相交，则用如下规则表示：

$$\mu(A\cup B)=\mu(A)+\mu(B)-\mu(A\cap B)$$

其中∪表示集合的并运算，∩表示集合的交运算。

实际计算度量值时，通常使用积分法。可以把积分看作是表示法：

$$A(S)\equiv\int_{x\in S}\mathrm{d}A(x)$$

公式右边可以简单理解为“取区域S中所有的点x，并把它们有关的微分面积相加。”积分通常还表示为以下几种形式：

$$\int_S\mathrm{d}A,\quad\int_{x\in S}\mathrm{d}x,\quad\int_{x\in S}\mathrm{d}Ax,\quad\int_x\mathrm{d}x.$$

以上所有的公式都表示“区域S的面积”。在以后的讨论中将使用第一个公式，因为它最长，可以避免歧义。为了用解析法求解积分的值，通常需要弃用一些坐标，同时使用大量的微积分技巧来求解方程式。不过即使忘记了这些技巧也不用感到恐慌，由于通常只要求数值上近似的积分计算，因此只要少量简单的技巧，这些技巧在本章的后续章节中将有所介绍。

假设集合$\mathbb{S}$上的一个度量，通常可以通过一个非负加权函数$\omega:\mathbb{S}\to\mathbb{R}^+$构造一个新的度量。用积分表示法可以最好地表示这种方法。例如，我们以$[0,1]^2$区域的面积为例开始介绍这种方法： 280

$$\int_{x\in[0,1]^2} \mathrm{d}A(\boldsymbol{x})$$

通过插入半径的平方这一加权函数，可以使用“径向加权”度量：

$$\int_{x\in[0,1]^2} \|\boldsymbol{x}\|^2\,\mathrm{d}A(\boldsymbol{x})$$

为了解析地求解积分的值，可以利用笛卡儿坐标系的坐标，用$\mathrm{d}A\equiv\mathrm{d}x\mathrm{d}y$对积分展开：

$$\int_{x\in[0,1]^2} \|\boldsymbol{x}\|^2\,\mathrm{d}A(\boldsymbol{x})=\int_{x=0}^{1}\int_{y=0}^{1}(x^2+y^2)\mathrm{d}x\,\mathrm{d}y$$

这里的关键点是，如果认为$\|\boldsymbol{x}\|^2$项与$\mathrm{d}A$项相结合，那么两项结合就构成了一个新的度量，记做度量υ。在书写时，可以使用$\upsilon(S)$替代整个积分式。如果只把它们当作是一串符号和簿记就对了。不过它的确允许按照各自的喜好把公式写成简洁的或扩展的形式。

14.1.1 度量和均值

当对一个函数求均值时，度量才真正开始发挥作用。可以只计算关于一个特定度量的平均值，并选取一个相对于应用或定义域来讲“自然的”度量。一旦选定了度量，函数f在区域S上关于度量μ的平均值为

$$\mathrm{average}(f)\equiv\frac{\int_{x\in S} f(\boldsymbol{x})\mathrm{d}\mu(\boldsymbol{x})}{\int_{x\in S}\mathrm{d}\mu(\boldsymbol{x})}$$

例如，函数$f(x,y)=x^2$在$[0,2]^2$上关于面积度量的平均值为

$$\mathrm{average}(f)\equiv\frac{\int_{x=0}^{2}\int_{y=0}^{2}x^2\mathrm{d}x\,\mathrm{d}y}{\int_{x=0}^{2}\int_{y=0}^{2}\mathrm{d}x\,\mathrm{d}y}=\frac{4}{3}$$

这种机制有助于解决看起来很难的问题，其中如何选择度量是一个需要技巧的部分。这样的问题经常出现在积分几何学中，这一领域研究关于几何实体的度量，如直线和平面。例如，穿过$[0,1]^2$的所有直线平均长度，用定义表示为 281

$$\mathrm{average}(\text{长度})=\frac{\int_{\text{穿过}[0,1]^2\text{的直线}L}\mathrm{length}(L)\mathrm{d}\mu(L)}{\int_{\text{穿过}[0,1]^2\text{的直线}L}\mathrm{d}\mu(L)}$$

剩下的问题，就是需要为应用选择一个合适的度量μ。这也是在下一节中对直线所做的处理。

14.1.2 示例：二维平面中直线的度量

什么样的度量μ是“自然的”？

如果把直线表示成参数形式$y=mx+b$，那么给定的直线可以看作是在“斜率-截距”空间上的一个点(m,b)。可以用一个简单的度量$\mathrm{d}m\,\mathrm{d}b$，但是由于相同规模的直线束可能具有不同的度量，

因此$dm\ db$可能不是一个好的度量。更精确地讲，如果坐标系改变，则度量一定会跟着改变。例如，如果使所有的直线穿过正方形$[0,1]^2$，那么对于穿过$[0,1]^2$的直线和穿过旋转45° 的正方形的直线的度量一定是不同的。我们真正希望得到的是“合理的”度量，它不随一组线的旋转和平移而改变。如图14-1和图14-2所示。

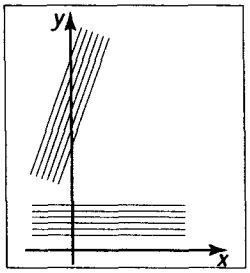


图14-1　两束直线应当采用相同的度量。由于它们与y轴交线的长度不同，所以对于微分度量，使用db并不是一个好的选择

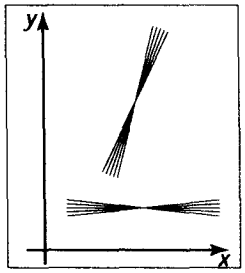


图14-2　两束直线应当采用相同的度量。由于它们的斜率变化量不同，所以对于微分度量，使用dm并不是一个好的选择

为了设计直线上自然的度量，首先把直线看作对偶空间上的点。这是一个很简单的概念：直线$y=mx+b$可以看作斜率-截距空间上的点(m,b)，如图14-3所示。而在(ϕ,b)空间上形成度量则更加简便。如图14-4所示，在(ϕ,b)空间中，b是直线在y轴的截距，ϕ是直线与x轴形成的夹角。微分度量$d\phi\ db$几乎可以满足要求，但是在图14-1所示的情况下它不是一个合理的度量。一束固定宽度的直线形成的跨度b不同，为了避免b值影响，必须增加一个余弦因子：

$$d\mu = \cos\phi\ d\phi\ db$$

可以看出这个度量相当于是一个常量，是唯一不随平移和旋转变化的。

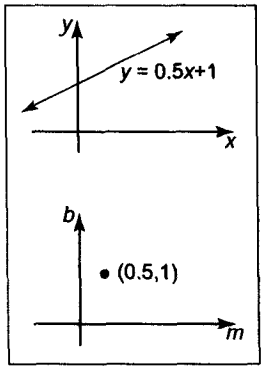


图14-3　(x,y)空间中直线$y=mx+b$上所有点的集合可以用(m,b)空间中单独的一个点来表示，因此图像上方的直线和下方的点表示的是同一个几何实体：二维空间的一条直线

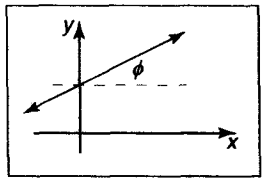


图14-4　在角度-截距空间中，可以用角度$\phi\in[-\pi/2,\pi/2)$，而不是用斜率表示直线

这个度量可以转换成适合于直线其他表示形式的度量。例如，适用于(m,b)空间的度量为

$$d\mu = \frac{dm\ db}{(1+m^2)^{\frac{3}{2}}}$$

282

在(u,v)空间中，直线的表达式是

$$ux+vy+1=0$$

适当的度量是

$$\mathrm{d}\mu = \frac{\mathrm{d}u\ \mathrm{d}v}{(u^2+v^2)^{\frac{3}{2}}}$$

对于用x轴和y轴的截距(a,b)作参数来表示的直线，度量为

$$\mathrm{d}\mu = \frac{ab\ \mathrm{d}a\ \mathrm{d}b}{(a^2+b^2)^{\frac{3}{2}}}$$

注意以上几种坐标空间对于定义直线都是等效的，哪个是最好的取决于不同的环境。然而可能会有疑问，是否存在一个坐标系，使得对一组直线的度量恰好是对偶空间的一个面积。事实上，的确有这样的一个坐标系，而且相当简单，称为正则坐标，它用原点到直线的垂直距离和直线的法线与x轴的夹角来定义直线（如图14-5所示）。这种直线的隐式方程式为

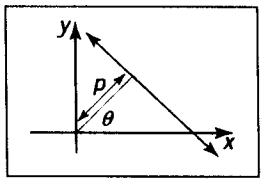


图14-5 直线的正则坐标使用原点到直线的垂直距离和直线的法线与x轴的夹角来表示直线

$$x\cos\theta + y\sin\theta - p = 0$$

在正则坐标空间中直线的度量为

$$\mathrm{d}\mu = \mathrm{d}p\ \mathrm{d}\theta$$

下一节将使用以上的度量来选择合理的随机直线。

14.1.3 示例：三维空间中直线的度量

在三维空间中，有许多用参数表示直线的方式。最简单的方式可能是用直线与某一特定平面的交点和某种对直线方向的定义。例如，可以用直线与xy平面的交点坐标和直线在球坐标系中的方向。因此，每条直线都可以用四元组(x,y,θ,ϕ)表示。这表明三维空间中的直线是四维实体，即三维空间中的直线可以用四维空间中的点表示。

直线的微分度量不应该随着(x,y)的改变而变化，但是具有等横截面的直线束应当具有相同的度量。因此，一个合理的微分度量是

$$\mathrm{d}\mu = \mathrm{d}x\ \mathrm{d}y\sin\theta\ \mathrm{d}\theta\ \mathrm{d}\phi$$

283

另一种用参数表示直线的方法是利用直线与两个平行平面的交点来表示。例如，如果直线与平面$z=0$相交于点$(x=u,y=v)$，与平面$z=1$相交于点$(x=s,y=t)$，那么可以用四元组(u,v,s,t)表示这条直线。注意，像前面的参数表示法一样，这种方法对与xy平面平行的直线是不适用的。这种参数表示法的微分度量更加复杂，尽管对于与z轴几乎平行的直线束可以用下式近似表示：

$$\mathrm{d}\mu \approx \mathrm{d}u\ \mathrm{d}v\ a\ \mathrm{d}s\ \mathrm{d}t$$

这种度量通常应用于基于图像的绘制（第25章）中。

对于与球面相交的直线，可以使用直线与球面的两个交点坐标作为参数来表示直线。在球坐标系下，点可以用四元组$(\theta_1,\phi_1,\theta_2,\phi_2)$表示，度量就是与每个点相关的微分面积：

$$\mathrm{d}\mu = \sin\theta_1\ \mathrm{d}\theta_1\ \mathrm{d}\phi_1\ \sin\theta_2\ \mathrm{d}\theta_2\ \mathrm{d}\phi_2$$

这意味着在球面上随机选择两个均匀分布的点将产生一条密度均匀的直线。Mateu Sbert在其论文中使用这一结论来计算波形因数（Sbert, 1997）。

注意，有时希望用参数来表示定向的直线，而有时希望与两个端点的次序无关。这是一个簿记的细节，尤其对于绘制应用是十分重要的，其中大量沿着直线传播的光线在沿直线的两个方向是不同的。

14.2 连续概率

大量图形学算法使用概率来构造随机采样，以解决积分和均值问题。这属于应用连续概率的范畴，它与度量理论有根本的联系。

14.2.1 一维连续概率密度函数

不严格地讲，连续随机变量（continuous random variable）x是一个标量或者矢量值，它在全体实数范围$\mathbb{R}=(-\infty,+\infty)$内“随机”取值。$x$的特性完全由它的取值分布来描述。关于$x$的概率密
284 度函数(probability density function, pdf)p（x与p的关系记为$x\sim p$）可以定量地描述取值的分布。假设x在区间$[a,b]$上取特定值的概率由下面的积分式确定：

$$\text{Probability}(x\in[a,b])=\int_a^b p(x)\mathrm{d}x \tag{14-1}$$

不严格地讲，概率密度函数p描述了一个随机变量取某一特定值的相对可能性。如果$p(x_1)=6.0$且$p(x_2)=3.0$，那么对于密度函数为p的随机变量，其取值“接近”x_1的可能性是“接近”x_2的可能性的两倍。密度函数p具有如下两个特征：

$$p(x)\geqslant 0\ \text{（概率值非负）} \tag{14-2}$$

$$\int_{-\infty}^{+\infty} p(x)\mathrm{d}x=1(\text{Probability}(x\in\mathbb{R})=1) \tag{14-3}$$

例如，标准随机变量ξ以相同的概率在0（包含）到1（不包含）之间取值，（这里相同意味着ξ取任何一个值的可能性是相等的）。ξ的概率密度函数q为

$$q(\xi)=\begin{cases}1 & \text{如果}0\leqslant\xi<1\\ 0 & \text{其他}\end{cases}$$

变量ξ的定义域为区间[0, 1)。ξ在区间$[a,b]\subseteq[0,1)$上取值的概率为

$$\text{Probability}(a\leqslant\xi\leqslant b)=\int_a^b 1\mathrm{d}x=b-a$$

14.2.2 一维期望

设一维随机变量x的概率密度函数为p，那么实函数$f(x)$的平均值称为期望，记作$E(f(x))$（有时记作$Ef(x)$）：

$$E(f(x))=\int f(x)p(x)\mathrm{d}x$$

285 一维随机变量的期望可以通过设$f(x)=x$计算。期望具有一个非常令人惊讶而又实用的性质：两个随机变量和的期望等于这两个变量期望的和，对于随机变量x和y有

$$E(x+y)=E(x)+E(y)$$

由于随机变量函数本身可以看作随机变量，因此期望值的线性规则同样对函数适用：

$$E(f(x)+g(y))=E(f(x))+E(g(y))$$

一个明显的问题是，如果相加的两个随机变量是相关的（不相关的变量称为独立的），那么这个性质是否仍然成立。实际上，无论变量是否相互独立这个线性性质都成立。相加性质对大部分的蒙特卡罗应用都是至关重要的。

14.2.3 多维随机变量

对于随机变量及其期望的讨论可以自然地扩展到多维空间。大多数图形问题都出现在高维空

间中。例如，大量的光照问题都在半球表面上处理。幸运的是，如果在随机变量占据的空间中定义一个度量μ，那么一切都与一维空间的情况非常类似。假设空间S中有一个相关的度量μ，例如S是一个球面，μ表示计算S的面积。定义一个概率密度函数$p:S\mapsto\mathbb{R}$，x是随机变量，满足$x\sim p$，则x在区域S_i（$S_i\subset S$）中取值的概率由下面的积分式确定：

$$\text{Probability}(x\in S_i)=\int_{S_i}p(x)\mathrm{d}\mu$$

这里的Probability（事件）是指事件为真的概率，因此积分表示x在集合S_i中取值的概率。

在图形学中，S通常表示一块区域（$\mathrm{d}\mu=\mathrm{d}A=\mathrm{d}x\,\mathrm{d}y$）或一组方向（即单位球面上的点：$\mathrm{d}\mu=\mathrm{d}\omega=\sin\theta\mathrm{d}\theta\,\mathrm{d}\phi$）。例如，二维随机变量$\alpha$是半径为$R$的圆盘上均匀分布的随机变量。这里均匀强调的是相对于面积的均匀，例如，一个技术很差的飞镖投手所投掷的飞镖可能分布在整个镖盘上。因为分布是均匀的，所以$p(\alpha)$是一个常数。圆盘的面积是πr^2，全概率为1，由此可以推出

$$p(\alpha)=\frac{1}{\pi R^2}$$ 286

所以α在圆盘中特定子集S_1上取值的概率为

$$\text{Probability}(\alpha\in S_1)=\int_{S_1}\frac{1}{\pi R^2}\mathrm{d}A$$

以上的积分式是十分抽象的。要实际使用它，需要将积分转换为可以进行计算的形式。假设S_i是圆盘上的一个区域。如果转换成极坐标，则α用(r,ϕ)对表示，S_1是$r<R/2$的区域。注意，仅仅因为α是均匀的，不能推断r或ϕ一定是均匀的（实际上，ϕ是均匀的，r是不均匀的）。微分面积$\mathrm{d}A$就是$r\mathrm{d}r\,\mathrm{d}\phi$。因此，

$$\text{Probability}(r<\frac{R}{2})=\int_0^{2\pi}\int_0^{\frac{R}{2}}\frac{1}{\pi R^2}r\,\mathrm{d}r\,\mathrm{d}\phi=0.25$$

多维实函数的期望是

$$E(f(x))=\int_S f(x)p(x)\mathrm{d}\mu$$

其中$x\in S$，$f:S\mapsto\mathbb{R}$，$p:S\mapsto\mathbb{R}$。例如，在单位正方形$S=[0,1]\times[0,1]$上定义函数$p(x,y)=4xy$，x的期望为

$$\begin{aligned}E(x)&=\int_S f(x,y)p(x,y)\mathrm{d}A\\&=\int_0^1\int_0^1 4x^2y\,\mathrm{d}x\,\mathrm{d}y\\&=\frac{2}{3}\end{aligned}$$

注意，这里$f(x,y)=x$。

14.2.4 方差

一维随机变量的*方差*$V(x)$的定义为x与$E(x)$差值的平方的期望，即

$$V(x)\equiv E([x-E(x)]^2)$$

通过代数运算可推导下列公式：

$$V(x)=E(x^2)-[E(x)]^2$$ 287

表达式$E([x-E(x)]^2$对于直观地考虑方差很有用，而代数上等价的公式$E(x^2)-[E(x)]^2$计算上方便。如果变量是独立的，那么随机变量和的方差等于随机变量方差的和。方差的这一求和性质是

方差被频繁用于概率模型分析中的一个原因。方差的平方根称作标准偏差σ，它给出了关于与期望之间的绝对偏差的期望信息。

14.2.5 估计均值

许多问题涉及到独立随机变量x_i的总和，其中各变量服从相同的密度函数p。这样的变量称为独立同分布（iid-independent identically distributed）的随机变量。当总和用变量的个数相除时，可以得到对$E(x)$的估计：

$$E(x) \approx \frac{1}{N}\sum_{i=1}^{N} x_i$$

随着N的增加，估计的方差会逐渐降低。我们希望N的值足够大，这样就可以保证估计值与真实值“足够近”。但是，在蒙特卡罗理论中没有确定的事，只能在统计意义上保证我们的估计是好的。只有令$N=\infty$，才能保证估计是正确的。这个置信度可以用大数定律表示：

$$\text{Probability}\left[E(x) = \lim_{N\to\infty}\frac{1}{N}\sum_{i=1}^{N} x_i\right] = 1$$

14.3 蒙特卡罗积分

本节将简要论述基本的蒙特卡罗定积分解法，随后将直接把这些方法应用到某些积分问题中。本节中所有的基本理论在几本经典的蒙特卡罗理论著作中都有所记载（见本章后面的文献注记）。

像前面讨论的一样，设函数$f:S\mapsto\mathbb{R}$，随机变量$x\sim p$，可以用下面的求和公式估计函数$f(x)$的期望值：

288

$$E(f(x)) = \int_{x\in S} f(x)p(x)\mathrm{d}\mu \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{14-4}$$

因为期望可以用一个积分表示，所以积分也可以用求和公式来估计。公式（14-4）的形式有些复杂，我们通常更希望估计单个函数g的积分，而不是乘积fp的积分。为此可以把fp置换为g（$g=fp$）作为被积函数：

$$\int_{x\in S} g(x)\mathrm{d}\mu \approx \frac{1}{N}\sum_{i=1}^{N}\frac{g(x_i)}{p(x_i)} \tag{14-5}$$

为了使公式合法，当g为非零时p必须为正数。

为了得到精确的估计，希望有尽可能多的采样，同时希望g/p具有较小的方差（g和p应该具有相似的形式）。对p进行巧妙的选取被称作重要性采样，因为如果p在g取值较大的地方取值也很大，那么在重要的区域内将有更多的采样。公式（14-4）也显示了蒙特卡罗积分的基本问题：报酬递减。因为估计的方差与$1/N$成比例，所以标准偏差与$1/\sqrt{N}$成比例。由于估计中的误差与标准偏差相类似，所以为了使误差减半需要把N扩大到原来的4倍。

另外一种降低方差的方法是把积分区域S分成为几个较小的区域S_i，并把S_i上积分的总和作为积分的值。这种方法称作分层采样，就是在像素采样中采用的抖动技术（第10章）。通常在每个S_i（密度为p_i）上只取一个采样，在这种情况下估计的方差为

$$\text{var}\left(\sum_{i=1}^{N}\frac{g(x_i)}{p_i(x_i)}\right) = \sum_{i=1}^{N}\text{var}\left(\frac{g(x_i)}{p_i(x_i)}\right) \tag{14-6}$$

可以证明，如果所有的层具有相同的度量，那么分层采样所得的方差绝对不会高于不分层时得到

的方差：

$$\int_{S_i} p(x)\,\mathrm{d}\mu = \frac{1}{N}\int_S p(x)\,\mathrm{d}\mu$$

图形学中最常见的分层采样的例子是对像素采样的抖动，如10.11节所讨论的。

下面的例子使用了蒙特卡罗方法求解积分I，设$g(x)$在区间(0, 4)上等于x：

$$I = \int_0^4 x\,\mathrm{d}x = 8 \tag{14-7}$$

表14-1显示了函数p的形式对N个采样估计的方差的影响。注意当p的形式与g的形式相类似时，方差值会降低。如果$p = g/I$，方差降为0，但是I的值通常是不知道的或者不一定借助蒙特卡罗方法计算。表14-1说明了一个重要的原则，分层采样通常远远优于重要性采样（Mitchell，1996）。虽然I的分层采样的方差与采样数量的立方成反比，但是对于分层采样的方差的特性没有一个普遍的结论。对于某些函数，分层采样的效果并不好。例如白噪音函数，方差对所有的区域都是常数。但是，大多数函数都适合于分层采样，因为各个子集的方差通常小于整个定义域的方差。

289

表 14-1 $\int_0^4 x\mathrm{d}x$ 的蒙特卡罗估计的方差

方　法	采样函数	方　差	标准误差为 0.008 时所需的采样数量
重要性采样	(6 − x) /16	$56.8N^{-1}$	887 500
重要性采样	1/4	$21.3\,N^{-1}$	332 812
重要性采样	(x + 2)/16	$6.3\,N^{-1}$	98 437
重要性采样	x /8	0	1
分层采样	1/4	$21.3\,N^{-3}$	70

准蒙特卡罗积分

一种流行的求积分的方法是在蒙特卡罗积分中用准随机点代替随机点。这些点是确定性的，但是在某种意义上又具有一致性。例如，在单位正方形$[0, 1]^2$上，N个准随机点的集合在正方形中的某区域A上应该具有下面的性质：

$$\text{区域中点的数目} \approx AN$$

例如，一组在点阵中的规则采样就具有这个性质。

在很多积分应用中，准随机点可以提高性能。有时必须小心保证准随机点不会引起走样。特别令人高兴的是，在任何调用了$[0, 1]^d$上的随机点或分层点的应用中，可以不加任何修改地用d维准随机点进行置换。

提出准蒙特卡罗积分直观的动机是，当估计被积分函数的均值时，只要是“合理的”采样点集，都可以采用。

290

14.4 选择随机点

对于分布光线跟踪这样的应用，我们通常希望在单位正方形上生成随机或伪随机的点集。有多种方法可以解决这一问题，例如抖动（见10.11节）。这些方法在单位正方形区域$[0, 1]^2$上生成N个等概率分布点：从(u_1, v_1)到(u_N, v_N)。

有时，采样空间可能不是正方形的（例如一个圆形透镜），或者是不均匀的（例如以像素为中心的滤波函数）。我们希望可以得到一个数学变换，将等分布的点(u_i, v_i)作为输入，并输出预期采样空间中的一个点集（样本空间具有期望的密度）。例如，为了对摄像机镜头采样，上述变

换把点(u_i, v_i)作为输入，并输出点(r_i, ϕ_i)，输出的点近似均匀地分布在镜头的盘片上。当要使用下列变换时会有一个严重的问题：

$$\phi_i = 2\pi u_i$$
$$r_i = v_i R$$

尽管输出的点的确覆盖了镜头，但是它们是不均匀的（图14-6）。在这种情况下需要一个变换，它将等面积区域变换到等面积区域——即将在正方形上的均匀分布变换成新的区域上的均匀分布。

有许多方法可以在非矩形的区域上生成均匀的或者不均匀的点。在下面的章节中，将对三种最常用的方法（求反函数，否决法和Metropolis方法）进行介绍。

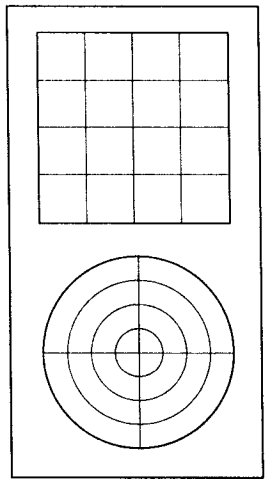


图14-6　把水平和垂直的维度均匀地变换到(r, ϕ)中并没有保持相应的面积，得到的面积并不都是相同的

14.4.1　求反函数

设密度函数$f(x)$是一维的，定义域为$x \in [x_{\min}, x_{\max}]$，可以从一组随机数$\xi_i$中生成具有密度$f$的随机数$\alpha_i$，其中$\xi_i \in [0, 1]$。为此，需要累积概率分布函数$P(x)$：

$$\text{Probability}(\alpha < x) = P(x) = \int_{x_{\min}}^{x} f(x')\mathrm{d}\mu$$

为了得到α_i，简单地对ξ_i进行变换：

291

$$\alpha_i = P^{-1}(\xi_i)$$

其中p^{-1}表示P的反函数。如果P不是解析可逆的，那么用数值法就足够了，因为所有有效的概率分布函数都存在反函数。

注意，由于符号的原因，解析地求函数的反函数比想象得要复杂一些。例如，对函数

$$y = x^2$$

其中$x > 0$，用y是x的函数形式表示其反函数：

$$x = \sqrt{y}$$

解析可逆的函数几乎总是简单的。但是使用标准表示法表示却不是很清晰：

$$f(x) = x^2$$
$$f^{-1}(x) = \sqrt{x}$$

这里x是哑变量。可以看出使用不太标准的表示法更简单一些：

$$y = x^2$$
$$x = \sqrt{y}$$

但要记住这两个函数彼此互为反函数。

例如，在区间[−1, 1]上选择具有下述密度的随机点x_i：

$$p(x) = \frac{3x^2}{2}$$

可以得到

$$P(x) = \frac{x^3 + 1}{2}$$

以及

$$P^{-1}(x)=\sqrt[3]{2x-1}$$

这样，可以从一组标准随机数（ξ_1，…，ξ_N）“转换”为正确分布的数：

$$(x_1,\cdots,x_N)=(\sqrt[3]{2\xi_1-1},\cdots,\sqrt[3]{2\xi_N-1})$$

当然，同样的转换函数可以用来将“均匀的”抖动样本变换成具有期望密度分布的采样。 292

设随机变量$\alpha=(\alpha_x,\alpha_y)$具有定义在$[x_{\min}, x_{\max}]\times[y_{\min}, y_{\max}]$上的二维密度$(x,y)$，那么需要一个二维分布函数：

$$\text{Probability}(\alpha_x<x\text{和}\alpha_y<y)=F(x,y)=\int_{y_{\min}}^{y}\int_{x_{\min}}^{x}f(x',y')\mathrm{d}\mu(x',y')$$

首先利用边缘分布$F(x, y_{\max})$选择x_i，然后根据$F(x_i, y)/ F(x_i, y_{\max})$选择$y_i$。如果$f(x,y)$是可以分解的（分解为$g(x)h(y)$），那么一维上的方法可以用在每一维上。

回到前面的例子，假设从半径为R的圆盘上进行均匀采样，有$p(r,\phi)=1/(\pi R^2)$。二维分布函数为

$$\text{Probability}(r<r_0\text{和}\phi<\phi_0)=F(r_0,\phi_0)=\int_0^{\phi_0}\int_0^{r_0}\frac{r\mathrm{d}r\mathrm{d}\phi}{\pi R^2}=\frac{\phi r^2}{2\pi R^2}$$

这意味着标准数对(ξ_1,ξ_2)可以变换成圆盘上的均匀随机点。

$$\phi=2\pi\xi_1$$
$$r=R\sqrt{\xi_2}$$

这一映射如图14-7所示。

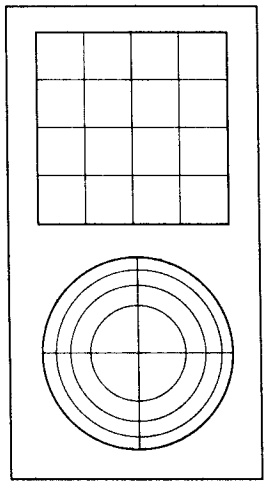


图14-7 从单位正方形的等面积区域到圆盘的等面积区域的映射

为了在真实感绘制应用中选择反射光线的方向，需要根据以下的密度函数在单位半球上选择点：

$$p(\theta,\phi)=\frac{n+1}{2\pi}\cos^n\theta$$

其中n是Phong 指数，θ是与表面法线的夹角，$\theta\in[0, \pi/2]$(在上半球上)，ϕ是方位角($\phi\in[0, 2\pi]$)。累积分布函数为

$$P(\theta,\phi)=\int_0^{\phi}\int_0^{\theta}p(\theta',\phi')\sin\theta'\mathrm{d}\theta'\mathrm{d}\phi' \qquad (14\text{-}8)$$

出现$\sin\theta'$项是因为在球面上$\mathrm{d}\omega=\cos\theta\mathrm{d}\theta\mathrm{d}\phi$。得到边缘密度后，$p$（希望得到的）是可分解的，同时标准随机数对$(\xi_1, \xi_2)$可以使用以下的公式变换为一个方向：

$$\theta=\arccos\left((1-\xi_1)^{\frac{1}{n+1}}\right)$$
$$\phi=2\pi\xi_2$$

293

同样，该方法的好处在于，单位正方形上的一组抖动点可以按照期望分布轻易地变换成半球上的一组抖动点。注意，把n设为1，得到的是在图形系统中常要的漫反射分布。

我们经常要将球面上的点映射到uvw坐标系中合适的方向上。为此，首先将角转变为单位向量$\vec{a}$：

$$\boldsymbol{a}=(\cos\phi\sin\theta,\sin\phi\sin\theta,\cos\theta)$$

为了提高效率，应该避免用反三角函数作为三角函数的参数(如$\cos(\arccos\theta)$)。例如，当$n=1$（漫反射分布）时，向量$\boldsymbol{a}$可以简化为

$$\boldsymbol{a}=\left(\cos(2\pi\xi_1)\sqrt{\xi_2},\sin(2\pi\xi_1)\sqrt{\xi_2},\sqrt{1-\xi_2}\right)$$

14.4.2 拒绝

拒绝（rejection）方法首先按照简单分布选择一些点，然后拒绝那些服从更复杂分布的点。拒绝方法有很多应用场合，下面举例说明。

假设我们需要单位圆内的随机点。首先可以在正方形区域$[-1, 1]^2$内选择随机点(x,y)，然后拒绝圆外的点。设函数$r()$返回一个标准随机数，程序如下：

```
done = false
while (not done) do
  x = −1 + 2r( )
  y = −1 + 2r( )
  if (x² + y² < 1) then
    done = true
```

如果我们需要一个随机数$x\sim p$，已知$p:[a,b]\mapsto\mathbb{R}$且对任意x有$p(x)<m$，那么可以在矩形区域$[a,b]\times[0,m]$中生成随机点，然后取出$y<p(x)$的点：

```
done = false
while (not done) do
  x = a + r( )(b − a)
  y = r( )m
  if (y < p(x)) then
    done = true
```

294

同样的方法可以用于在球面上选取随机点。为了选取一个服从均匀方向分布的单位随机向量，首先在单位球面上选取一个随机点，然后把点作为一个方向向量，方法是在相同方向上选取一个单位向量。

```
done = false
while (not done) do
  x = −1 + 2r( )
  y = −1 + 2r( )
  z = −1 + 2r( )
  if ((l = √(x² + y² + z²)) < 1) then
    done = true
x = x / l
y = y / l
z = z / l
```

虽然拒绝方法通常容易编程实现，但是却很难与分层法兼容。由于这个原因，其收敛的速度更慢，因此主要用于调试或者在特别困难的情况下使用。

14.4.3 Metropolis

Metropolis方法使用随机突变来生成具有期望密度的采样集合。这一原理广泛地用于Metropolis光线传播算法中，我们把它列在本章的文献注记中。假设在定义域S中有随机点x_0，再假设对任意的点x，都存在一种生成随机$y\sim p_x$的方式。使用边缘表示法$p_x(y)\equiv p(x\to y)$表示这一密度函数。现在，假设x_1是S中按照基本密度$p(x_0\to x_1)$选取的随机点。然后按照密度$p(x_1\to x_2)$生成x_2等。极限情况下，生成采样的数量为无穷大，可以证明采样将具有某一基本密度，密度由p决定而与初始点x_0无关。

现在，假设我们希望选取p，以使所有采样符合的基本密度与某个函数$f(x)$成比例，这里函数f是定义域为S的非负函数。进一步假设可以求函数f的值，但是我们却只有少量的或者根本没有关于函数性质的附加信息（在图形学中这样的函数十分普遍）。再假设我们有能力利用基本密度函数$t(x_i \to x_{i+1})$将x_i变换为x_{i+1}。为了增加灵活性，进一步假设我们添加x_i转变成它自身的概率（即$x_{i+1} = x_i$），该概率可能不为零。我们把它称为生成潜在的候选点$y \sim t(x_i \to y)$，并且以概率$a(x_i \to y)$“接受”这个候选点（即$x_{i+1} = y$），以概率$1 - a(x_i \to y)$拒绝这个候选点（即$x_{i+1} = x_i$）。注意，虽然序列$x_0, x_1, x_2, \cdots$是一个随机集合，但是在采样之间将存在某些关联。它们将适合于蒙特卡罗积分或密度估计，但是分析这些估计的方差是很具有挑战性的。 295

现在，假设给定转移函数$t(x \to y)$和模拟分布函数$f(x)$，是否可以用$a(y \to x)$使所有点按照f的形状分布呢？或者更精确地说，

$$\{x_0, x_1, x_2, \cdots\} \sim \frac{f}{\int_s f}$$

通过确保x_i在某种意义上是不变的，可以证明上述假设是成立的。如果想象有一个巨大的采样点x的集合，我们希望两点之间的“流动”在每个方向上都是一致的。如果假设x和y附近的点的密度分别与$f(x)$和$f(y)$成比例，那么在两个方向上的流动应该是一致的：

$$\text{flow}(x \to y) = kf(x)t(x \to y)a(x \to y)$$
$$\text{flow}(y \to x) = kf(y)t(y \to x)a(y \to x)$$

其中k是某个正常数。将两个流动设为常数，就对a加了一个约束：

$$\frac{a(y \to x)}{a(x \to y)} = \frac{f(x)t(x \to y)}{f(y)t(y \to x)}$$

因此，如果已知$a(y \to x)$和$a(x \to y)$两者中的一个，就能求出另外一个。增大它们的值就可以提高接受的可能性，因此常用的方法是将两者中较大的设为1。

使用Metropolis采样生成技术的难点是，在点集达到“好”之前很难估计还需要多少个点。如果丢弃前n个点速度就会提高，所以明智地选择n是非常重要的。

14.4.4 示例：在正方形上选择随机直线

作为设计采样策略的完整过程的例子，考虑这个问题：找出与单位正方形$[0, 1]^2$相交的随机直线。我们希望这是一个合理的过程；即希望直线在正方形内均匀地分布。我们可以直觉地感到问题存在一些微妙之处；在倾斜方向的直线比垂直或水平方向的“多”。这是因为正方形的横截面是不均匀的。 296

如果存在反函数，那么首要目标是找到求函数方法，如果失败的话，就要求助于拒绝方法或者Metropolis方法。这是因为我们希望在直线空间中得到分层样本。首先尝试使用正则坐标，因为在正方形内选择随机直线的问题就是在(r, θ)空间中寻找均匀随机点的问题，(r, θ)是与正方形中直线对应的空间。

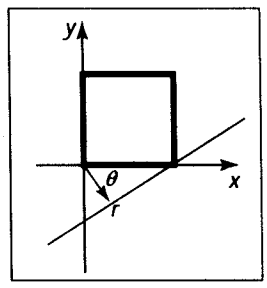


图14-8 对于$\theta \in [-\pi/2, 0]$，最大距离r对应与正方形相交的一条直线。因为正方形的边长为1，所以$r = \cos\theta$

考虑区域$-\pi/2 < \theta < 0$。与正方形中的直线对应的r等于多少？对于上述区域，$r < \cos\theta$就是与正方形相交的所有直线，如图14-8所示。在另外四个象限中应用类似的推理可以求得在(r, θ)空间中进行采样的区域，如图14-9所示。区域的边界方程$r_{\max}(\theta)$为

$$T_{\max}(\theta)=\begin{cases}0 & \text{如果}\theta\in\left[-\pi,-\dfrac{\pi}{2}\right]\\ \cos\theta & \text{如果}\theta\in\left[-\dfrac{\pi}{2},0\right]\\ \cos\theta+\sin\theta & \text{如果}\theta\in\left[0,\dfrac{\pi}{2}\right]\\ \sin\theta & \text{如果}\theta\in\left[\dfrac{\pi}{2},\pi\right]\end{cases}$$

因为$r_{\max}(\theta)$下面的区域是简单函数，下限是$r=0$，所以可以首先根据下列密度函数选取θ对区域采样：

$$p(\theta)=\frac{r_{\max}(\theta)}{\int_{-\pi}^{\pi}r_{\max}(\theta)\mathrm{d}\theta}$$

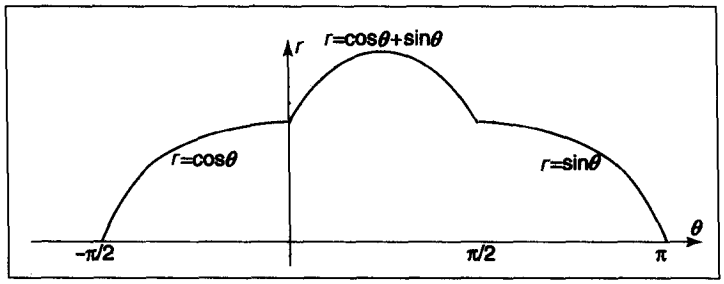


图14-9　与单位正方形$[0,1]^2$相交的直线的最大的半径，作为θ的函数

上式的分母等于4。现在，可以计算累积概率分布函数：

$$P(\theta)=\begin{cases}0 & \text{如果}\theta\in\left[-\pi,-\dfrac{\pi}{2}\right]\\ (1+\sin\theta)/4 & \text{如果}\theta\in\left[-\dfrac{\pi}{2},0\right]\\ (2+\sin\theta-\cos\theta)/4 & \text{如果}\theta\in\left[0,\dfrac{\pi}{2}\right]\\ (3-\cos\theta)/4 & \text{如果}\theta\in\left[\dfrac{\pi}{2},\pi\right]\end{cases}$$

297

把$\xi_1=P(\theta)$转换为$\theta=g(\xi_1)$的形式，可以得到上述公式的反函数：

$$\theta=\begin{cases}\arcsin(4\xi_1-1) & \text{如果}\xi_1<\dfrac{1}{4}\\ (\arcsin((4\xi_1-2)^2))/2 & \text{如果}\xi_1\in\left[\dfrac{1}{4},\dfrac{3}{4}\right]\\ \arccos(3-4\xi_1) & \text{如果}\xi_1>\dfrac{3}{4}\end{cases}$$

一旦得到θ，则r为

$$r=\xi_2 r_{\max}(\theta)$$

正如前面讨论的那样，有许多用参数描述直线的形式，每种都有相关的“合理”度量。我们也可以在任意一种空间中产生随机直线。例如，在斜率-截距空间中，与正方形相交的区域如图14-10所示。通过类似对正则空间的推理，斜率的密度函数为

$$p(m)=\frac{1+|m|}{4}$$

相应的微分度量为

$$\mathrm{d}\mu=\frac{\mathrm{d}m}{(1+m^2)^{\frac{3}{2}}}$$

298

由此可以得到累积分布函数：

$$P(m)=\begin{cases}\dfrac{1}{2}-\dfrac{m-1}{2\sqrt{1+m^2}} & \text{如果}m<0\\ 1+\dfrac{m-1}{2\sqrt{1+m^2}} & \text{如果}m\geqslant 0\end{cases}$$

通过求解二次方程可以得到其反函数。假设m是利用ξ_1生成的，得

$$b=\begin{cases}-m+2(m+1)\xi_2 & \text{如果}\xi<\dfrac{1}{2}\\ (1-m)(2\xi_2-1) & \text{其他}\end{cases}$$

这种方式并不比使用正则坐标的方式好，它仅仅是一种可供选择的方式。

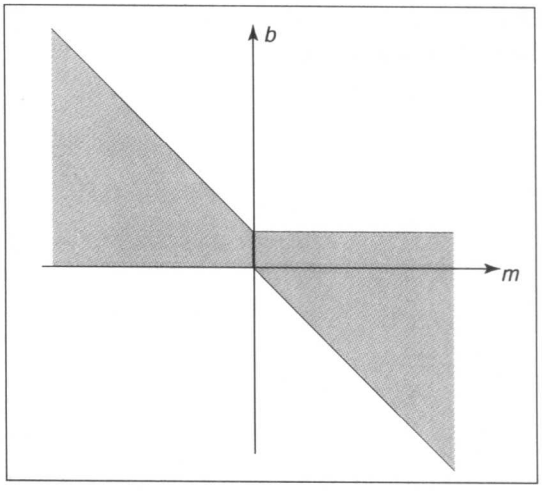


图14-10 (m,b)空间中的区域，它包含与单位正方形$[0,1]^2$相交的直线

常见问题

- **本章讨论的是概率而不是统计学。它们的主要区别是什么？**

概率研究的是事件的可能性。统计学要推导出巨大的但有限数量的随机变量特征。从这种意义上说，统计可以看作应用概率的一个特例。

- **Metropolis采样是否等同于Metropolis光线传播算法？**

不是。Metropolis光线传播（Veach & Guibas, 1997）算法将Metropolis采样作为其过程的一部

分，但是它是为绘制特别设计的，因此还包括其他的步骤。

文献注记

Geometric Probability（Solomon，1978）是关于几何概率的经典参考书目。*Random-Edge Discrepancy of Supersampling Patterns*（Dobkin & Mitchell，1993）给出了另一种在正方形上选择随机边的方法。Efficient Multidimensional Sampling（Kollig & Keller, 2002）中记载了更多关于图形学中的准蒙特卡罗方法的信息。有三本关于蒙特卡罗方法的经典著作：*Monte Carlo Methods*（Hammersley & Handscomb, 1964）、*Monte Carlo Methods, Basics*（Kalos & Whitlock,1986）和*The*
299 *Monte Carlo Method*（Sobel, Stone, & Messer,1975）。

习题

1. 单位立方体 $(x, y, z) \in [0, 1]^3$ 上的函数 xyz 的平均值是多少？
2. 单位圆盘 $(r, \phi) \in [0, 1] \times [0, 2\pi)$ 上 r 的平均值是多少？
3. 证明将标准随机点 (ξ_1, ξ_2) 均匀映射到任意三角形的重心坐标系中的公式是：$\beta = 1 - \sqrt{1-\xi_1}$，$\gamma = (1-u)\xi_2$。
4. 单位正方形中的直线的平均长度是多少？在单位正方形内生成1000万条直线并计算它们长度的平均值来验证你的答案。
5. 单位立方体内的直线的平均长度是多少？在单位立方体内生成1000万条直线并计算它们长度的平均值来验证你的答案。

300 6. 利用方差的定义证明 $V(x) = E(x^2) - [E(x)]^2$。

第15章

曲　线

Michael Gleicher

15.1 曲线

一提到曲线（curve），我们直觉地认为是可以用钢笔画出来的。曲线是钢笔在一段时间内经过的一系列点的集合。尽管我们通常认为钢笔是在纸上画东西（比如说，在二维空间是一条曲线），笔也可以在三维空间绘制出空间曲线（space curve），或者可以想象钢笔在其他空间中移动。

在数学上，至少可以有两种方式定义曲线：

(1) 在n维空间中某区间上的连续图像。

(2) 从一维空间到n维空间的连续映射。

这两种定义都来源于时间间隔范围（钢笔画出曲线的时间）的思想。然而有一个明显的区别：在第一个定义中，曲线是钢笔画过的一系列点的集合（图像），而在第二个定义中，曲线是时间和点集之间的映射。在本章中我们使用第一个定义。

一条曲线是一个无限大的点集。曲线上的点具有以下性质：除了极个别的点（它们是端点）只有一个相邻点，其余的都有两个相邻点。一些曲线没有端点，可能是因为他们是无限的（就像直线一样），或者是因为它们是封闭的（循环一圈并与自身相连）。

因为生成曲线的“钢笔”非常细（无穷小的），所以很难生成一个填充区域。尽管可能会有空间填充的曲线（通过对它们进行无限次的折叠），在这里，我们不考虑这种复杂的数学特殊情况。通常曲线作为事物的外部轮廓，而不是“内部的”。

我们要解决的问题是如何定义一条曲线，即给出曲线的名字或表达式，这样就可以在计算机上表示曲线。对于某些曲线来说，给曲线命名是非常容易的问题，因为它们的形状都是已知的：如线段、圆弧、椭圆弧等。一般的曲线，如果没有“经过命名”的形状，有时称为*自由形式*（free form）曲线。因为自由形式的曲线可以是任意的形状，所以它们都很难定义。

用数学方法描述曲线主要有三种方式：

- **隐式曲线**　表示法通过一个过程定义曲线上的点集，这个过程可以用来判断一个点是否在曲线上。通常，隐式曲线表示法通过*隐式函数*来定义，其形式如下：

$$f(x,y)=0$$

  所以曲线是满足上面等式的点的集合。注意隐式函数f是个标量函数（它返回一个实数）。

- **参数曲线**　表示法定义了从*自由参数*到曲线上点集的映射。即自由参数为曲线上的点提供了一个索引。曲线的参数形式是一个函数，它给自由参数的每一个值指派了一个位置。直觉上，如果认为曲线是用钢笔在纸上画出来的，那么自由参数就是时间，取值范围就是从开始画曲线到结束的时间间隔。曲线的*参数函数*指出在任意时刻钢笔的位置：

$$(x,y)=\boldsymbol{f}(t)$$

注意，参数方程是一个向量值函数。这个例子是一个二维曲线，所以函数的输出值是二维向量，在三维情况下输出结果为三维向量。

- **生成式或过程式曲线**　表示法提供了一个过程，可以生成曲线上的点，得到是不同于上述两类的曲线。生成式曲线描述例子包括子分模式和分形。

302 别忘了曲线是一个点集。这些表示法给出了定义点集的方法。任何曲线可能都有很多表示形式。为此，数学家们通常都很仔细地区分曲线和它的表示形式之间的区别。在计算机图形学中我们经常混淆，因为我们通常指的是表示形式，而不是实际的曲线本身。所以当提到“隐式曲线”，这可能是指用隐式函数表示的曲线，也可能指表示某条曲线的隐式函数。所以它们之间的差别往往不是很重要的，除非要考虑同一曲线的不同表达形式。在本章中将要考虑不同的曲面表示形式，所以我们要仔细一些。当使用术语如“多项式曲线”时，我们指的是能够用多项式表示的曲线。

按照本章开始给出的定义，如果要成为一条曲线，它必须有一种参数表示形式。然而，很多曲线还有其他的表示形式。例如，在二维空间中，以原点为圆心半径等于1的圆可以写成下面的隐式形式：

$$f(x,y)=x^2+y^2-1=0$$

或写成参数形式：

$$(x,y)=\boldsymbol{f}(t)=(\cos t,\sin t),\ t\in[0,2\pi\ ).$$

对于给定的曲线，参数形式不一定是最方便的表示形式。事实上，有些曲线可能有简单的隐式或生成式表示形式，但却很难找到它们的参数表示形式。

不同的曲线表示形式，具有不同的优缺点。比如，参数曲线更容易画出来，因为可以在自由参数中采样。通常，在计算机图形学中参数形式是最常用的，因为它们更容易处理。我们将把重点放在曲线的参数表示形式上。

15.1.1　参数化和重新参数化

参数曲线指的是由特别的区间上的具体参数函数定义的曲线。更准确地说，一条参数曲线具有一个给定的函数，这个函数是从参数区间的一个映射。通常令参数位于从0到1的单元区间是很方便的。当自由参数在单元区间内变化时，我们通常用u来表示参数。

如果把参数曲线作为用钢笔画的一条线，可以认为$u=0$是钢笔在纸上开始落笔的时刻，单
303 位时间是钢笔画曲线所用的总时间（曲线的结束时参数$u=1$）。曲线可以用一个函数定义，该函数把时间（在单位坐标系内）映射到位置。本质上，曲线的定义是一个函数，它能回答“在时间u笔在哪儿”的问题。

如果函数$\boldsymbol{f}(t)$在区间$[a,\ b]$上定义了一条曲线，我们可以很容易地定义一个新函数$\boldsymbol{f}_2(u)$，使它在单位区间上定义同一条曲线。首先定义：

$$g(u)=a+(b-a)u,$$

然后定义：

$$\boldsymbol{f}_2(u)=\boldsymbol{f}(g(u)).$$

两个函数$\boldsymbol{f}$和$\boldsymbol{f}_2$都代表同一条曲线，然而它们提供了曲线不同的参数化（parameterization）形式。为一条已知曲线生成新的参数化形式的过程称为重新参数化（re-parameterization），从旧参数到新参数的映射（本例中就是g）称为重新参数化函数（re-parameterization function）。

如果通过某种参数化方法定义了一条曲线，它还存在无数其他形式（因为总是可以通过重新参数化来得到新的形式）。曲线有多个参数化形式是很有用的，因为它允许创建方便的参数化形

式。但是这其中也有问题，因为很难比较两个函数来判断它们是否表示同一条曲线。

问题的实质可能更严重：自由参数（或者是时间元素）的存在给曲线的表示形式增加了潜在的不可见的未知要素。当曲线画完后我们再去观察它，我们不需要知道时间的安排。在整个时间间隔内钢笔可能一直是以恒定的速度运动，或者是缓慢地开始，然后再加速。例如，当$u=0.5$时，也就是在参数空间一半的时候，如果钢笔是缓慢开始然后加速的话，它就不会在沿曲线一半的地方。考虑下面的一条非常简单的曲线的表示形式：

$$\begin{aligned}(x,y) &= \boldsymbol{f}(u) = (u,u)\\ (x,y) &= \boldsymbol{f}(u) = (u^2,u^2)\\ (x,y) &= \boldsymbol{f}(u) = (u^5,u^5)\end{aligned}$$

所有的三个函数表示相同单元区间上的同一条曲线，然而，当u不是0或者1的时候，根据不同曲线表示形式$\boldsymbol{f}(u)$代表不同的点。

如果给定曲线的一种参数化形式，可以直接用它来定义曲线，也可以建立一个更方便的参数化形式。通常，自然参数化（natural parameterization）是为了便于定义曲线而创建的形式，这样 304
我们没有必要知道曲线上速度的变化。

如果钢笔以恒定的速度运动，那么自由参数的值就有更多的含义。参数空间的中点就是曲线上的中点，不要考虑时间，参数就是沿着曲线长度的度量。这样的参数化形式称为弧长参数化（arc-length parameterization），因为它们通过函数定义曲线，该函数把沿着曲线的长度（就是弧长）映射成曲线上的位置。通常用参数s来表示弧长参数。

从技术角度上来说，如果曲线的切线斜率（即参数化形式相对于其参数的导数）大小恒定的话，参数化表示就是一个弧度参数化表示。可以用公式表示：

$$\left|\frac{\mathrm{d}\boldsymbol{f}(s)}{\mathrm{d}s}\right|^2 = c$$

计算沿曲线的长度是非常麻烦的。一般来说，它是通过对导数的积分（直观上讲，导数的大小是笔在曲线上移动的速度）定义的。所以，给定参数v的值，可以用下列公式计算参数s（曲线上从点$\boldsymbol{f}(0)$到点$\boldsymbol{f}(v)$的弧长距离）：

$$s = \int_0^v \left|\frac{\mathrm{d}\boldsymbol{f}(t)}{\mathrm{d}t}\right|^2 \mathrm{d}t \qquad (15\text{-}1)$$

其中$\boldsymbol{f}(t)$是用自然参数化表示定义曲线的函数。

要利用弧长参数化表示需要求解式（15-1），给定s求解t。对于要考察的多种曲线，必须用数值分析的方法求解，不可能用闭形式（简单）的方式求解。

通常用参数u来表示在单位区间中变化的自由参数，s表示弧长自由参数，t表示其他的参数。

15.1.2 分段的参数表达式

对某些曲线，定义一个函数来表示其形状是很容易的事情。例如，直线、圆和椭圆都有简单的函数，用参数来表示点。对许多其他的曲线，找一个函数来定义它们的形状是很困难的。定义复杂曲线的主要策略是分治法：把曲线分成许多简单的小段，每一段都有简单的描述形式。 305

例如，考虑图15-1中的曲线。前两条曲线可以很容易地用两段来定义。对于图15-1b中的曲线，需要不同类型的片断：一条直线和一个圆弧。

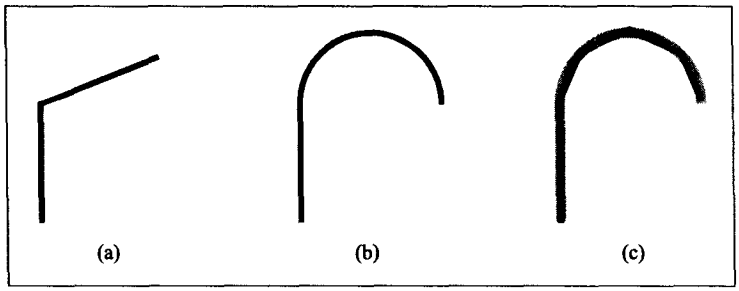


图15-1　(a)可以简单地用两条直线表示的曲线；(b)可以简单地用一条直线和一个圆弧表示的曲线；(c)用5条直线段逼近曲线(b)得到的曲线

为了创建复合曲线的参数化表示（如图15-1b中的曲线），需要用参数函数表示不同的曲线段。如果我们定义了参数函数的范围在$0 \leqslant u \leqslant 1$内，在图15-1a或图15-1b中可以定义为如下方程：

$$f(u)=\begin{cases} f_1(2u) & \text{如果 } u \leqslant 0.5 \\ f_2(2u-1) & \text{如果 } u > 0.5 \end{cases} \tag{15-2}$$

其中f_1表示第一段的参数化形式，f_2表示第二段的参数化形式，两个函数都定义在单位区间上。

定义f_1和f_2时，要保证曲线的分段相吻合。如果$f_1(1) \neq f_2(0)$，那么曲线的分片就不能连接在一起形成一个连续的曲线。

为了表示图15-1b中的曲线，需要两种分段函数：直线和圆弧。为简单起见，我们更希望用一种片段。但是如果只用一种片段（线段）表示图15-1b的曲线，就不能够准确地生成曲线（除非用无穷个片段来表示）。尽管用线段表示的曲线（如图15-1c所示）的形状和图15-1b的曲线表示不太一致，但是我们用的话也足够了。在这种情况下，为简单起见我们宁可选择用简单线段表示，
306 而不用那些能更准确表示曲线形状的方法。

同时，注意到随着片段数的增加，近似的结果就越好。在极限情况下（用无穷多的片段），我们就能够准确地表达原始的形状。

利用分段表达的优点是，它允许在下列选项中找到一个折中：

(1) 当前表示的曲线与真实的曲线形状的近似程度；

(2) 使用的片段的复杂度；

(3) 使用的片段的数量。

所以，如果要表示复杂形状，我们需要确定能够接受的近似程度，并使用少量的简单片段。为了提高近似程度，我们可以选择使用更多的片段或者是使用更复杂的片段。

在计算机图形学的实际使用中，我们倾向于使用相对比较简单的曲线片段（线段、弧或者多项式片段）。

15.1.3　样条

在电脑使用之前，当绘图员要画一条平滑的曲线时，他们使用的工具是一片坚硬的可以被弯成任意形状的金属片。因为金属可以被弯曲，不是折叠，它就可以形成比较光滑的形状，坚硬意味着它可以尽量弯成我们想要的形状。这个坚硬的金属片称为曲线尺（spline）。

数学家们发现可以用分段多项式函数表示用绘图员的曲线尺创建的曲线。最初，他们用术语spline（后面我们把它称为“样条”）指平滑的分段多项式函数。最近，术语样条用来表示任意的分段多项式函数，我们采用的是后面的定义。

对我们而言，样条是分段多项式函数，这样的函数对于表示曲线是非常有用的。

15.2 曲线的性质

为了描述曲线，我们需要给出一些它的性质。对于已经命名的曲线，根据曲线的类型它往往有具体的性质。例如，为了描述圆，要给出圆的半径和它的圆心位置。对于椭圆，还要提供它的主轴的方向和轴长的比例。然而对于自由形状的曲线，需要更一般的性质集合来描述特定的曲线。 307

曲线的一些性质是只归结于曲线的一个局部，而其他的性质则是需要关于整个曲线的信息。为了从直觉上明白它们的不同，我们可以把曲线想象成火车道，在一个雾蒙蒙的天气里，站在火车道上可以说出轨迹是直的或者是弯曲的，是否在终点。这些就是*局部*（local）性质。但是不能回答轨道是否是封闭的，或者它本身是否交叉，或者它的长度。我们把这种类型的性质称为*全局*（global）性质。

关于几何体（曲线和表面）的局部性质的研究称为*微分几何*（differential geometry）。从技术上讲，对于微分的性质，有许多关于性质的数学限制（大概地说，在火车轨迹的例子中，就是我们不能具有GPS或者指南针）。不用担心这种区别，我们将使用*局部*性质而不是微分性质。

局部性质对描述曲线是非常重要的工具，因为他们不需要关于整个曲线的信息，局部性质包括：

- 连续性。
- 曲线上具体地方的位置。
- 曲线上具体地方的方向。
- 曲率（和其他导数）。

我们经常指定曲线要包含一个特殊的点。如果该点是曲线的一部分，就称为曲线*插值*（interpolate）了一个点。函数f插值了一个值v，如果参数t存在一个值使$f(t) = v$。我们把插值的地方（即参数t的值）称为*插值位置*。

连续性

理解曲线的局部性质是非常重要的，尤其是在两个参数片段连接的地方。如果曲线用如式（15-2）一样的公式定义，那么必须注意片段是如何定义的。如果$f_1(1) \neq f_2(0)$，那么曲线就会“断开”——我们也不可能用钢笔连续地画出曲线。这个条件称为是曲线段连接的*连续性*（continuity）条件，因为如果它们满足条件，曲线就能作为一条连续的片段画出。在本章的刚开始时“曲线”的定义要求曲线是连续的，从技术上讲“断开的曲线”不是一条曲线。 308

除了位置，我们还可以检查片段的导数是否正确匹配。如果$f'_1(1) \neq f'_2(0)$，合并后的曲面在连接点处的一阶导数将有一个突变，一阶导数是不连续的。通常来说，如果在片段间所有的导数直到n阶导数都能匹配，那么曲面是C^n连续的。我们把位置本身称为0阶导数，所以C^0连续性条件表示在曲线的位置是连续的。C^1连续性表示在曲线的位置及其一阶导数都是连续的，曲线的定义要求曲线满足C^0。

图15-2是一些连续性条件的示意图。一阶导数不连续（曲线满足C^0，而不满足C^1）通常是非常明显的，因为它显示出一个明显的尖角。二阶导数不连续有时候也是比较明显的。更高阶的不连续性可能是一个要紧的问题，这主要取决于实际的应用。例如，如果曲线代表一定的运动，二阶导数上突变是非常明显的，所以经常会用到三阶导数。如果有液体从曲线上流过去（例如，如果它的形状是飞机的机翼或者是船体），在四阶或五阶导数的不连续可能会引起湍流。

因为参数化表示中的速度可能是不同的，即使导数是匹配的，所以我们定义了另外一种连续性，它忽略了速度。我们定义*几何连续性*（geometry continuity）条件是在一个片段的终点和下一

段的始点的一阶导数只是在量度上不同，也就是C^1条件要求：

309

$$f_1'(1) = f_2'(0)$$

G^1连续性条件要求，对于某些值k

$$f_1'(1) = k f_2'(0)$$

参数连续性比几何连续性更严格。（G^n是满足C^n的必要条件,但反过来不对）。

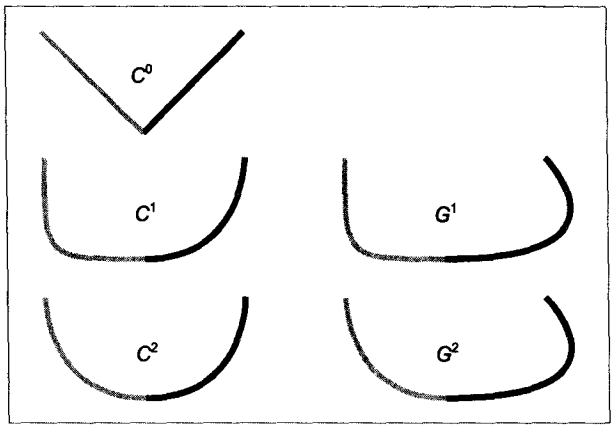


图15-2　两个曲线段之间不同类型的连续性示意图

15.3　多项式片段

在计算机图形学中广泛使用的曲线表示方法是把基本图元连接在一起，这些图元是用多项式定义的，称为多项式片段，例如，直线图元通过线性多项式定义，在15.3.1节中，我们给出一个正式的定义，并且讲解如何把多项式片段连接在一起。

15.3.1　多项式的表示

多项式是具有以下形式的函数：

$$f(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \tag{15-3}$$

其中，a_i称为系数，如果$a_n \neq 0$,n称为多项式的次数，还可以把式（15-3）写成下列形式：

$$f(t) = \sum_{i=0}^{n} a_i t^i \tag{15-4}$$

上面的公式称为多项式的规范形式。

我们可以对规范形式进行泛化，得

$$f(t) = \sum_{i=0}^{n} c_i b_i(t) \tag{15-5}$$

其中$b_i(t)$是多项式。针对不同的应用，可以选择便于处理的多项式。这些多项式称为基函数（basis function）或是混合函数（blending function）（见15.3.5节）。在式（15-4）中，t^i就是式（15-5）中t^i的$b_i(t)$。如果能正确地选择基函数集，可以选择适当的c表示任意的n+1次多项式。

310　规范形式并不总是有方便的系数。本章中，出于实用的目的，我们将找到几组基函数，它们能使系数方便地控制多项式表示的曲线。

为了确定二维空间中的一条曲线，可以确定两个t的多项式：一个是x随着t变化的多项式，另一个是y随着t变化的多项式；或者定义一个多项式，其中每一个$\boldsymbol{a}_i$都代表一个二维点。类似的情形对于n维空间中的任意曲线也存在。

15.3.2 线段

为了介绍分段多项式曲线的概念，首先讨论线段。在实际中，线段非常简单，用数学方法求导好像是多余的。然而，通过理解这个简单的情况，就可以很容易地把这种方法使用在其他更复杂的多项式上。

假设线段从点$\boldsymbol{p}_0$连接到点$\boldsymbol{p}_1$，线段单位区间上的参数方程如下：

$$\boldsymbol{f}(u)=(1-u)\boldsymbol{p}_0+u\boldsymbol{p}_1 \tag{15-6}$$

曲线写成向量形式，可以隐藏点的维数以及要分开处理每一个维的事实。例如在二维中，要创建两个分开的公式：

$$f_x(u)=(1-u)x_0+ux_1$$
$$f_y(u)=(1-u)y_0+uy_1$$

该线段是由两个端点来确定的，但是从现在开始将一直使用向量符号，因为它更简洁。我们将把控制参数向量$\boldsymbol{p}$称为控制点集，$\boldsymbol{p}$中的每一个元素称为一个控制点。

尽管通过端点的位置来描述线段很明显，通常也很方便，但是还有很多描述线段的其他方式，例如：

(1) 线段中点的位置、方向和长度。

(2) 一个端点的位置和第二个点相对于第一个点的位置。

(3) 线段中点的位置和其中一个端点。

显然，给定线段的一种描述方法，我们可以转换成其他的描述方法。 311

另一种描述线段的方法是利用多项式的规范形式（如15.3.1节中讨论的）：

$$\boldsymbol{f}(u)=\boldsymbol{a}_0+u\boldsymbol{a}_1 \tag{15-7}$$

任何线段都可以通过指定$\boldsymbol{a}_0$和$\boldsymbol{a}_1$或者两个端点（$\boldsymbol{p}_0$和$\boldsymbol{p}_1$）来表示。确定端点更方便，因为可以从端点计算出其他的参数。

为了把规范形式写成向量表达式，定义向量$\boldsymbol{u}$，它是表示u的一系列幂数的向量：

$$\boldsymbol{u}=\left[1\ u\ u^2 u^3\cdots u^n\right]$$

这样式（15-4）就可以写成如下的形式：

$$\boldsymbol{f}(u)=\boldsymbol{u}\cdot\boldsymbol{a} \tag{15-8}$$

这个向量表示法将使曲线不同形式之间的转换更容易。

式（15-8）描述了用一组多项式系数定义的曲线段，系数是指多项式的简单形式的系数。这种表达式称为规范形式，其参数记为$\boldsymbol{a}$。

虽然规范形式在数学上很简单，但是它并不一定是定义曲线最方便的方式。例如，我们可能喜欢通过端点的位置来定义线段。如果定义$\boldsymbol{p}_0$是线段的始点（线段中参数u = 0的位置），$\boldsymbol{p}_1$是线段的终点（线段中参数u = 1的位置）。可以写成下面的式子：

$$\begin{aligned}\boldsymbol{p}_0&=\boldsymbol{f}(0)=\left[1\ 0\right]\cdot\left[\boldsymbol{a}_0\ \boldsymbol{a}_1\right]\\ \boldsymbol{p}_1&=\boldsymbol{f}(1)=\left[1\ 1\right]\cdot\left[\boldsymbol{a}_0\ \boldsymbol{a}_1\right]\end{aligned} \tag{15-9}$$

求解两个公式，得到$\boldsymbol{a}_0$和$\boldsymbol{a}_1$：

$$\boldsymbol{a}_0 = \boldsymbol{p}_0$$
$$\boldsymbol{a}_1 = \boldsymbol{p}_1 - \boldsymbol{p}_0$$

多项式的矩阵表达形式

尽管第一个例子比较容易，能够进行求解，但是对于更复杂的例子，把式（15-9）写成下面的形式更简单：

312

$$\begin{bmatrix}\boldsymbol{p}_0\\ \boldsymbol{p}_1\end{bmatrix} = \begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}\begin{bmatrix}\boldsymbol{a}_0\\ \boldsymbol{a}_1\end{bmatrix}$$

还可以写成下面的形式：

$$\boldsymbol{p} = \boldsymbol{C}\boldsymbol{a} \tag{15-10}$$

其中$\boldsymbol{C}$称为*约束矩阵*（constraint matrix）[1]。如果对点的向量形式感到困扰，可以对每一个维度单独考虑（所以点$\boldsymbol{p}$是$[x_0\ x_1]$或者是$[y_0\ y_1]$，$\boldsymbol{a}$也进行相应的处理）。

通过寻找$\boldsymbol{C}$的逆矩阵，可以从式（15-10）解出$\boldsymbol{a}$。这个逆矩阵称为基矩阵，用$\boldsymbol{B}$表示。基矩阵很方便，因为它告诉我们如何在方便的参数$\boldsymbol{p}$和规范形式$\boldsymbol{a}$之间进行转换，所以给了我们一种计算曲线值的简单方式

$$f(u) = \boldsymbol{u}\boldsymbol{B}\boldsymbol{p}$$

对于任何形式的曲线，都可以找到一个基矩阵，假设在参数的定义中没有非线性关系。非线性定义的参数包括线段的长度和角度。

现在假设要对线段进行参数化，也就是让$\boldsymbol{p}_0$代表中间点（即$u = 0.5$），$\boldsymbol{p}_1$代表终点（即$u = 1$）。为了得到这个参数方程的基矩阵，设

$$\boldsymbol{p}_0 = f(0.5) = 1\,\boldsymbol{a}_0 + 0.5\,\boldsymbol{a}_1$$
$$\boldsymbol{p}_1 = f(1) = 1\,\boldsymbol{a}_0 + 1\,\boldsymbol{a}_1$$

所以

$$\boldsymbol{C} = \begin{bmatrix}1 & 0.5\\ 1 & 1\end{bmatrix}$$

因而

$$\boldsymbol{B} = \boldsymbol{C}^{-1} = \begin{bmatrix}2 & -1\\ -2 & 2\end{bmatrix}$$

15.3.3 超线段

线段是非常简单的，以至于寻找其基矩阵是很直接的。然而，它对于高次的曲线是个好的练习。首先考虑二次方程（曲线是二次的）。规范式（式（15-4））的优点就是它适用于更复杂的曲
313 线，令n取更大的数值就是了。

一个二次方程（二次多项式）有三个系数，$\boldsymbol{a}_0$、$\boldsymbol{a}_1$和$\boldsymbol{a}_2$。这些系数对于描述曲线的形状是不很方便的。但是可以使用基矩阵的方法来推导出更方便的参数。如果已知u的值，式（15-4）就变成了参数的线性方程，那么就可以使用上一节中线性代数的机制了。

假定要用曲线的始点（$u = 0$），中点[2]（$u = 0.5$）和终点（$u = 1$）来描述曲线。在式（15-4）中代入合适的值，得

1. 我们假设向量（行或者列）的形式在上下文中是比较明显的，我们将忽略所有向量的转置符号。
2. 注意中点是指在参数空间的中间，它不一定在曲线的中间。

$$\begin{aligned}
\boldsymbol{p}_0 = \boldsymbol{f}(0) &= \boldsymbol{a}_0 + 0^1 \quad \boldsymbol{a}_1 + 0^2 \quad \boldsymbol{a}_2 \\
\boldsymbol{p}_1 = \boldsymbol{f}(0.5) &= \boldsymbol{a}_0 + 0.5^1 \quad \boldsymbol{a}_1 + 0.5^2 \quad \boldsymbol{a}_2 \\
\boldsymbol{p}_2 = \boldsymbol{f}(1) &= \boldsymbol{a}_0 + 1^1 \quad \boldsymbol{a}_1 + 1^2 \quad \boldsymbol{a}_2
\end{aligned}$$

所以约束矩阵为

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0.5 & 0.25 \\ 1 & 1 & 1 \end{bmatrix}$$

基矩阵为

$$\boldsymbol{B} = \boldsymbol{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 4 & -1 \\ 2 & -4 & 2 \end{bmatrix}$$

有一个附加的约束（或参数），有些时候很容易指定它：曲线的导数（相对于它的自由参数）必须是特定的值。直觉上，导数值反映了曲线的变化情况，所以一阶导数表示了曲线的走向，二阶导数表示曲线方向变化的快慢，等等。后面将会看到一些例子，说明为什么指定导数是非常有用的。

对二次方程

$$\boldsymbol{f}(u) = \boldsymbol{a}_0 + \boldsymbol{a}_1 u + \boldsymbol{a}_2 u^2$$

其一阶导数很简单：

$$\boldsymbol{f}'(u) = \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}u} = \boldsymbol{a}_1 + 2\boldsymbol{a}_2 u$$

二阶导数为

$$\boldsymbol{f}''(u) = \frac{\mathrm{d}^2\boldsymbol{f}}{\mathrm{d}u^2} = \frac{\mathrm{d}\boldsymbol{f}'}{\mathrm{d}u} = 2\boldsymbol{a}_2$$

314

或者更一般地表达为：

$$\begin{aligned}
\boldsymbol{f}'(u) &= \sum_{i=1}^{n} i u^{i-1} \boldsymbol{a}_i \\
\boldsymbol{f}''(u) &= \sum_{i=2}^{n} i(i-1) u^{i-2} \boldsymbol{a}_i
\end{aligned}$$

例如，假设我们用点的位置、中间点（$u = 0.5$）处的一阶导数和二阶导数定义一条二次曲线段。

$$\begin{aligned}
\boldsymbol{p}_0 = \boldsymbol{f}(0.5) &= \boldsymbol{a}_0 + 0.5^1 \quad \boldsymbol{a}_1 + 0.5^2 \quad \boldsymbol{a}_2 \\
\boldsymbol{p}_1 = \boldsymbol{f}'(0.5) &= \boldsymbol{a}_1 + 2 \quad 0.5 \quad \boldsymbol{a}_2 \\
\boldsymbol{p}_2 = \boldsymbol{f}''(0.5) &= 2 \quad \boldsymbol{a}_2
\end{aligned}$$

约束矩阵为

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0.5 & 0.25 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

基矩阵为

$$\boldsymbol{B} = \boldsymbol{C}^{-1} = \begin{bmatrix} 1 & -0.5 & 0.125 \\ 0 & 1 & -0.5 \\ 0 & 0 & 0.5 \end{bmatrix}$$

15.3.4　三次方程的基矩阵

三次多项式在图形学中是很常用的（见15.5节），不同形式的三次方程的导数和本节中看到的导数一样。作为练习我们将用另一个例子。

一个非常有用的三次多项式是埃尔米特（Hermite）形式,其中要指定始点和终点的位置和一阶导数，即

$$
\begin{aligned}
p_0 = f(0) &= a_0 + 0^1 a_1 + 0^2 a_2 + 0^3 a_3 \\
p_1 = f'(0) &= a_1 + 2\,0^1 a_2 + 3\,0^2 a_3 \\
p_2 = f(1) &= a_0 + 1^1 a_1 + 1^2 a_2 + 1^3 a_3 \\
p_3 = f'(1) &= a_1 + 2\,1^1 a_2 + 3\,1^2 a_3
\end{aligned}
$$

315

所以约束矩阵为

$$
\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix}
$$

基矩阵为

$$
\boldsymbol{B} = \boldsymbol{C}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & -2 & -3 & -1 \\ 2 & 1 & -2 & 1 \end{bmatrix}
$$

我们将在15.5.2节中讨论埃尔米特三次样条。

15.3.5　混合函数

如果已知基矩阵$\boldsymbol{B}$，可以把它与参数向量$\boldsymbol{u}$相乘，得到一个函数向量：

$$
\boldsymbol{b}(u) = \boldsymbol{u}\,\boldsymbol{B}
$$

注意用公式$\boldsymbol{b}(u)$表示向量，是为了强调它的值依赖于自由参数u。$\boldsymbol{b}(u)$的元素为混合函数因为它们定义了如何把控制点的向量的值混合在一起：

$$
f(u) = \sum_{i=0}^{n} \boldsymbol{b}_i(u)\,\boldsymbol{p}_i \tag{15-11}
$$

选择参数u的值也是非常重要的，式（15-11）是一个线性方程，它定义了控制点的线性混合（加权平均）。无论“隐藏”在$\boldsymbol{b}_i$函数中的多项式是几次的，这都成立。

混合函数为描述曲线提供了很好的抽象。任何类型的曲线都可以通过其控制点的线性组合来表示，其中权值可以作为自由参数的任意函数来计算。

15.3.6　插值多项式

316

一般来说，n次多项式可以在n+1个值间插值。如果给定向量$\boldsymbol{p} = (p_0, \cdots, p_n)$和一个递增向量$\boldsymbol{t} = (t_0, \cdots, t_n)$，$t_i \neq t_j$，可以用前一节描述的方法确定一个$(n+1)\times(n+1)$的基矩阵，它给出了一个函数$f(t)$使得$f(t_i) = p_i$。对于任何给定的向量$\boldsymbol{t}$，需要建立并求解一个$n = 1\times n + 1$的线性方程组。这为我们提供了$n$+1个执行插值的基函数：

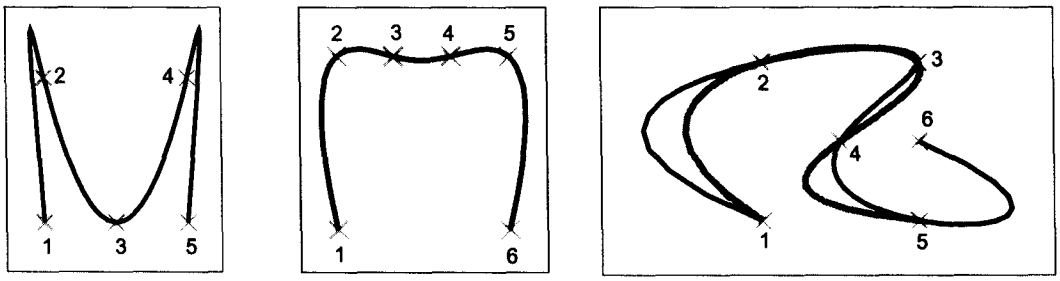
$$f(t)=\sum_{i=0}^{n} p_i b_i(t)$$

这些插值基函数可以通过其他的方式得到。定义基函数最漂亮的方式是拉格朗日形式：

$$b_i=\prod_{j=0,j\neq i}^{n}\frac{x-t_j}{t_i-t_j} \tag{15-12}$$

有很多种表示基函数的形式比拉格朗日形式计算效率更高（具体细节见De Boor(1978)）。

插值多项式为定义曲线（曲线就是对一组点进行插值）提供了一种机制。图15-3显示了一些例子。尽管有能对任意数量的点进行插值的单一多项式，在计算机图形学中很少使用高次多项式表示曲线。相反，我们首选插值样条（分段多项式函数），在15.5.3节中给出了一些理由。 317



（a）通过5个点的插值多项式　(b) 通过6个点的插值多项式　(c) 通过5个点和6个点的插值多项式

图15-3 通过多点的插值多项式。注意到点之间特别的摆动和过冲。（c）中，当第6个点加进去之后，由于插值多项式的非局部性，它完全改变了曲线的形状

15.4 片段连接

我们已经知道如何创建多项式曲线的单个片段，现在可以考虑如何把这些片段连接在一起了。

15.4.1 节点

分段参数方程的基本思想是每一小段只在一定的参数范围使用。例如，如果要定义这样一个函数，两个线段把三个点连接在一起（如图15-4a所示），可以定义：

$$f(u)=\begin{cases} f_1(2u) & \text{如果} 0\leqslant u<\dfrac{1}{2} \\ f_2(2u-1) & \text{如果} \dfrac{1}{2}\leqslant u<1 \end{cases} \tag{15-13}$$

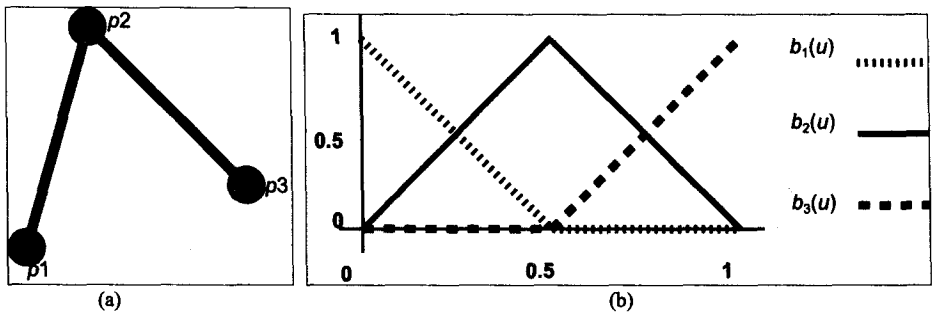


图15-4 （a）连接三个点的两个线段；（b）右边画出的是每一个点的混合函数

其中f_1和f_2分别代表两个线段的函数。注意这里修改了每一个片段的参数的比例，为了便于把它们的方程写成

$$f_1(u)=(1-u)p_1+up_2$$

对于分段函数的每一个多项式，都有一个开始和结束的点（或参数值），一个片段函数开始或结束的点称为节点。例如在式（15-13）中，节点的值是0、0.5和1。

也可以把分段的多项式函数写成基函数的和，每一个函数通过系数改变比例。例如，可以把式（15-13）的两个线段重写成如下的方程：

318

$$f(u)=p_1b_1(u)+p_2b_2(u)+p_3b_3(u) \tag{15-14}$$

其中函数$b_1(u)$定义为

$$b_1(u)=\begin{cases}1-2u & \text{如果 } 0\leqslant u<\dfrac{1}{2}\\ 0 & \text{其他}\end{cases}$$

并且b_2和b_3的定义和上面的相似。这些函数画在图15-4b中。

多项式函数的节点是所有片段的节点的组合。节点向量是由按照升序排列的所有节点值构成的向量。

注意在本节中使用了两种不同的机制组合多项式片段：一是对于不同的参数范围使用独立的多项式片段；二是把分段多项式函数混合在一起。

15.4.2　使用独立的片段

15.3节中定义了在一个单位参数范围内的多项式片段。如果要整合这些片段，就需要把整个函数的参数值转换为分段函数的参数值。最简单的方法是在参数范围[0, n]内定义整个曲线，n表示分段的数目。根据参数的值，可以把它移动到需要的范围内。

15.4.3　连接线段

如果要用两条线段创建一条简单的曲线，需要保证第一条线段的终点和第二条线段的始点在同一个位置。有三种连接两条线段的方式（按照由简单到复杂的顺序）

(1) 用它们各自的端点表示线段，然后对它们使用同一个点。这称为*共享点*（shared point）模式。

(2) 每当第一个段发生变化时，就把第一段终点的值复制到第二段始点的值上。这称为依赖（dependency）模式。

(3) 写出连接的显式方程，当其他参数改变时，通过数字方法执行它。

我们总是选择简单的模式，因为简单模式需要比较少的工作，但同时它们也对线段参数化的
319 方式增加了更多的限制。例如，如果要用线段的中心作为参数（这样用户可以直接指定它），将把每条线段的始点和线段的中心作为它们的参数。那么必须使用依赖模式。

注意，如果使用共享点模式或依赖模式，那么控制点的总数小于$n*m$，其中n表示线段的数目，m表示每条线段控制点的数目。独立片段的许多控制点通过其他片段的函数来计算。注意如果使用直线的共享点模式（每条线段用它的两个端点作为参数，并和邻近的线段共享内部点），
320 或者使用依赖模式（如使用第一个端点和中间点的例子），那么对于n条线段的曲线，要进行$n+1$次控制。

依赖模式有更严重的问题：曲线上一个地方的改变能够在整个曲线上传播，这称为缺乏局部性。局部性的意思是，如果移动曲线上的一点只会影响曲线上的一个局部范围。这个局部的范围可能很大，但它是有限的。如果曲线的控制点不具有局部性，那么改变一个控制点会影响到无限远处的控制点。

为了通过实例说明局部性和缺少局部性的效果，考虑两条线段组成的链，如图15-5所示。一条链中的线段通过其端点进行参数化，利用共享点模式来保持连续性。另一条链则利用端点和中间点定义线段，通过依赖模式把线段连接在一起。这两条链表示了同一条曲线：它们都是n条相互连接的线段的集合。然而由于局部性的问题，对用户来说共享端点形式可能比较方便些。在每一条链中，考虑改变第一个控制点的位置，对于共享端点的链，只有这第一段会发生变化，而对于用中间点的链，所有的线段都会受到影响，如图15-5所示。实际上，在共享端点的链中，对于任何点的移动最多只改变两条线段。在用中间点的链中，当控制点移动后，所有的线段都会随着发生改变，即使链是无限长的。

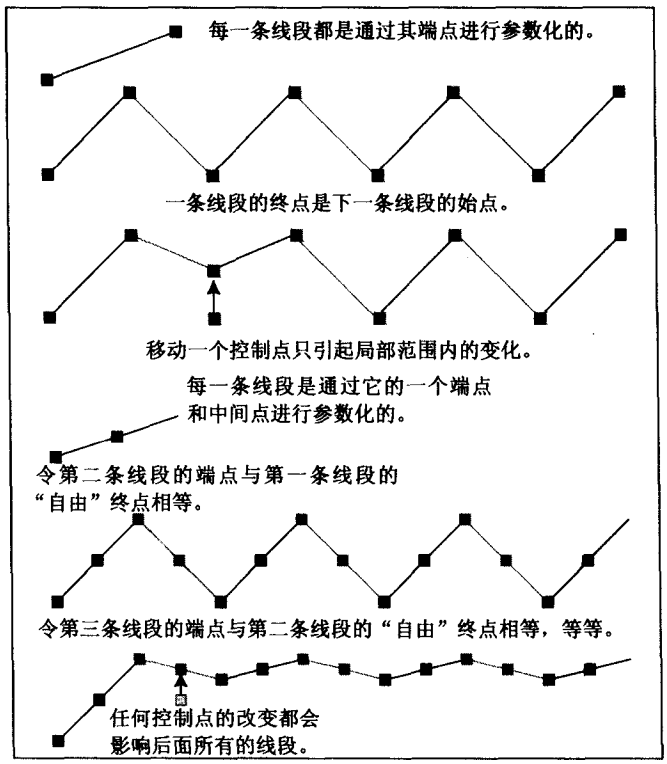


图15-5 具有使用局部控制和非局部控制的线段链

在这个例子中，依赖传播模式是不具有局部控制性的。这并不是永远成立的。有一些直接共享模式没有局部性，而传播模式具有局部性。

我们强调局部性对于控制问题具有方便性。虽然它对于每次修改整条曲线不是很方便，但同样的变化也可以作用在曲线上。这只需要同时移动多个点。

15.5 三次多项式

在图形学中，当用分段多项式表示曲线时，通常用线段或三次多项式表示片段。在计算机图

形学中，为什么三次多项式比较流行，主要有下列原因：

- 分段三次多项式是C^2连续的，这通常可以满足大多数可视化的应用。二次方程提供的C^1
321 平滑经常不能满足要求。更高次的多项式提供的更平滑的性质几乎是不重要的。
- 对于一个点集，三次曲线给出了曲率最小的插值。即如果有一个$n+3$ 的点集，定义通过该点集“最平滑的”曲线（即曲线上的曲率最小），那么可以用n段的分段三次多项式表示曲线。
- 三次多项式具有很好的对称性，曲线的位置和导数都可以在始点和终点处指定。
- 三次多项式在数值计算问题和平滑性之间取得了很好的折衷。

注意我们不是必须要使用三次多项式。它们只是在平滑性和计算复杂度上取得很好的折衷。不同的应用可以有不同的折衷。我们研究三次多项式因为它是最常用的。

三次多项式的规范形式是：

$$f(u)=a_0+a_1u+a_2u^2+a_3u^3$$

正如15.3节中所讨论的，这些规范形式的系数在描述三次片段时是不很方便的。

我们要一种形式的三次多项式，其系数对于控制三次多项式表示的曲线是非常方便的。主要的方便之处是它提供了一种方式，能保证片段之间相互连接，并能保证片段之间的连续性。

每一个三次多项式片段需要4个系数或控制点，这意味着对于具有n段的分段多项式，如果各段之间不共享端点或者不采用依赖模式，则需要$4n$个控制点。通常，每一片段中的某个部分会和相邻的片段共享端点，或者依赖于相邻的片段，所以所有控制点的数量要少得多。同样，注意一个控制点可能就是曲线的位置或者导数。

但是，分段三次多项式没有唯一“最好”的表示方法。不可能有分段多项式曲线具有下面所有的性质：

(1) 曲线的每一片段都是三次的。

(2) 曲线对控制点插值。

(3) 曲线具有局部控制。

322 (4) 曲线具有C^2连续性。

可以同时有任意三个性质，但不能同时有四个。有一些表示方法可以具有上面任意三个性质。本书中将讨论三次B样条曲线，它不对控制点进行插值（但是有局部控制和C^2连续的性质）；基数样条和Catmull-Rom样条都对控制点进行插值，并且提供了局部控制，但是不具有C^2连续性；自然三次样条对控制点进行插值，并且具有C^2连续性，但是没有局部控制。

三次多项式的连续性质是指片段之间（节点处）的连续性。三次片段本身在导数上具有无限的连续性（导数是目前讨论连续性的方式）。注意如果有许多控制点（或节点），曲线的摆动可能会比较大，它看起来可能就不“平滑”了。

15.5.1 自然三次多项式

利用一个分段三次曲线，就有可能创建一条具有C^2连续性的曲线。为此需要在每一个片段的开始指明位置、一阶导数和二阶导数（这样可以保证它与上一个片段的终点具有相同的值）。注意到每一个曲线段的4个参数中有三个都是从片段链中前一个曲线段中得到的。三次多项式的C^2连续链有时候就称为自然三次样条。

对于自然三次多项式的每一个片段，需要通过其端点的位置、一阶导数和二阶导数来对曲线进行参数化。所以控制点是

$$
\begin{aligned}
p_0 = f(0) &= a_0 + 0^1 a_1 + 0^2 a_2 + 0^3 a_3 \\
p_1 = f'(0) &= 1^1 a_1 + 2\ 0^1 a_2 + 3\ 0^2 a_3 \\
p_2 = f''(0) &= 2\ 1^1 a_2 + 6\ 0^1 a_3 \\
p_3 = f(1) &= a_0 + 1^1 a_1 + 1^2 a_2 + 1^3 a_3
\end{aligned}
$$

因此，约束矩阵为

$$
C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}
$$

基本矩阵为：

$$
B = C^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ -1 & -1 & -0.5 & 1 \end{bmatrix}
$$

323

给定一组n个控制点，自然三次样条有$n-1$个三次片段。第一段是利用控制点定义其开始的位置、结束的位置以及开始位置的一阶导数和二阶导数。依赖模式把第一段终点的位置坐标和一阶、二阶导数复制给第二个片段。

自然三次样条的缺点是它不具有局部特性。任何片段的任何变化都会使整个曲线发生变化（至少在发生变化的片段后的部分）。更糟糕的是，自然三次样条有一个不好的特点：曲线开始时的一个小变化，能够引起后面很大的变化。另一个问题是我们只能在曲线的开始控制曲线的导数曲线中的片段是根据其始点确定它们的导数的。

15.5.2 埃尔米特三次多项式

埃尔米特三次多项式，在15.3.4节中有介绍。一条三次埃尔米特样条片段允许指定它的位置和两个端点的一阶导数。一个片段链可以与一个C^1样条连接，只要使一个片段终点的位置和导数与下一个片段的始点相同。

给定一组n个控制点，其中每隔一个控制点是导数值。埃尔米特三次样条包含有$(n-2)/2$个三次片段。样条对控制点进行插值，如图15-6所示，但是只能保证C^1连续。

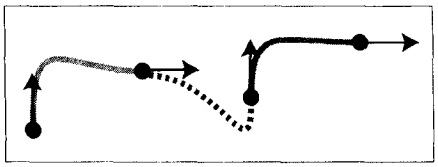


图15-6 由三段组成的埃尔米特三次样条

埃尔米特三次样条很方便，因为他们提供了对形状的局部控制，并且提供了C^1连续性。然而用户必须指定位置和导数，所以必须提供针对导数特别的界面。一种方式是向用户表明，如果它们被放在某个特定点的位置，导数将指向哪个方向。

324

15.5.3 基数三次多项式

基数三次样条（cardinal cubic spline）是由三次多项式片段组成的满足C^1的插值样条。给定

一组n个控制点，基数三次样条用$n-2$个三次多项式片段对所有的点进行插值，除了第一个和最后一个点。

基数样条有一个称为张力（tension）的参数，它控制着点之间插值的“松紧”度。张力参数是在范围[0,1）中的一个数字，它控制着曲线向下一个控制点弯曲的程度。对$t=0$的特殊情况，样条就变成了Catmull-Rom样条。

基数样条中每一个片段都有4个控制点。例如，对于片段i，使用的控制点有i，i+1，i+2和i+3，因为片段和其相邻的片段之间共享三个点。每一个片段从其第二个控制点开始，在第三个控制点结束。曲线开始的导数由第一个点和第三个点之间的向量决定，而曲线结束时的导数由第二个和第四个点之间的向量来决定，如图15-7所示。

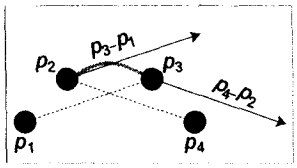


图15-7　基数三次样条的片段对其第二个和第三个控制点（p_2和p_3）进行插值，并且利用其他的控制点确定曲线开始和结束时的导数值

张力参数调整导数缩放的程度。具体地说，用$(1-t)/2$对导数进行缩放。所以对三次样条的约束是

$$
\begin{aligned}
f\ (0) &= p_2 \\
f\ (1) &= p_3 \\
f'(0) &= \frac{1}{2}(1-t)(p_3-p_1) \\
f'(1) &= \frac{1}{2}(1-t)(p_4-p_2)
\end{aligned}
$$

从方程中解出控制点（假设$s=(1-t)/2$）得

$$
\begin{aligned}
p_0 = f(1)-\frac{2}{1-t}f'(0) &= a_0 + (1-\frac{1}{s})\ a_1 + a_2 + a_3 \\
p_1 = f(0) &= a_0 \\
p_2 = f(1) &= a_0 + a_1 + a_2 + a_3 \\
p_3 = f(0)+\frac{1}{s}f'(1) &= a_0 + \frac{1}{s}\ a_1 + 2\frac{1}{s}\ a_2 + 3\frac{1}{s}\ a_3
\end{aligned}
$$

这就得到了基矩阵

$$
\boldsymbol{B}=\boldsymbol{C}^{-1}=\begin{bmatrix} 0 & 1 & 0 & 0 \\ -s & 0 & s & 0 \\ 2s & s-3 & 3-2s & -s \\ -s & 2-s & s-2 & s \end{bmatrix}
$$

因为片段i的第三个点是片段i+1的第二个点，基数样条中相邻的片段是相连接的。类似地，同样的点可以用于确定一阶导数，只要满足C^1连续性。

325

基数样条是非常有用的，因为它们提供了对一组点进行插值的方法，得到的曲线具有C^1连续和局部控制特性。因为只满足C^1连续性，所以有时它们会“打结”。张力参数能够控制插值点之间的形状，如图15-8所示，其中显示了通过相同控制点的一组基数样条。曲线使用相同的控制点，

但其张力参数不同。注意第一个和最后一个点没有被插值。

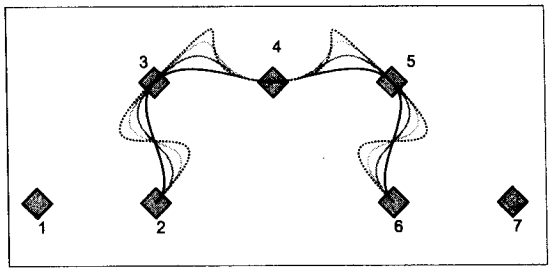


图15-8 通过7个控制点的基数样条，张力参数t的值在变化

给定一组n个插值点，为什么宁可使用基数三次样条（它由$n-2$个三次片段组成），也不使用一个15.3.6节中描述的n次多项式呢？插值多项式的缺点如下：

- 插值多项式很容易对点形成过冲，如图15-9所示。随着控制点的增加，过冲就越来越严重。基数样条在控制点之间的表现比较好。
- 插值多项式的控制不是局部的，改变样条开始的一个点会影响到整个样条。基数样条是局部的：在样条的任何位置最多只由与它相邻的4个点决定。
- 计算插值多项式也不是局部的。计算多项式上的每一个的，都需要访问所有其他的点。计算分段三次多项式上的点，都只需要固定数量的计算，不管总点数是多少。

随着点数的增加，利用插值样条，还有很多数值上和技术上的问题。详细信息参见（De Boor, 2001）。

基数插值样条的缺点是，它不对第一个或最后一个点插值。这个问题很容易解决，只要在序列的最前或最后加入一个额外的点就可以了。基数插值样条也不是连续的，它只在节点处提供了C^1连续性。 326

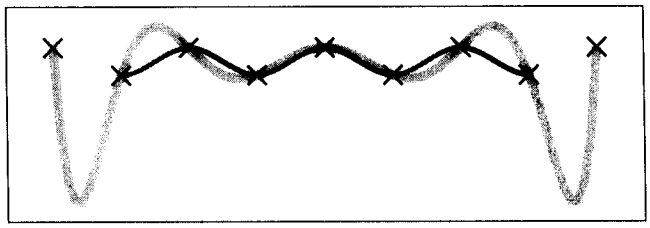


图15-9 对9个点（用小叉表示）插值的样条。粗的灰色曲线表示的是一个插值多项式，细的深色曲线表示的是Catmull-Rom 样条。后者是用7段三次曲线段组成的，每一个曲线段都交替地用不同灰度色调显示

15.6 拟合曲线

通过指定一组点进行插值来控制曲线似乎是一种比较容易的方式。然而在实践中，插值模式通常没有期望的性质，因为它们连续性不好，并且对于控制点之间的形状没法控制。更常用的是拟合模式，即只对控制点进行拟合。利用拟合模式，控制点会影响曲线的形状，但是不精确地确定它。虽然我们放弃了直接指定曲线经过点的能力，但我们得到了更好的曲线行为和局部控制。如果需要对某点进行插值，插值点的位置可以计算出来，这样曲线就可以经过这些插值点。

在计算机图形学中，最重要的两类拟合曲线是贝塞尔曲线和B样条曲线。

15.6.1 贝塞尔曲线

贝塞尔曲线是计算机图形学中自由形式曲线最普遍的一种表示方法。曲线以皮埃尔·贝塞尔（Pierre Bézier）的名字命名，皮埃尔·贝塞尔是一个在其发展过程中起过重大作用的人。贝塞尔曲线有一个很有趣的历史，它是被好几个独立科研组同时研究出来的。

327 贝塞尔曲线是一条多项式的曲线，近似地表示它的控制点。曲线可以是任何次数的多项式。一条d次曲线由$d+1$个控制点来控制。曲线对第一个点和最后一个点插值，其形状直接受其他的控制点的影响。

通常，复杂的形状是通过连接许多低次的贝塞尔曲线形成的，在计算机图形学中，三次（$d=3$）贝塞尔曲线通常被用来组成复杂的曲线。许多流行的图形程序，比如Adobe Illustrator和一些字体表示方案（如Postscript里使用的），都使用了贝塞尔曲线。贝塞尔曲线在计算机图形学中极其流行，因为它们易于控制，有大量有用的性质，并且使用贝塞尔曲线有许多非常有效的算法。

贝塞尔曲线通过下面的方法来构造：

- 曲线分别用$u=0$和$u=1$对第一个和最后一个点进行插值。
- 在曲线的开始位置（结束位置）的一阶导数是通过第一个和第二个控制点（倒数第二个控制点和最后一个控制点）之间的向量来定义的。导数以控制点之间的向量形式给出，这些控制点是按照曲线的次数进行缩放的。
- 在曲线的开始位置（结束位置）的高阶导数取决于曲线开始处（结束处）的多个控制点。n阶导数依赖于开始（最后）的$n+1$个控制点。

例如，考虑三次贝塞尔曲线，如图15-10所示。曲线有4个（$d+1$）控制点。曲线开始于第一个控制点（p_0），结束于最后一个控制点（p_3）。开始处的一阶导数是与第一个和第二个节点之间
328 的向量（p_1-p_0）成比例的。具体地说是$f'(0)=3(p_1-p_0)$。同样，曲线的结尾处的一阶导数为$f'(1)=3(p_3-p_2)$。在曲线开始处的二阶导数通过p_0、p_1和p_2来确定。

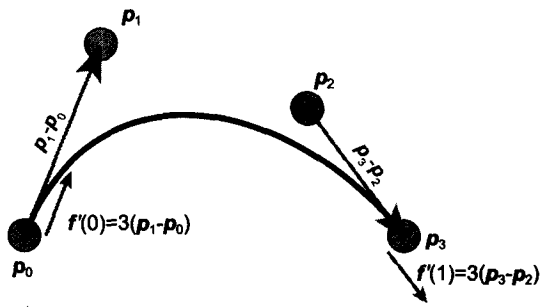


图15-10　由4个点控制贝塞尔曲线。它对第一个和最后一个点进行插值，开始和结束处的导数是前两个点（最后两个点）之间向量的1/3

利用上一段中关于贝塞尔曲线的事实，可以使用15.5节中的方法为它们创建参数方程。开始和最后的插值以及导数的定义是：

$$
\begin{aligned}
p_0 = \quad & f(0) \quad = a_3 0^3 + a_2 0^2 + a_1 0 + a_0 \\
p_3 = \quad & f(1) \quad = a_3 1^3 + a_2 1^2 + a_1 1 + a_0 \\
3(p_1 - p_0) = \quad & f'(0) \quad = 3a_3 0^2 + 2a_2 0 + a_1 \\
3(p_3 - p_2) = \quad & f'(1) \quad = 3a_3 1^2 + 2a_2 1 + a_1
\end{aligned}
$$

它们可以通过基矩阵来求解

$$\boldsymbol{B}=\boldsymbol{C}^{-1}=\begin{bmatrix}1 & 0 & 0 & 0\\ -3 & 3 & 0 & 0\\ 3 & -6 & 3 & 0\\ -1 & 3 & -3 & 1\end{bmatrix}$$

参数方程为

$$\boldsymbol{f}(u)=(1-3u+3u^2-u^3)\boldsymbol{p}_0+(3u-6u^2+3u^3)\boldsymbol{p}_1+(3u^2-3u^3)\boldsymbol{p}_2+(u^3)\boldsymbol{p}_3$$

或者

$$\boldsymbol{f}(u)=\sum_{i=0}^{d}b_{i,3}\boldsymbol{p}_i$$

其中$b_{i,3}$是三次贝塞尔曲线的混合函数：

$$\begin{aligned}b_{0,3}&=(1-u)^3\\ b_{1,3}&=3u(1-u)^2\\ b_{2,3}&=3u^2(1-u)\\ b_{3,3}&=u^3\end{aligned}$$

幸运的是，贝塞尔曲线的混合函数有一个特别的形式，对于任意次数曲线都适用。这些方程是伯恩斯坦基本多项式（Bernstein basis polynormial）并且有一般的形式：

$$b_{k,n}(u)=C(n,k)u^k(1-u)^{(n-k)}$$

其中n是贝塞尔曲线的次数，k是混合函数的个数，从1到n（包括）。$C(n,k)$是二项式的系数：

$$C(n,k)=\frac{n!}{k!(n-k)!}$$

329

给定控制点$\boldsymbol{p}_k$的位置，计算n次贝塞尔曲线（用n+1个控制点）的函数是

$$\boldsymbol{p}(u)=\sum_{k=0}^{n}p_kC(n,k)u^k(1-u)^{(n-k)}$$

图15-11中显示了一些贝塞尔曲线段。

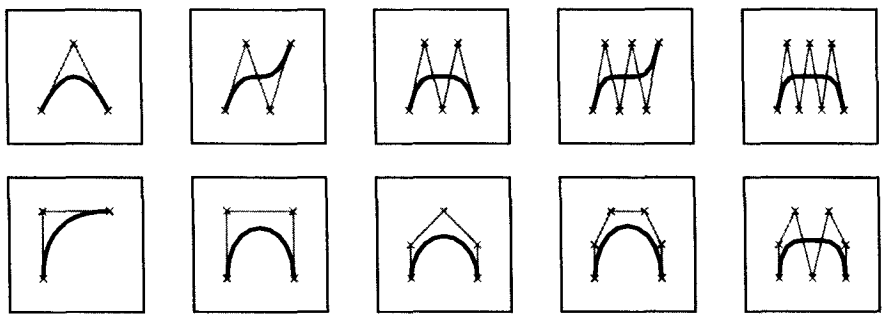


图15-11 二到六次的贝塞尔曲线。控制点用叉表示，同时显示了控制多边形（连接控制点的线段）

贝塞尔曲线段具有几个有用的性质：

- 曲线被有控制点组成的凸包包围着。
- 任何直线与曲线相交的次数都不比与连接控制点的线段的交点多，这称为**变差缩减性**。这个性质在图15-12中有图例说明。

- 曲线是对称的：颠倒曲线控制点的顺序得到同一条曲线，具有相反的参数表达形式。
- 曲线具有仿射不变性。这意味着对控制点进行平移、缩放、旋转或者倾斜，和直接作用在曲线上一样。
- 对于计算和分割贝塞尔曲线有很多简单有效的算法。利用本章后面的算法可以对贝塞尔曲线进行高效分割，得到更多贝塞尔曲线。所以可以用分治法建立有效的算法实现一些重要的任务，如绘制贝塞尔曲线、用线段来近似贝塞尔曲线，以及确定两条曲线的交点等。

330

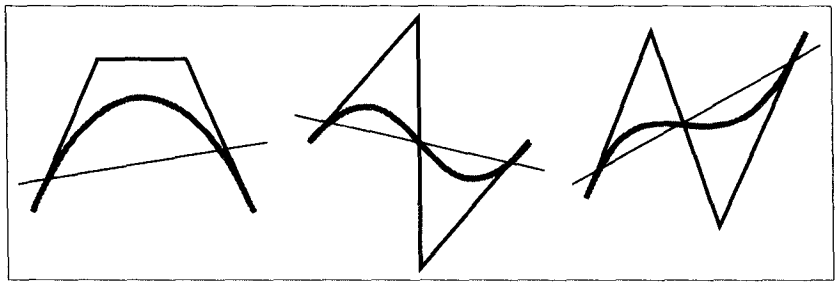


图15-12　贝塞尔曲线的变差缩减性意味着，曲线穿过一条直线的次数不能比它的控制多边形穿过的次数多。所以，如果控制多边形没有摆动，那么曲线也不会摆动。B样条曲线（15.6.2节）也具有这个性质

当要把贝塞尔曲线段连成一起形成样条时，通过共享端点建立各段之间的连续性。然而导数的连续性必须通过其他控制点的位置建立。这样提供给用户的是可以控制曲线平滑性的贝塞尔样条。对于G^1连续性，第一条曲线上倒数第二个点和第二条曲线上的第二个点必须和它们公用的端点在同一条线上。对于C^1连续性，相邻点的距离也应该相等。图15-13对此有图示说明。更高阶的连续性可以通过正确地放置更多的控制点来实现。

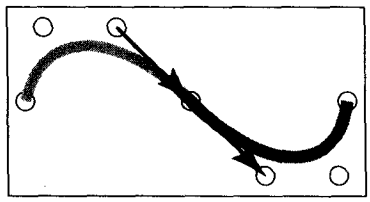


图15-13　两条贝塞尔曲线连接形成一条C^1样条，因为第一个段中最后两个点之间的向量和下一个分段中前两个点之间的向量是相等的

1．贝塞尔曲线的几何解释

贝塞尔曲线可以用几何原理和上面描述的代数方法推导出来。下面对这些几何原理进行概略的介绍，因为它们能直观地解释贝塞尔曲线工作的原理。

设想有一组控制点，我们要用它们产生一条平滑的曲线。简单地用直线连接这些点（形成控制多边形），得到的结果是不平滑的，它可能有尖角。想象通过切除尖角来“平滑”多边形，产生一个平滑一些的多边形，但从数学意义上仍然不“平滑”（曲线仍然是多边形，所以只有C^1连续性）。我们可以重复这个过程，每次都产生一个平滑一些的多边形，如图15-14所示。在有限范围内，如果在无限次重复上述过程，就能得到C^1平滑的曲线。

切角的处理定义为一个细分模式，也就是通过把曲线分成小片段的过程来定义曲线（例如，

331

对它细分）。通过无限次的应用这个过程得到的曲线是极限曲线，如果细分模式的定义是正确的，结果将是平滑的曲线，并且有参数形式。

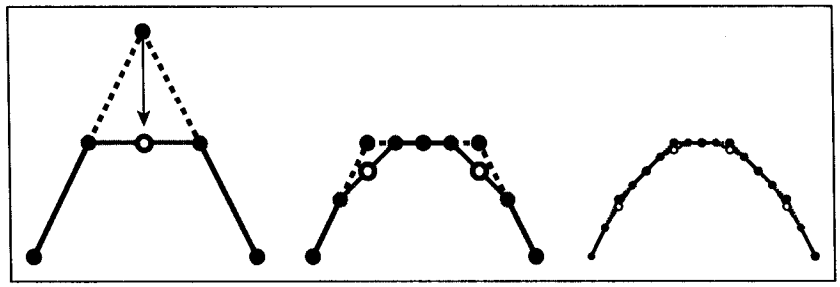


图15-14 对二次贝塞尔曲线的细分过程。从中间把每一条线段分开，然后连接它们的中点（灰色的点和线）。内部的控制点移到新的线段的中点（白圈）上

假设把切角的过程应用到一个角上。给定三个点$(\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2)$，重复地进行“切角”的过程，如图15-15所示。在每一步中，从中间把每一条线段分开，连接中点，然后把角点移动到新线段的中点。注意在这个过程中，新引入的点被移动了一次，然后在后面的过程中保持在那个位置。端点位置不变。

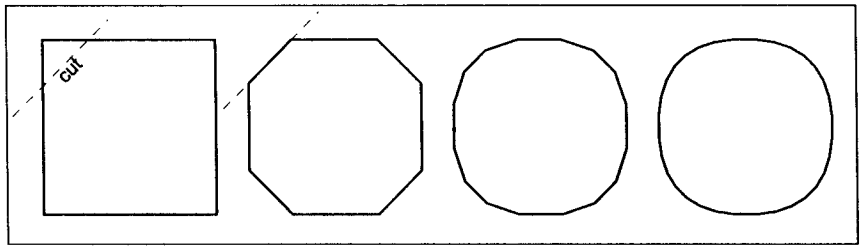


图15-15 通过不断地对多边形进行切角，就得到了一条平滑的曲线

如果计算$\boldsymbol{p}_2$的“新”位置，即中点的中点，得到下列表达式：

$$\boldsymbol{p}'_2 = \frac{1}{2}(\frac{1}{2}\boldsymbol{p}_0 + \frac{1}{2}\boldsymbol{p}_1) + \frac{1}{2}(\frac{1}{2}\boldsymbol{p}_1 + \frac{1}{2}\boldsymbol{p}_2)$$

实际上这个构造方法对沿着每一条线段的距离满足其他比例的情况也是适用的（不只适用于1/2）。令u为每一条线段从开始到结束的距离，我们要把中点放在该线段上，可以把上述公式重写为 332

$$\boldsymbol{p}(u) = (1-u)((1-u)\boldsymbol{p}_0 + u\boldsymbol{p}_1) + u((1-u)\boldsymbol{p}_1 + u\boldsymbol{p}_2)$$

重新组合各项，得到二次贝塞尔曲线函数：

$$\boldsymbol{B}_2(u) = (1-u)^2\boldsymbol{p}_0 + 2u(1-u)\boldsymbol{p}_1 + u^2\boldsymbol{p}_2$$

2. de Casteljau算法

贝塞尔曲线的一个好处是，有比较简单而通用的方法来计算和细分曲线。这个方法就是de Casteljau算法，它利用一系列线性插值计算沿任意次数的贝塞尔曲线的位置，是对前一节介绍的细分模式的扩展。

de Casteljau算法首先用直线连接所有相邻的点，然后在直线上找到满足比例u的点，得到$n-1$个点的集合。接着用直线连接这些点，再次对直线进行插值（仍然按照u），得到$n-2$个点的集合。重复这个过程，直到只有一个点。这个过程的图示在图15-16中。

计算贝塞尔曲线片段上的点的过程同时提供了一种在某点细分曲线的方法。在de Casteljau算法中计算的中间点，形成了一个新的更小片段的控制点，如图15-17所示。

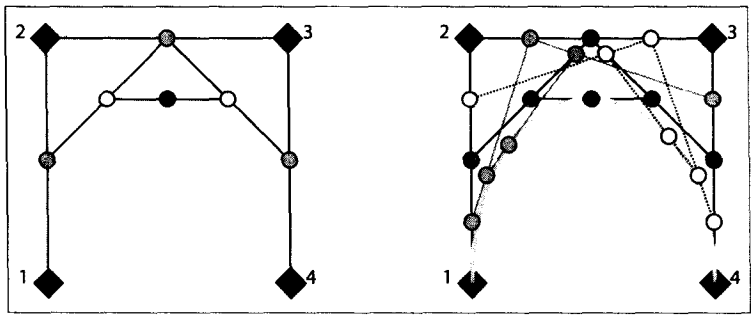


图15-16　三次贝塞尔曲线的de Casteljau算法的示意图。左边的图表示的是按照参数 $u=0.5$ 构造的。右边的图显示的是按照0.25、0.5和0.75构造的

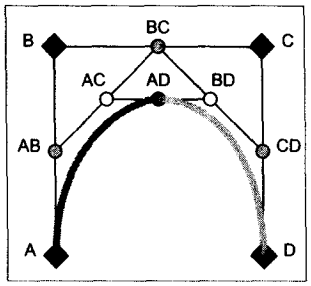


图15-17　用de Casteljau算法分割三次贝塞尔曲线。对初始点（黑色菱形A、B、C、D）进行线性插值产生灰色圆圈（AB、BC、CD），再对新产生的点进行线性插值产生用白色圆圈表示的（AC、BD），然后对新产生的点进行线性插值产生三次曲线上的点AD。这个过程也把具有控制点A、B、C、D的贝塞尔曲线段分成两个贝塞尔曲线段，其控制点分别是A、AB、AC、AD和AD、BD、CD、D

因为分割贝塞尔曲线存在好的算法，所以可以使用分治算法。例如，当绘制贝塞尔曲线的片段时，因为曲线包含在凸包之内，所以可以非常容易地检查曲线是否非常接近一条直线。如果曲
333 线的控制点几乎是共线的，就可以把曲线画成一条直线。否则曲线可以被分成更小的片段，上述过程可以一直重复。同样的算法也用来确定两条曲线的交点。由于存在这样的算法，其他的曲线表示法常常被转化成贝塞尔形式来处理。

15.6.2　B样条曲线

B样条曲线（B-spline）给出了一种方法，用具有 $C^{(d-1)}$ 连续性的 d 次的多项式曲线对 n 个控制点集合进行拟合。和上一章的贝塞尔曲线不同，B样条曲线允许任意阶连续（几乎可以和点的数量相等）。因为这个原因，在计算机图形学中优先选择B样条曲线来定义非常平滑的曲线。如果要通过任意数量的点生成满足 C^2 或更高阶连续性的曲线，B样条曲线可能是正确的选择。

我们可以利用基于B样条基函数的线性组合表示一条曲线。由于基函数本身就是样条，所以它们称为基本样条或者简称为B样条。每一条B样条或基函数是由 $d+1$ 个 d 次多项式组成的。B样条方法提供了定义这些函数的一般方法。

术语B样条专指其中的一个基函数，而不是指通过一组B样条的线性组合创建的函数。然而，这和计算机图形中用的术语是不一致的。通常来说，“B样条曲线”用来指用B样条的线性组合表示的曲线。

在15.3.1节和15.3.5节中已介绍了用其他多项式的线性组合表示一个多项式的思想。在15.4.1
334 节中讨论了用其他样条的线性组合表示一个样条。实际上，其中的例子只是B样条曲线的一个简

单情况。

用其他函数的线性组合表示一个函数的一般形式是

$$f(t)=\sum_{i=1}^{n} p_i b_i(t) \tag{15-15}$$

其中p_i是系数，b_i是基函数。如果系数是点（例如2或3个向量），我们指的是控制点。要使这种方法可行的关键就是要恰当地定义函数b_i。B样条曲线为此提供了一种很通用的方法。

一组B样条曲线可以用很多系数n和一个参数值k[1]定义。k的值比用来生成B样条曲线（$k=d+1$）的多项式的次数大1。

B样条曲线很重要是因为它提供了一个比较通用的生成函数的方法（函数对表示曲线很有用），这些函数有很多有用性质。具有n个控制点，用参数k生成的B样条曲线具有以下性质：

- C^{k-2}连续。
- 由$k-1$次多项式组成。
- 具有局部控制性。曲线的任何位置只依赖于k个控制点。
- 包围在控制点的凸包中。
- 具有变差缩减性，如图15-12所示。

一条用B样条生成的曲线不必对控制点进行插值。

为了介绍B样条曲线，首先看一个具体的简单的情况来介绍这个概念。然后将对这种方法进行泛化，其中可以看到它是很有趣的。因为计算B样条曲线的方法非常通用，我们先介绍完这些泛化方法，然后再介绍它。 335

1. 均匀线性B样条曲线

考虑一组具有以下形式的基函数：

$$b_{i,2}(t)=\begin{cases} t-i & \text{如果 } i\leqslant t<i+1 \\ 2-t+i & \text{如果 } i+1\leqslant t<i+2 \\ 0 & \text{其他} \end{cases} \tag{15-16}$$

每一个函数，在i和$i+2$之间看起来就像一个小三角“帽子”，顶在$i+1$处。每一函数都是分段多项式，其节点在i、$i+1$和$i+2$处。图15-18中显示了其中的两个。

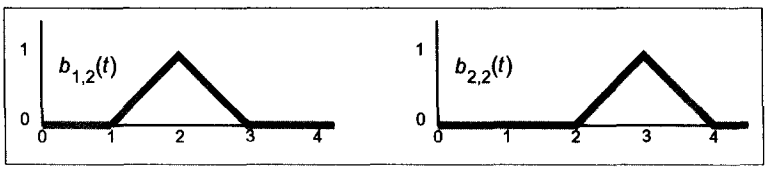


图15-18 参数$d=2$的B样条曲线

每一个$b_{i,2}$函数都是一个一阶线性B样条。因为在后面将要使用B样条其他的参数值，在下标中用2标注这些函数。

注意我们已经把B样条下面的边放在i处。所以第一个B样条（$i=1$）的第一个点位于1。在B样条或者系数向量的元素上的相互作用是从1到n（见式（15-15））。当实现B样条时，和其他讨论B样条的地方一样，通常它们的编号是从0到$n-1$。

1. B样条的参数实际上是在B样条中使用的多项式的次数。虽然这个术语在文献中不是统一的，但是B样条参数k是一个比多项式次数大1的值，这种用法是很广泛的，尽管某些文献（见本章的文献注记）在所有公式中用的都是术语多项式的次数。

使用式（15-15）可以从n个控制点创建一个函数，这个函数用在b_i中创建受系数影响的“总的函数”。如果要用这些（$k=2$）B样条定义总的函数，需要定义一个分段的多项式函数，它在$t=k$和$t=n+1$之间对系数P_i进行线性插值。注意尽管（$k=2$）B样条对所有系数进行线性插值，更高阶的B样条要在特定的情况下才能完成插值，后面将在15.6.3节中讨论。

通过这个简单的情况，可以看出B样条曲线的某些性质。我们用参数k和n表示系数和控制点的个数，把函数写成一般的形式。

- 每一个B样条都有$k+1$个节点。
- 在第一个节点之前和最后一个节点之后，B样条的值都是零。
- 总的样条具有局部控制特性，因为每个系数只与一个B样条相乘，并且B样条只在$k+1$个节点之间不为零。
- 总的样条有$n+k$个节点。
- 每个B样条是C^{k-2}连续的，所以这个总的样条是C^{k-2}连续的。
- 一组B样条，对于在节点k和$n+1$之间的所有参数值的和为1。范围是k个B样条非零的范围。总和为1是非常重要的，因为这意味着B样条具有平移不变性：对控制点进行变换就相当于对整个曲线作变换。
- 在每两个节点之间，B样条是一个次数为$d=k-1$的多项式。所以，总的曲线（所有曲线加在一起）在任意相邻的节点之间也可以用一个d次的多项式表示。

在这个例子中，选择的节点都是均匀分布的。后面将考虑非均匀分布的B样条曲线。当节点在空间中均匀分布时，每一个B样条除了平移的位置之外都是确定的。具有均匀分布节点的B样条有时候称作均匀B样条或者周期性B样条。

2．均匀二次B样条曲线

在前一节中列出的B样条的性质，是针对任意的n和k的。后面将提到构造B样条的一般过程，但这里首先考虑一种具体情况$k=3$。

图15-19中显示了B样条$b_{2,3}$。它是由三个二次片段组成的。样条具有C^1连续性并且只在它跨越的4个节点之间具有非零值。注意到二次B样条曲线由三个片段构成，一个是在节点1和2之间，一个是在节点2和3之间，另一个在节点3和4之间。在15.6.3节中，我们将看到建立这些函数的一般过程，现在我们简单地考察下列函数：

$$b_{i,3}(t)=\begin{cases}\dfrac{1}{2}u^2 & \text{如果 } i\leqslant t<i+1 \quad u=t-i\\ -u^2+u+\dfrac{1}{2} & \text{如果 } i+1\leqslant t<i+2 \quad u=t-(i+1)\\ \dfrac{1}{2}(1-u)^2 & \text{如果 } i+2\leqslant t<i+3 \quad u=t-(i+2)\\ 0 & \text{其他}\end{cases} \tag{15-17}$$

为了使表达式更简单些，把函数的每一部分都写成把它应用在范围0到1之间的形式。

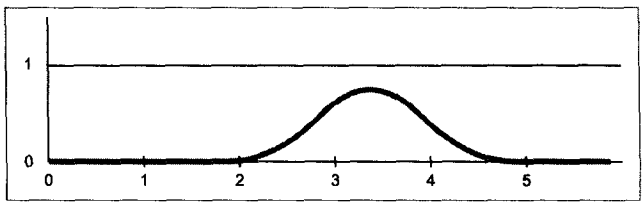


图15-19　具有均匀分布节点的B样条$b_{2,3}$

如果要计算由多个B样条相加得到的总的函数，在任意时刻，只有k（这个例子中是3）个是非零的。一个将是式（15-17）中的第一部分，一个将是式（15-17）中的第二部分，最后一个将是式（15-17）中的第三部分。所以可以把总的函数中的任意片段看作是由次数为$d=k-1$的多项式组成的，该多项式依赖于k个系数。对于$k=3$的情况，可以写出函数：

$$f(u)=\frac{1}{2}(1-u)^2\boldsymbol{p}_i+(-u^2+u+\frac{1}{2})\boldsymbol{p}_{i+1}+\frac{1}{2}u^2\boldsymbol{p}_{i+2}$$

其中$u=t-i$。它定义了在$i\leqslant t<i+1$情况下总的函数的片段。

如果有n个控制点，就可以用B样条产生一条曲线。如果有7个点，那么将需要7个B样条，对于$k=3$的7个B样条如图15-20所示。注意有$n+k$（10）个节点，在从k到$n+1$（节点3到8）的范围内B样条的总和是1。在图15-21中显示了利用这些B样条和一个点集定义的曲线。 338

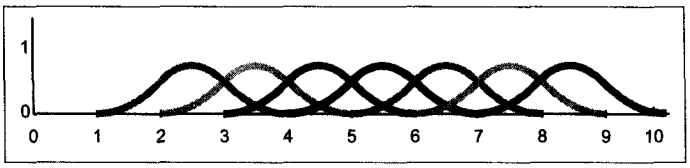


图15-20 7个B样条（$k=3$），节点均匀分布

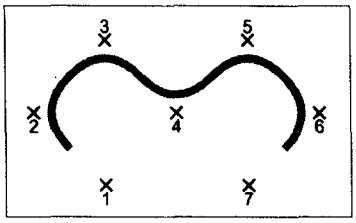


图15-21 由7个二次（$k=3$）B样条组成的曲线，用了7个控制点

3. 均匀三次B样条曲线

因为三次多项式在计算机图形学中是非常流行的，$k=4$的B样条是非常重要的，所以在讨论一般情况之前，首先考虑这个特例。三次B样条是通过4个三次多项式片段定义的。定义这些片段的一般过程后面再讨论，但结果是：

$$b_{i,4}(t)=\begin{cases}\frac{1}{6}u^3 & \text{如果 } i\leqslant t<i+1 & u=t-i\\ \frac{1}{6}(-3u^3+3u^2+3u+1) & \text{如果 } i+1\leqslant t<i+2 & u=t-(i+1)\\ \frac{1}{6}(3u^3-6u^2+4) & \text{如果 } i+2\leqslant t<i+3 & u=t-(i+2)\\ \frac{1}{6}(-u^3+3u^2-3u+1) & \text{如果 } i+3\leqslant t<i+4 & u=t-(i+3)\\ 0 & \text{其他} & \end{cases} \qquad (15\text{-}18)$$

对于$i=1$的三次B样条显示在图15-22中。

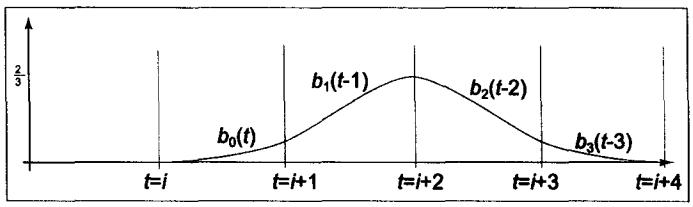


图15-22 具有均匀节点的三次B样条（$k=4$）

我们可以把整个曲线在节点$i+3$和$i+4$之间的函数，写成参数u的函数，u的取值范围是0

到1，并且它受到4个控制点的影响：

$$f(u)=\frac{1}{6}(-u^3+3u^2-3u+1)\boldsymbol{p}_i+\frac{1}{6}(3u^3-6u^2+4)\boldsymbol{p}_{i+1}$$

339

$$+\frac{1}{6}(-3u^3+3u^2+3u+1)\boldsymbol{p}_{i+2}+\frac{1}{6}u^3\boldsymbol{p}_{i+3}$$

利用前面提到的矩阵表示法来改写公式，给出了三次B样条的基矩阵：

$$\boldsymbol{M}_{\mathrm{b}}=\frac{1}{6}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}$$

与从15.5节中的约束推导出的矩阵不同，这个矩阵可以从一个多项式推导出来，该多项式是由下一节中定义的一般B样条过程确定的。

15.6.3　非均匀 B 样条曲线

B样条一个好的特征是，对于任意$k>1$它都能定义。所以如果需要更平滑的曲线，可以简单地增加k的值，如图15-23所示。

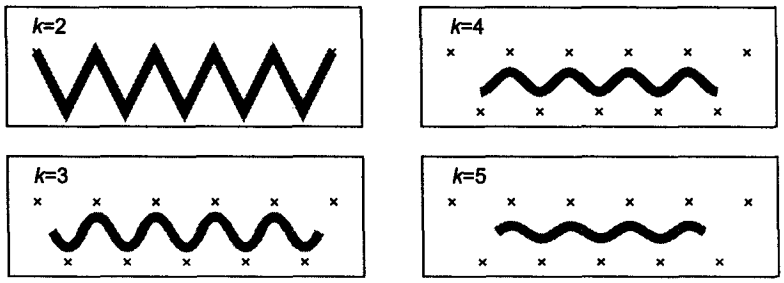


图　15-23

到目前为止，我们已经说过，B样条可以泛化到任意$k>1$和任意$n\geqslant d$。在介绍如何实际地计算这些B样条之前，这是要介绍的最后一种扩展。B样条可以定义在任意非递减的节点向量上。

对于给定的n和k，（通过它们的线性组合得到的）B样条的集合有$n+k$个节点。可以把这些节点写成向量的形式，用t表示。对于均匀B样条，节点向量是$[1, 2, 3, \cdots, n+k]$。然而，只要节点的值是不递减的（例如$t_{i+1}\geqslant t_i$），对于任何长度为$n+k$的向量都可以生成B样条。

340 为什么非均匀分布的节点是非常有用的，主要有两个原因：它允许我们控制每个系数对总的函数产生影响的参数范围，也允许我们重复节点（例如，创建没有间隔的节点），以使得到的函数在这些点上具有特殊的性质。后一种情况在本节的后面考虑。

为B样条定义节点与为插值样条曲线定义插值的位置是一样。它允许我们把曲线的特征和参数的值联系在一起。通过定义一个非均匀节点向量，可以定义每一个B样条系数影响的参数范围。记住B样条i只在节点i和$i+k$之间是非零的。所以与之相联的系数只影响这些参数值之间的曲线。

在序列开始处的插入和删除节点，是对节点值进行控制的很有用的位置。为了进一步说明，考虑在15.6.2节定义的线性B样条（$k=2$）曲线。对于n=4，均匀的节点向量是[1, 2, 3, 4, 5, 6]。曲线被4个控制点的集合控制着，参数从$t=2$到$t=5$。曲线的“终点”（$t=5$）对最后一个控制点进行插值。如果在点集的中间插入一个新的节点，就需要一个更长的节点向量。B样条的局部特性，使插入新节点不会影响曲线终点的值。更长的曲线在其终点处仍然要对最后一个控制点进行插

值。然而，如果继续使用均匀分布节点，新的节点向量将是[1, 2, 3, 4, 5, 6,7]。曲线的终点位于$t=6$，在对最后一个控制点进行插值的位置的参数值，将与插入新节点之前不同。对于非均匀分布节点，可以使用节点向量[1, 2, 3, 3.5, 4, 5, 6]，这样曲线的终点值就不会因此而改变。使用非均匀分布节点的能力，使B样条的局部特性变成一个代数特性，也是一个几何特性。

下面介绍定义B样条的一般方法。给定一组系数n，B样条参数k和节点向量t（长度是$n+k$），下面用递归方程定义B样条曲线：

$$b_{i,1,t}(t)=\begin{cases}1 & \text{如果 } t_i \leqslant t < t_i+1\\ 0 & \text{其他}\end{cases} \tag{15-19}$$

$$b_{i,k,t}(t)=\frac{t-t_i}{t_{i+k-1}-t_i}b_{i,k-1}(t)+\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}b_{i+1,k-1}(t) \tag{15-20}$$

这个方程就是著名的Cox-de Boor循环。它可以用来计算具体B样条曲线的具体值。然而它更常被用来推导公式，如式（15-17）或式（15-18）。

作为一个例子，思考如何推导式（15-17）。把均匀分布节点向量[1, 2, 3,…],$t_i=i$和$k=3$代入 341
式（15-20），得：

$$\begin{aligned}b_{i,3}(t)&=\frac{t-i}{(i+2)-i}b_{i,2}+\frac{(i+3)-t}{(i+3)-(i+1)}b_{i+1,2}\\&=\frac{1}{2}(t-i)b_{i,2}+\frac{1}{2}(i+3-t)b_{i+1,2}\end{aligned} \tag{15-21}$$

继续循环，必须计算下列递归表达式：

$$\begin{aligned}b_{i,2}(t)&=\frac{t-i}{(i+2-1)-i}b_{i,1}+\frac{(i+2)-t}{(i+2)-(i+1)}b_{i+1,1}\\&=(t-i)b_{i,1}+(i+2-t)b_{i+1,1}\\b_{i+1,2}(t)&=\frac{t-(i+1)}{((i+1)+2-1)-(i+1)}b_{i+1,1}\\&\quad+\frac{((i+1)+2)-t}{((i+1)+2)-((i+1)+1)}b_{(i+1)+1,1}\\&=(t-i+1)b_{i+1,1}+(i+3-t)b_{i+2,1}\end{aligned}$$

把上述结果代入式（15-22）中，得

$$\begin{aligned}b_{i,3}(t)=&\frac{1}{2}(t-i)((t-i)b_{i,1}+(i+2-t)b_{i+1,1})\\&+\frac{1}{2}(i+3-t)(t-i+1)b_{i+1,1}+(i+3-t)b_{i+2,1}\end{aligned}$$

为了证明这个表达式和式（15-17）是等价，注意每一个（$k=1$）B样条就像一个开关，只在特定的参数范围内打开。例如，$b_{i,1}$只在i和$i+1$范围内不为零。所以，如果$i\leqslant t<i+1$，公式中只有第一个（$k=1$）B样条为非零，所以

$$b_{i,3}(t)=\frac{1}{2}(t-i)^2 \text{如果 } i\leqslant t<i+1$$

对于式（15-17）的其他部分可以按照同样的方式计算。

重复节点和B样条插值

尽管B样条有许多好的性质，但是利用它们定义的函数不对系数进行插值。如果要用它们定

义一条曲线，要求对特定点进行插值时，它就很不方便了。这里对于如何用B样条对具体点进行
342 插值作一个简短介绍。更完整的细节在本章的文献注记中所列的书中可以找到。

让B样条对系数插值的一种方式是重复节点。如果对于特定的B样条所有的内部节点都有同样的值，那么总的函数将对B样条的系数插值，图15-24显示的就是一个这样的例子。



(a) 均匀的节点　　(b) 非均匀的节点

图15-24　具有7个控制点的二次B样条（$k=3$）曲线。左边用的是均匀节点向量[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]。右边用的是非均匀节点分布[1, 2, 3, 4, 4, 6, 7, 8, 8, 10]，第4个和第8个节点的重复意味着，B样条所有内部的第3个和第7个节点是相等的，所以曲线对与这些点相关的控制点进行插值

通过重复节点实现插值的代价很高：它牺牲了B样条、总的函数及其表示曲线的平滑性。而在样条的开始和结束点，连续性不是一个问题，所以重复节点对于创建端点插值B样条（endpoint interpolating B-spline）是非常有用的。因为第一个（或最后一个）节点的值对于插值是不很重要的，所以为了简化，令前（或后）k 个节点有同样的值以实现插值。

图15-25显示的是端点插值二次B样条。前两个和最后两个B样条与均匀的B样条是不同的。它们的表达式可以通过利用Cox-de Boor循环推导出来：

343

$$b_{1,3,[0,0,0,1,2,\ldots]}(t)=\begin{cases}(1-t)^2 & \text{如果 } 0\leqslant t<1\\ 0 & \text{其他}\end{cases}$$

$$b_{2,3,[0,0,0,1,2,\ldots]}(t)=\begin{cases}2u-\dfrac{3}{2}u^2 & \text{如果 } 0\leqslant t<1,\ u=t\\ \dfrac{1}{2}(1-u)^2 & \text{如果 } 1\leqslant t<2,\ u=t-1\\ 0 & \text{其他}\end{cases}$$

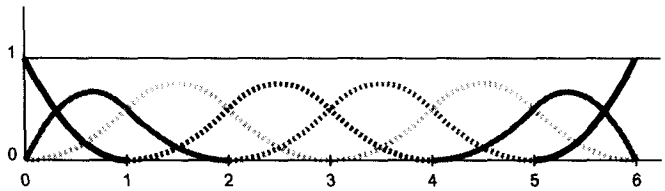


图15-25　对于 $n=8$ 的端点插值二次B样条（$k=3$）。节点向量是[0, 0, 0, 1, 2, 3, 4, 5, 6, 6, 6]。前两个和后两个B样条是非周期的，而中间的四个（用虚线表示）是周期性的，并且和图15-20中的是一样的

15.6.4　非均匀有理 B 样条曲线

虽然B样条有很多优点，但是仍然有一些函数不能准确地用它们来表示。特别是，B样条不能表示圆锥的部分。为了表示这样的曲线，采用了两个多项式的比例。用非均匀B样条曲线表示分子和分母。最一般的形式是非均匀有理B样条（Non-Uniform Rational B-Spline，NURBS）。

非均匀有理B样条把每一个控制点$\boldsymbol{p}_i$与一个权值h_i相连，并对分子分母使用同样的B样条曲线：

$$\boldsymbol{f}(u)=\frac{\sum_{i=1}^{n} h_i \boldsymbol{p}_i b_{i,k,\mathbf{t}}}{\sum_{i=1}^{n} h_i b_{i,k,\mathbf{t}}}$$

其中$b_{i,k,t}$是具有参数k和向量$\boldsymbol{t}$的B样条。

由于非均匀有理B样条除了具有B样条的很多有用的性质外，还有提供了令人惊奇的多种功能，所以在几何建模中，被广泛地应用于曲线和曲面的表示。

15.7 小结

本章中讨论了大量自由形式曲线的表示方法。对计算机图形学最重要的是：

- 基数样条函数用一组三次多项式片段对控制点插值。由于它们具有局部性并且易于计算，所以经常用它们对多项式插值。
- 贝塞尔曲线拟合其控制点，并且具有许多很好的性质和相关算法。为此，它们在计算机图形学中非常流行。
- B样条曲线用B样条函数的线性组合来表示曲线。它们非常通用，并且具有很多有用的性质，比如包围在其凸包内、具有变差缩减性等。当需要比较平滑的曲线时经常用B样条曲线。 344

文献注记

用数学方法表示形状的问题本身是一个完整的领域，一般称为几何建模。表示曲线仅仅是个开始，是对表面和实体进行建模的前一步。在许多几何建模的教科书中都有关于曲线更全面的讨论，如*Geometric Modeling*（Mortenson,1985），这本书对于学习计算机图形学的学生是可以理解的。许多几何建模的书主要讨论平滑的曲线和曲面。教科书*An Introduction to Splines for Use in Computer Graphics*(Bartels, Beatty, & Barsky, 1987)、*Curves and Surfaces for CAGD: A Practical Guide*（Farin, 2002）和*Geometric Modeling with Splines: An Introduction* (E.Cohen, Riesenfeld, & Elber, 2001)给出了关于曲线和曲面表示方法的详细内容。其他的书都集中讨论样条数学，*A Practical Guide to Splines* (De Boor, 2001)是一本很权威的参考书。

曲线曲面表示方法的发展历史是非常复杂的，可以参考*Handbook of Computer Aided Geometric Design*（Farin,Hoschek,& Kim,2002）中的相关章节，或者*An Introduction to NURBS*（D. F. Rogers, 2000）中的讨论。许多思想都是由多个研究组独立提出来的，他们根据不同的原理研究了这个问题。就是因为这个原因，所以很难把一个思想归功某一个人，或者找到一个“起源”点。这样同样导致了文献中的很多差异。如表示法、术语和介绍概念的方式等。

习题

对于习题1至习题4，要求找到约束矩阵、基矩阵和基函数。可以借用软件（例如MATLAB或者OCTAVE）来求取矩阵的逆矩阵。

1. 一条线段：用两个点进行参数化表示：$\boldsymbol{P}_0$沿着线段中25%的位置（u=0.25），$\boldsymbol{P}_1$沿着线段中75%的位置（u = 0.75）。
2. 一个二次多项式：用三个值进行参数化表示：$\boldsymbol{P}_0$是始点的位置（u=0），$\boldsymbol{P}_1$是始点的一阶导数，$\boldsymbol{P}_2$是始点的二阶导数。

345

3. 一个三次多项式：它的控制点是平均分布的（$\boldsymbol{P}_0$对应u=0，$\boldsymbol{P}_1$对应u=1/3，$\boldsymbol{P}_2$对应u=2/3，$\boldsymbol{P}_3$对应u=1）。
4. 一个五次多项式：（矩阵是6×6的）$\boldsymbol{P}_0$是始点的位置，$\boldsymbol{P}_1$是始点的一阶导数，$\boldsymbol{P}_2$是中间的位置（$u=0.5$），$\boldsymbol{P}_3$是中间位置的导数，$\boldsymbol{P}_4$是终点的位置，$\boldsymbol{P}_5$是终点的导数。
5. 拉格朗日形式（式（15-12））可以用来表示对习题3中三次多项式的插值。利用几个不同的参数值证明它产生的基函数的确与习题3中推导出来的相同。
6. 对下列参数方程表示的曲线，设计一个弧长参数化表示。

$$f(u)=(u,u^2)$$

7. 给定一个埃尔米特样条片段的4个控制点，计算等价的贝塞尔片段的控制点。
8. 设三次贝塞尔曲线的控制点为（0，0）、（0，1）、（1，1）和（1，0）。利用de Casteljau算法计算参数$u=0.5$和$u=0.75$时，曲线上对应的位置。画一个草图将有助于解题。

346

9. 利用Cox/de Boor循环推导式（15-16）。

第16章

计算机动画

Michael Ashikhmin

动画（animation）源于拉丁文anima，意思是给予生命、兴趣、精神、运动或者行为的动作、过程或结果。运动是一种定义生命的性质，而很多真实的动画艺术都是关于如何通过运动来描述故事、表现情感或者表达人类性格中的微妙细节。计算机只是实现这些目标的次要的工具。熟练的动画设计者（animator）可以利用计算机的帮助来更快地获得他想要的结果，而不需要把精力集中在他不感兴趣的技术上。不使用计算机的动画，现在通常被称为“传统”动画，它有着悠久的历史。如今，“传统”动画技术仍然活跃在动画艺术领域的很多人使用着。就像在所有已经建立的领域一样，一些经过时间检验的规则已经被明确下来。这些规则从普遍性的高度上指导哪些事情该做，哪些事情应该避免。传统动画的原理同样地应用到计算机动画中，稍后将对它们进行讨论。

然而，计算机不仅仅只是一个工具。除了能使动画设计者的主要工作变得不那么沉闷乏味，它还能提供一些独特的功能。这些功能在过去简直不可想象，或者很难获得。现代建模工具允许相对容易地创建细致的三维模型。绘制算法能够产生一个令人印象深刻的外观。从完全的写实照片到高度地风格化，强大的数字仿真算法能够帮助人们产生期望的基于物理的运动，而这种运动的产生过去对于动画对象特别困难。运动捕捉系统使人们能够记录和使用现实中的运动。这些发展使计算机动画技术在电影和商业广告、汽车设计和建筑、医疗和众多领域的科学研究中得到广泛应用。还出现了许多全新的领域和应用，包括：纯计算机动画电影、虚拟现实系统，当然，还有电脑游戏。

本书的其他章节更直接地讲述了上面提到的许多新技术（比如，几何建模和绘制）。这里，只对直接用于创建和控制运动的技术与算法做一个概述。特别地，我们将不严格地区别并简要描述四种主要的计算机动画技术：

- **关键帧**（keyframing）赋予动画设计者最直接的控制。动画设计者及时在某些时刻给出必要的数据，而后由计算机来填充剩余的部分。
- **过程动画**（procedural animation）包括一些特别设计的数学函数和过程，往往是根据经验确定的，函数的输出模拟了某些特定的运动。
- **基于物理**（physics-based）的技术求解运动的微分方程。
- **运动捕捉**（motion capture）使用专门的设备或技术来记录真实世界的运动，然后将这种运动传递给相应的计算机模型。

在这里，我们完全没有涉及动画领域的艺术性问题。总的来说，使用计算机来创建运动是一个很有吸引力的主题，而我们只触及了它的皮毛。我们希望真正对这个主题感兴趣的读者能够在本章的材料之外继续他们的学习历程。

16.1　动画原理

1987年，在一篇具有开创性的SIGGRAPH论文（Lasseter,1987）中，John Lasseter使传统动画的关键性原理引起了当时正在形成的计算机动画界的关注。迪士尼（Walt Disney）动画制作室中的传统动画设计者早在20世纪30年代就采用了这些原理。其中提到了12条原理：*压缩与伸展*（squash and stretch），时间控制（timing），预期性（anticipation），*跟随动作与重叠动作*（follow through and overlapping action），*慢入与慢出*（slow-in and slow-out），*设计*（staging），*圆弧动作*（arcs），*次要动作*（secondary action），*向前延伸与重点动作*（straight-ahead and pose-to-pose action），*夸张*（exaggeration），*扎实的绘画技能*（solid drawing skill），*感染力*（appeal）。大约20年之后，这些经过时间考验的规则，仍然和过去一样重要。它们能够区别对待自然有趣的动画和机械乏味的动画。另外，对于计算机动画，充分发挥计算机的性能优势，平衡控制和动画设计者的灵活性是非
348 常重要的。虽然这些原理已经广为人知，但是在实践中，对规则的关注程度受到很多因素的影响。尽管设计长片电影的角色动画设计者可能花很多时间去遵守这些建议中的某些部分（如：调试时间控制），而许多游戏设计者相信他们的时间最好花在其他地方。

16.1.1　时间控制

时间控制，或动作的速度是任何动画的核心。事情发生的速度有多快影响到动作的含义、感情状态，甚至与感觉到的物体重量有关。根据动作的速度，角色的头部从左边转到右边，这一动作可以有多种含义。可以是被重物击中后的反应，也可以是在书架上缓慢地寻找书籍，或者是在伸展颈部肌肉。对于当前具体的动作，设置适当的时间速率非常地重要。动作要占有足够的时间才能被注意到，但是要避免太慢甚至无聊的运动。对于涉及录音的计算机动画工程，声音提供了自然的时间锚点（anchor）。实际上，在大多数动画制作中，首先要录制演员的声音，然后使完成的动画与录音同步。因为大而笨重的物体比小而轻的物体移动得慢（更精确的说是加速度更小），时间控制提供了有关物体重量的重要信息。

16.1.2　动作布局

在动画中的任何一个时刻都应该让观众清楚所要表达的思想（动作、情绪、表情）。好的设计（staging），或动作的高层规划，应该把观众的目光引导到当前重要动作关注的地方，无须任何话语，就能有效地告诉观众“现在看这儿，或者看那儿”。了解人类的一些感知能力，有助于解决这个困难的任务。因为人类的视觉系统通常对相对变化有反应，而不是绝对刺激。在一个静止环境中的突然运动，或者在一个繁忙场景中没有运动的部分自然能够吸引人们的注意力。同样的动作，物体轮廓线的变化常常比正面的变化更引人注目（如图16-1a所示）。

在低一些的层次中，每个动作可以分解成三个部分：预期（对动作的准备）、动作本身和*跟随动作*（结束动作）。在许多实例中，动作本身是最短的部分，在一些场景中，甚至最没有意思。
349 例如：踢足球的动作可能包括踢球者局部广泛的准备动作、长时间地盯着离开的球，充分显示了当时的压力、踢球者的情绪状态，甚至对期望动作结果的反应。而动作本身（腿踢球的运动）却相当简单，在这个例子中只有几分之一秒。

预期的目的是使观众对将要发生的动作有提前的准备。如果动作很快，很重要或很困难，那
350 么预期将变得特别重要。为这样的动作创建更广泛的预期可以强调动作的这些特性，并且在快速事件发生的情况，可以确保动作不会被忽略（如图16-1b所示）。

在现实生活中，主要的动作常常引起一个或多个其他*重叠动作*。如图16-1c所示，物体不同

的附加部分或松散部分被主导部分带动，并在主要动作的跟随动作部分持续运动一段时间。此外，下一动作往往在前一动作完全结束前就已经开始了。当一个运动员还在追踪他刚才踢出的球时，他可能已经开始奔跑了。忽略这样的自然过渡，通常会使人感到动作之间有停顿，而且会造成像机器人一样的机械运动。但是，重叠动作必须保持动作的自然性，动画设计者常常加上*次要动作*来使运动更有趣，并借以实现真实复杂的动画。而不使次要动作支配主要动作是非常重要的。

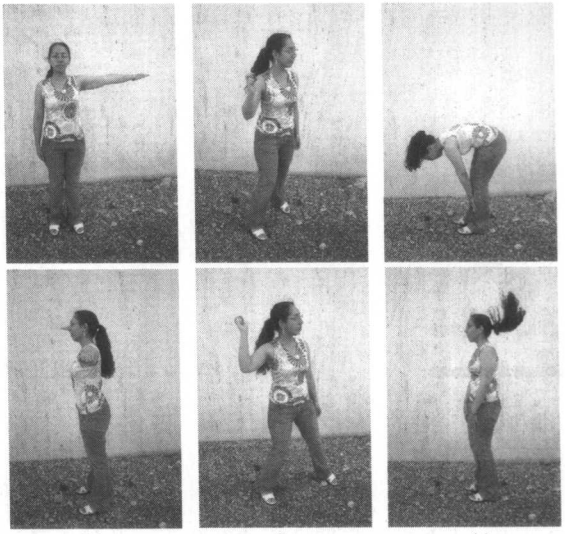


图16-1 动作设计：(a)合适的动作设计对于把观众的注意力吸引到当前重要的运动上是至关重要的。上图中抬手的动作很引人注目，但在下图中却很难注意到。相反，在第一种情况下，鼻子长度的变化可能根本看不见。注意这可能是有意隐藏的，例如：之后要突然展现出来。如果两个运动都该被注意，那么这两个场景的设计都不好。(b)预期可以告诉我们很多关于接下来动作的信息。接下来要发生的动作（扔球）非常短暂，但是我们很清楚将要发生什么。角色越紧张，接下来的动作就感觉越快。(c)手臂伸直的状态对于次要的附属物（头发）特别重要。它的运动跟随主导部分（头部）。头部的运动非常简单，但是导致了头发本身重要的伸直动作。在这个实例中缺少伸直的状态和重叠动作，不能创建一个自然的动画。图片由Peter Shirley和Christina Villarruel提供

16.1.3 动画技术

可以用一些特殊的技术使运动看起来更自然，其中最重要的一种可能就是*压缩和拉伸*了。当物体运动时，这种技术使用一种特殊方法来改变运动物体的形状。在运动方向上一个物体通常会被拉伸，而当一个力作用于物体上时则被压缩，如图16-2所示的一个经典弹球动画。重要的是保持物体的总体体积，以避免物体有变大或缩小的错觉。运动的速度（力量）越大，需要应用的拉伸（压缩）就越大。出于某些原因需要应用这样的变形。对于非常快的运动，物体在连续两帧之间的移动如此之快，以致在当前帧时间和前一帧时间内的物体之间没有重叠。这将导致影象闪失(strobing)（一种走样）。如果物体在运动方向上被拉长，则可以确保更好的重叠，并帮助眼睛克服不舒服的感觉。拉伸/压缩也可以用来显示物体的弹性，越柔软的物体将产生越大的变形。如果表现的是刚性物体，那么当它运动时，形状将保持不变。

自然的运动很少是沿着直线发生的，所以在动画中通常应该避免这种情况，应该用圆弧代替。同样地，非现实世界中的运动可以立即改变它的速度——这需要对物体施加无限大的力。在动画中这种情形也应该避免。特别地，运动应该逐渐地开始和结束（*慢入与慢出*）。手绘的动画有时

通过向前延伸的动作来实现这种效果。动画设计者先开始第一帧，然后按顺序一帧接一帧地绘制直到结束。姿态到姿态的动作也被称为关键帧，更适合计算机动画。在这种技术中，通过一系列间隔相对稀疏的关键帧，可以仔细地进行动画设计。关键帧设置好后，再填充动画的剩余部分（中间帧）（如图16-3所示）。这不仅允许精确的时间控制，而且允许计算机使用下一节中的算法来完成最无聊的处理部分（中间帧的创作）。

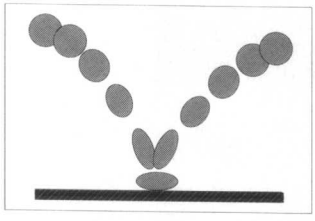


图16-2　应用压缩和拉伸原理的经典例子。注意在动画过程中，弹球的体积应该大体上保持不变

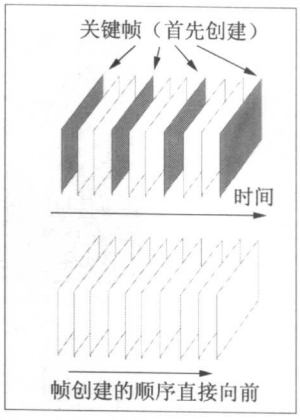


图16-3　关键帧（顶部）促进了动作细节的设计，而向前延伸的动作（底部）得到更自然的效果

为了得到更好的艺术效果，或者为了强调动作或角色中一些特殊的属性，上面提到的所有技术在使用时几乎都可以带有一定程度的夸张。最终的目的是制作出一些观众喜闻乐见的、生动有趣的作品。非常复杂或过于对称的角色和动作往往不那么生动有趣。为了创作出优秀的作品，传统动画设计者需要扎实的绘画技能。同样地，计算机动画设计者当然也应该了解计算机图形学，并且对所使用的工具有扎实的知识。

16.1.4　人工控制与自动化方法

在传统动画中，动画设计者对制作过程的所有方面具有完全的控制，没有什么能影响作品最终的样子，因为它的每个细节都是设计者设计的。这种灵活性的代价是每一帧都必须手绘，这是非常耗时耗力的工作。在计算机动画中，这两者之间有着非常清楚的折中。一方面，让动画设计者对最终结果有更多的直接控制，但是要求他做更多的工作。而另一方面，可以依靠自动化程度更高的技术。这种技术只要求设置几个输入参数，但是却对结果的性质提供了很少控制方法或没有提供任何控制方法。一个好的算法应该能够提供充分的灵活性，只向动画设计者要求直觉的、易于提供的信息和任何他感觉为得到预期效果必要的信息。然而，完美的依从这种要求在现实中是不可能的，因为这样使它像是一个思想阅读器，我们鼓励读者从是否达到这种平衡的角度来评估所有计算机动画技术。

16.2　关键帧

当应用于三维计算机动画时，术语关键帧容易引起误解，因为通常没有涉及实际完整的帧（即：
352 图像）。在任何给定时刻，要模拟的三维场景由一个数值集合指定：所有物体的中心位置、它们的RGB颜色值、在每个轴上应用于每个物体的缩放值、复杂物体中不同部分之间的模型变换、相机的位置和方向、光源的强度等。为了模拟一个场景，部分数值必须随时间变化。当然可以直接在每

一帧中设置这些值，但是这样效率并不高。除此之外，在时间中一些重要时刻（关键帧t_k）可以沿动画的时间轴针对每一个参数进行选择，而且只能设定这些选中帧中的参数值（关键值f_k）。我们把关键帧和关键值的组合(t_k, f_k)简单地称为一个关键点。对于不同的参数，关键帧不必相同，但是同步地设置关键点，至少是一部分关键点往往是合理的。例如：分别针对物体的x、y和z坐标选择的关键帧可能完全是相同的帧，这形成单一的位置向量关键点$(t_k, \boldsymbol{p}_k)$。但是，这些关键帧可能与针对物体方向或颜色选择的关键帧完全不同。相邻的关键帧离得越近，动画设计者对结果的控制就越多。然而，必须估计设置关键点所要多做的工作开销。因此，通常在简单相关的动画部分中关键点之间会有很大的间距，而把关键点都集中到发生复杂动作的部分中，如图16-4所示。

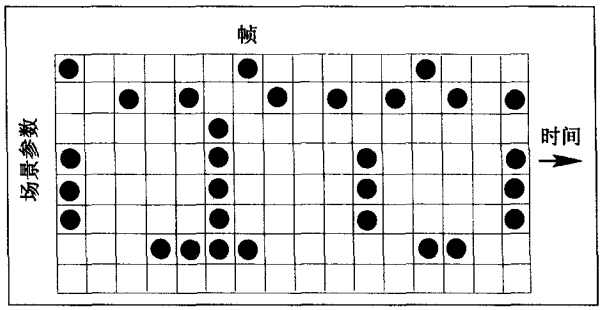


图16-4 设置关键点（上面的黑色圆圈）的不同模式可以同时应用于同一场景。假设在这部分之前和之后有更多的帧

一旦动画设计者设置了关键点(t_k, f_k)，系统就必须计算出所有其他帧的值。虽然我们最终只关心一些离散的值，但是将其视为经典的插值问题是比较方便的。它通过一组给定的数据点集，拟合出一条连续的*动画曲线*$f(t)$（图16-5）。曲线拟合算法的深入讨论见第15章，这里不再赘述。 353
由于动画设计者开始只提供了关键点，而没有导数（正切值），所以在动画中，直接通过关键点来计算所有必要信息的方法更合适。曲线上参数变化的速度由曲线的导数决定，和时间$\mathrm{d}f/\mathrm{d}t$有关。因此，为了避免速度上突然的跳跃，一般要求C^1连续。通常不要求动画曲线具有更高阶的连续性，因为二阶导数对应于加速度或外加作用力，这在现实世界中可能会突然改变（球撞击结实的墙壁），而且更高阶的导数并不直接对应于任何物理运动参数。出于这些考虑，Catmull-Rom样条是创建初始动画曲线的最佳选择之一。

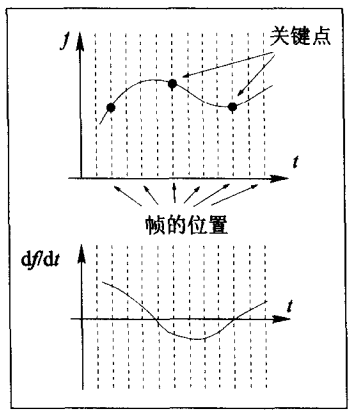


图16-5 一条连续的曲线$f(t)$是通过由动画设计者指定的关键点来拟合的，即使只有在帧的位置上的值有意义。函数的导数给出了参数变化的速度，首先由拟合程序自动确定

大部分动画系统允许动画设计者仔细地对初始曲线进行交互式编辑，包括插入更多的关键点、调整已存在的关键点或自动修改计算的正切值。另一个有助于调整曲线形状的技术是TCB控制（TCB代表张量（tension）、连续量（continuity）和偏移量（bias））。其思想是引入三个新的参数，通过调节一个关键点处输入和输出的正切值来改变关键点附近的曲线形状。由于关键点将时间分成相同的间隔，彼此之间相距 Δt，所以内部关键点(t_k, f_k)处的输入正切T_i^{in}和输出正切T_i^{out}的标准Catmull-Rom表达式可以改写为

$$T_k^{\text{in}} = T_k^{\text{out}} = \frac{1}{2\Delta t}(f_{k+1} - f_k) + \frac{1}{2\Delta t}(f_k - f_{k-1})$$

TCB样条修改后的正切值为

$$T_k^{\text{in}} = \frac{(1-t)(1-c)(1+b)}{2\Delta t}(f_{k+1} - f_k) + \frac{(1-t)(1+c)(1-b)}{2\Delta t}(f_k - f_{k-1})$$

$$T_k^{\text{out}} = \frac{(1-t)(1+c)(1+b)}{2\Delta t}(f_{k+1} - f_k) + \frac{(1-t)(1-c)(1-b)}{2\Delta t}(f_k - f_{k-1})$$

通过缩放输入和输出的正切值，张量参数t控制着关键点附近曲线形状的尖锐程度。正切值越大（张量越小），关键点附近的曲线形状就越平坦。偏移量b使动画设计者能够有选择地增大与关键点相邻的点将曲线拉近到一条直线的程度，该直线连接关键点和其左邻关键点（b接近1，“过冲”）或其右邻关键点（b接近–1，“负过冲”）。连续量c取非零值时，输入和输出正切值不相等，这使动画设计者可以扭曲关键值处曲线。实际上，有用的TCB参数值通常限制在区间[–1，1]上，
354 默认值$t = c = b = 0$对应于原始Catmull-Rom样条。图16-6展示了一些可能的曲线形状。

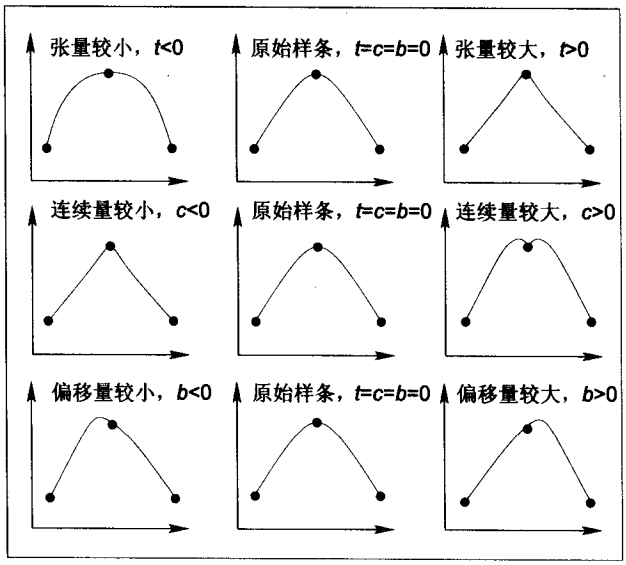


图16-6　使用TCB控制修改默认的插值样条（中间列）。注意所有的关键点都保持相同位置

16.2.1　运动控制

到目前为止，我们已经讲述了如何通过定位关键点和适当地调节关键点处的正切值来控制动画曲线的形状。但是，当要控制物体移动的路径和沿着路径移动物体的速度时，以上的这些就不够了。假定以空间中一组位置作为关键点，自动曲线拟合技术可以用这些关键点来拟合一条曲线。但是，由此而得到的运动是有限制的，它只能使物体运动到相应的关键帧中指定的关

键位置p_k，而关于两个关键点之间的运动速度却没有直接的表示，这会产生问题。例如：假设物体沿着x轴移动，以每秒11m的速度运动了1s，然后以每秒1m的速度运动了9s。则物体将在10s后到达$x = 20$的位置。如果动画设计者设置的关键点是(0,0)和(10,20)，则这样的结果符合关键点的要求。然而，事实上，当设置这些关键点时，实际希望一般不会是这种急速不平稳的运动。动画设计者可能更想得到每秒2米的匀速运动。虽然通常不会出现前述那种极端情况，但是，由标准拟合程序产生的多项式曲线，在两关键点之间确实表现出了非匀速运动，如图16-7所示。虽然，对于一些参数来说能够接受（在限制之内），因为人类的视觉系统并不能很好地确定非均匀变化的速率（像颜色或旋转率），但是，我们仍必须更好地确定物体的位置p，使p处的速度完全符合日常的经验。 355

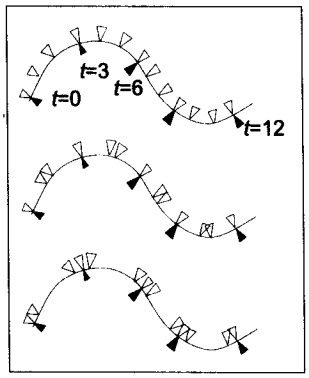


图16-7 这三个运动都沿着同样的二维路径，并且都满足黑色三角形末端表示的一组关键值。白色三角形末端表示物体间隔为$\Delta t = 1$的位置。两个关键点之间的匀速运动（图中最上面）可能更符合动画设计者的希望，然而自动拟合程序只能产生其他两种运动情况

我们将先区分拟合程序与动画中使用的曲线参数化。当通过定位关键点来拟合一条曲线时，可将其视为某个参数u的函数$p(u)$。它将描绘出曲线在空间中的几何形状。弧长s是曲线的实际长度。对于动画设计者，要控制沿已经存在的曲线的运动，一种自然的方式是指定一个额外的函数$s(t)$，它表示物体在任何给定的时间内沿曲线应该移动的距离。为了得到空间中实际的位置，还需要一个辅助函数$u(s)$，它根据给定的弧长s计算参数u的值。根据给定的时间t来计算物体位置的完整过程，就是把这些函数复合起来（见图16-8）：

$$P(t) = P(u(s(t)))$$

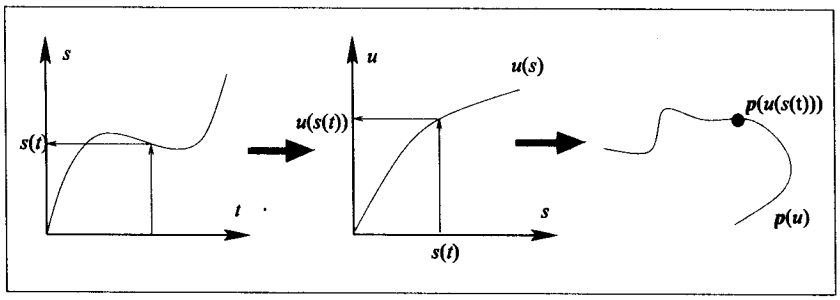


图16-8 为了得到给定时间t的空间位置，首先利用用户指定的运动控制得到沿曲线移动的距离$s(t)$，然后计算相应的曲线参数值$u(s(t))$。前面拟合的曲线$p(u)$现在可以用来寻找位置$p(u(s(t)))$

一些标准的函数可以作为时间-距离函数$s(t)$。其中最简单的一个是与恒定速度有关的线性函

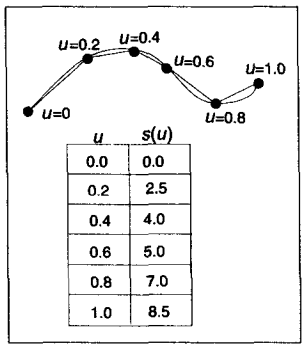
数：$s(t) = vt$，其中v = 常数。另一个常见的例子是带恒定加速度a的运动（初始速度为v_0），它可以用抛物线$s(t) = v_0t + at^2/2$来描述。因为在这里速度是逐渐变化的，所以这个函数可以帮助建立渐进渐出的行为模型。更一般地，$s(t)$的斜率给出了运动的速度，而斜率为负值时，表示运动沿曲线反向进行。为了达到最大的灵活性，动画设计者可以通过动画系统交互地编辑$s(t)$。时间-距
356 离函数并不是控制运动的唯一方法。对于用户来说，在某些情况下，指定时间-速度函数$v(t)$，或者时间-加速度函数$a(t)$可能更方便。因为这些是$s(t)$相应的一阶和二阶导数。使用这种控制，系统首先会结合用户的输入来重新确定时间-距离函数（对于$a(t)$则需要两次计算）。

曲线参数u和弧长s的关系是由系统自动确定的。实际上，系统可以首先根据参数u（即$s(u)$的反函数）来决定弧长。使用这种函数，对于任何给定的S，可以通过解关于未知数u的等式$s(u) - S = 0$，来得到$u(S)$。对于大部分曲线，函数$s(u)$不能用封闭解析式表示，而需要使用数值积分（见第14章）。标准的数值求解过程（例如：牛顿迭代法）能够直接解出等式$s(u) - S = 0$的u。

另一种方法是把曲线本身近似为一组点$\boldsymbol{p}_i$间的线段，点$\boldsymbol{p}_i$可在一组充分密集的空间参数值u_i处计算。然后创建近似弧长的数据表：

$$s(u_i) \approx \sum_{j=1}^{i} \| \boldsymbol{p}_j - \boldsymbol{p}_{j-1} \| = s(u_{i-1}) + \| \boldsymbol{p}_i - \boldsymbol{p}_{i-1} \|$$

因为$s(u)$是u的非递减函数，所以通过简单地搜索表格（见图16-9）就可以找到包含值S的时间间隔。对时间间隔u的终点做线性插值，最终就可以得到$u(S)$。如果要求更高的精度，可以先用牛顿迭代算法做几步迭代计算，然后将得到的值作为计算开始点。



u	s(u)
0.0	0.0
0.2	2.5
0.4	4.0
0.6	5.0
0.8	7.0
1.0	8.5

图16-9　创建一个$s(u)$表，曲线可以由大量连接点的线段来近似。这些点位于曲线上参数值等量增加的地方。根据给定的S值，可以从表中查找到时间间隔u。以图上的曲线为例，与S=6.5位置相对应的u值位于u =0.6和u =0.8之间

16.2.2　旋转插值

对于描绘场景的大部分参数，以上介绍的方法可以用来对关键点集进行插值。三维旋转是一种重要的运动，对此有更多专用的插值方法和表示法。因为将标准方法应用于三维旋转常常会导致严重的实际问题。旋转（物体方向的一种改变）是除平移外唯一能够保持物体外形不变的运动。
357 因此旋转在刚性物体动画中扮演着特殊的角色。

确定物体朝向有很多方法。首先，可以使用第6章描述过的变换矩阵。然而，旋转矩阵的朴素（逐个元素进行的）插值法并不能产生正确的结果。例如,在二维顺时针和逆时针90° 旋转中间的矩阵是零矩阵：

$$\frac{1}{2}\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

当然，正确的结果应该是单位矩阵对应于无旋转。其次，按照特定的顺序选择绕坐标轴的三次旋转，可以任意地指定旋转方向。这些轴可以固定在空间中（固定角表示法）或嵌入物体中。如果是嵌入物体中，每次旋转后轴都将改变（如图16-10所示的欧拉角表示法）。这三个旋转角可以直接通过标准的关键帧来驱动，但是一个棘手的问题是会产生万向节锁（gimbal lock）。在旋转时，如果三个旋转轴其中之一碰巧与另一个共线，则将产生万向节锁，这将减少一个自由度，如图16-11所示的一个实际装置。这个结果比想象的更常见——向右（或向左）旋转90°，很可能会使物体陷入万向节锁。最后，在空间中选择一个恰当的轴和绕该轴旋转的角度，可以任意确定一个方向。然而，在这种表示方法中，动画相对简单，但组合两个旋转并不容易。即寻找经过连续两次旋转后对应的轴和角度并不容易。因此产生了一种特别的数学表达式，四元数 358
（quaternion），它既适合多个旋转合并成为单个旋转的要求，又适合于动画。

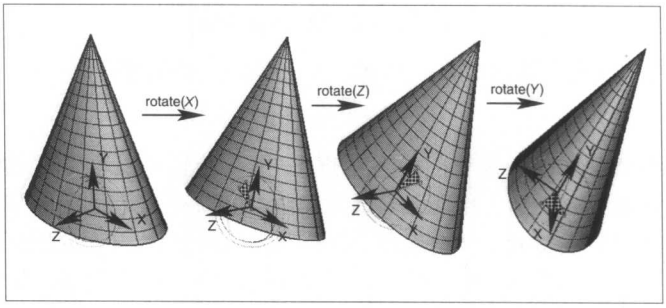


图16-10 通过围绕嵌入物体的坐标轴（*Y*轴总是指向圆锥的尖端）的三次旋转，可以用三个欧拉角指定物体任意的朝向。注意每次旋转都是在新的坐标系中进行。固定角表示法与其非常相似，但是它使用的坐标轴在空间中是固定的，并不随物体旋转

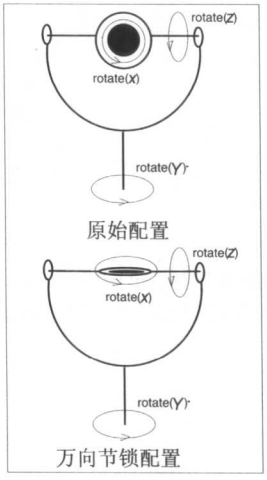


图16-11 在这个例子中，当绕*Z*轴转90度时，出现万向节锁。*X*和*Y*现在看上去都沿相同的轴旋转，因此丢失了一个自由度

给定一个三维向量$\boldsymbol{v} = (x, y, z)$和一个标量$s$，则一个四元数$q$可以将这两个部分合并成一个有四个元素的对象：$q = [s\ x\ y\ z] = [s; \boldsymbol{v}]$。然后为四元数定义一些新的操作。四元数加法定义为分别将标量和向量部分相加：

$$q_1 + q_2 \equiv \left[s_1 + s_2;\ \boldsymbol{v}_1 + \boldsymbol{v}_2\right]$$

标量a和四元数的相乘运算为

$$aq \equiv [as;\ av]$$

更复杂的四元数乘法定义如下：

$$q_1 \cdot q_2 \equiv [s_1 s_2 - v_1 v_2;\ s_1 v_2 + s_2 v_1 + v_1 \times v_2]$$

这里×表示向量叉乘运算。容易发现，与矩阵相似，四元数乘法运算满足结合律，但不满足交换律。我们通常会关注正规化四元数——该四元数的模$|q| = \sqrt{s^2 + v^2}$等于1。最后需要的定义是四元数的倒数：

$$q^{-1} = (1/|q|)[s;-v]$$

为了用从原点开始围绕轴转动的角度ϕ表示一个旋转（其方向可以用标准化向量n给出），标准化四元数是

$$q = [\cos(\phi/2); \sin(\phi/2)n]$$

要旋转点p，可将其变成四元数$q_p = [0; p]$，并计算四元数的乘积：

$$q'_p = q \cdot q_p \cdot q^{-1}$$

它确保有一个零标量部分和作为向量部分的旋转点。复合旋转可以通过四元数的乘积定义，每一个四元数表示一次单独旋转。为了用四元数计算动画，可以将四元数看成四维空间中的点，然后在这个空间中直接地设置关键点。严格地说，保证四元数标准化，应该把插值过程限制在四维空间中一个单位球（一个三维物体）中。但是，球面线性插值（通常称为slerp）会导致非常麻烦的数学运算。而投影到单位球上的四维线性插值非常地简单，而且在实际应用中这已经足够了，如
359 图16-12所示。利用de Casteljau算法进行反复的线性插值可以得到更平滑的结果。

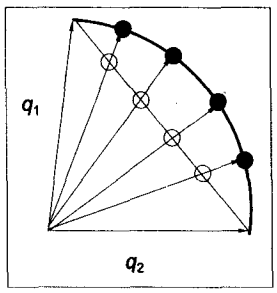


图16-12　四元数的线性插值应该在四维空间中的三维单位球表面上进行。然而，简单地沿着一个四维直线（空心圆圈）进行插值，然后再将结果投影到球面上（黑色圆圈），这通常已经足够了

16.3　变形

虽然把物体变形技术看作建模工具可能更合适些，但是通常把它们和动画方法放在一起讨论。改变物体形状的最简单的例子可能就是非均匀的缩放。更一般的情况是可以把一些函数应用到定义物体的所有点的局部坐标上（即三角形网格的顶点或样条表面的控制多边形），重新配置这些点就创建了新的形状：$p' = f(p,\gamma)$，这里γ是变形函数中使用的向量参数。选择不同的f（并依次把它们组合起来）可以创建出非常有趣的变形。如图16-13所示，一些有用的简单函数有：弯曲（bend），扭曲（twist）和锥化（taper）。通过设定关键帧中变形函数的参数，改变动画形状就变得很容易。这种技术的缺点，一是很难确定一些非标准变形的数学函数，二是变形的结果事实上具有全局的意义，它是对整个物体而言，而不仅只对物体某个部分的变形。

为了能够改变物体的局部，同时提供对结果更直接的控制，可以选择一个顶点，将其移动一个新的位置，并跟随这个种子顶点来调整周围邻近的顶点。受变形影响的区域和物体不同部分上移动的具体数量，由一个衰减函数来控制，该函数根据到种子顶点的距离（一般计算物体的表面距离）而递减。可以将种子顶点的运动设成关键帧来产生动画的形变。

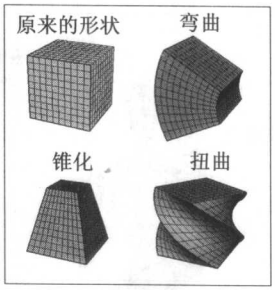


图16-13 常见的全局变形的例子。弯曲和扭曲角，以及锥化的程度都可以用来实现动态的形状改变

更通用的变形技术称为自由变形技术（free-form deformation，FFD）（Sederberg & Parry, 1986）。首先建立一个局部（大多数情况是直线）坐标栅格来封装（encapsulate）物体要变形的部分，而所有相关点的坐标(s, t, u)都根据这个栅格来计算。然后，用户可以自由地改变栅格上格点的形状$\boldsymbol{P}_{ijk}$，扭曲后新的格点为$\boldsymbol{P'}_{ijk}$（图16-14）。利用计算得到的坐标可以重新构造物体，坐标是在原来三元类贝塞尔插值（见第15章）中未扭曲的栅格中计算的，而扭曲后的格点$\boldsymbol{P'}_{ijk}$作为下列表达式中的控制点：

$$P(s,u,t)=\sum_{i=0}^{L}\binom{i}{L}(1-s)^{L-i}s^{i}\sum_{j=0}^{M}\binom{j}{M}(1-t)^{M-j}t^{j}\sum_{k=0}^{N}\binom{k}{N}(1-u)^{N-k}u^{k}\boldsymbol{P'}_{ijk}$$

这里L、M、N是每一维中格点的最大索引值。实质上，为了进行物体变形，格点被看作低分辨率的物体形状，它允许对任意复杂的物体进行平滑的形变，这可以通过进行少量符合直觉调整来实现。FFD格点本身可以看成系统中的普通物体，并且可以进行变换、运动。必要的话，甚至还可以进行更进一步的变形，这使附属于格点上的物体发生相应的改变。例如：移动一个由原始格点构成的变形工具，扭曲后的格点表示物体上一个凸起，结果是使得物体上也出现了一个凸起。 360

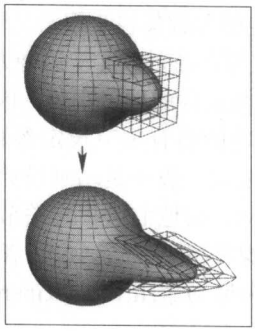


图16-14 调整FFD格点使物体产生变形

16.4 角色动画

关节角色动画通常是通过结合关键帧和特定的变形技术来实现的。动画中使用的角色模型一

般至少由两个主要的层次组成，如图16-15所示。观众最终在作品中看到的是精细的表面运动，表面是由角色的外壳（outer shell）或皮肤（skin）表现出来的。其下的骨架（skeleton）是一个关节层次结构（一棵树），该结构提供了人物的运动学模型，并被广泛地应用于动画。在某些情
361 况下，会在骨架与皮肤之间插入一些与肌肉大致相关的附加中间层。

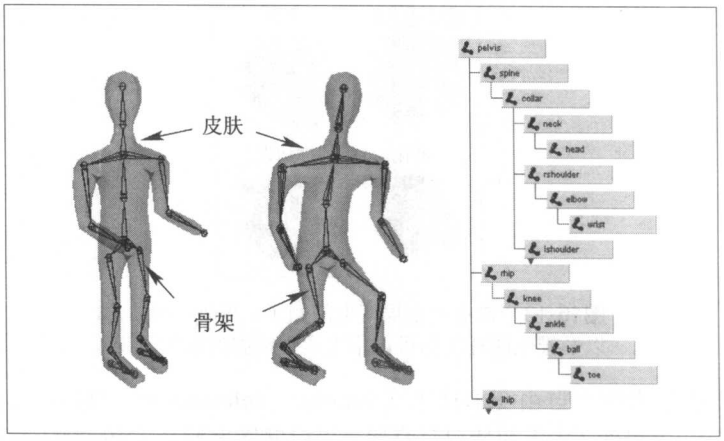


图16-15　（左图）一个关节层次结构和一个骨架，作为角色的运动学抽象；（中间图）重新定位的骨架使附着的不同皮肤对象产生变形。（右图）用来表现骨架的树形数据结构。为了简洁，隐藏了一些节点的内部结构（它们与相应的兄弟节点相同）

骨架中的每个关节都可以作为在它下一层次的父节点。根结点表示整个角色，并在世界坐标系中直接定位。如果提供了局部变换矩阵，该矩阵把关节与其父结点相关联，就可以得到任意关节从局部空间到世界系统（即根结点的坐标系统）的变换，方法是沿着从根到关节的路径把所有的变换连接起来。为了估算整个骨架（即计算所有关节的位置和方向），可以按深度优先遍历全部关节树。对于这样的任务，变换栈是一个比较自然的数据结构。当遍历关节树时，当前复合矩阵被压入栈中，然后，将当前矩阵与存储在关节处的矩阵相乘，得到一个新的矩阵。当回退到父节点时，在开始访问另一个分支之前，应该取消这个额外的变换。可以通过简单的出栈操作来实现。虽然这种普通而简单估算层次的方法在计算机图形学中广泛应用，但是在动画（和机器人技术）中，它被赋予了一个专门的名字——*正向运动学*(Forward Kinematics，FK)。尽管所有变换都可以使用一般的表示方法，使用特定的参数集也是很常见的，如连接长度或关节角。用正向运动学实现的动画，可以直接操纵所有关节的旋转参数。这种方法也允许动画设计者改变关节之间的距离（连接长度），但是必须明白这与肢体的伸展有关，因而看上去常常相当不自然。

362 正向运动学要求用户为涉及运动的所有关节设定参数（图16-16（上））。但是，大部分关节都属于层次的内部结点，它们的运动一般不是动画设计者所关心的。在大部分情况下，动画设计者只是希望它们自己自然地运动，而他们对指定关节链端点的行为更感兴趣，这样的行为一般与某些具体的动作相关，如脚踝和指尖。动画设计者希望系统能够根据终端效应器的运动自动确定所有内部关节的参数。*逆向运动学*(Inverse Kinematics,IK)允许我们这样做（见图16-16（下））。

设$\boldsymbol{x}$为终端效应器的位置，α是一个参数向量，即沿着从根结点到终端关节的链上所有关节需要指定的参数。有时，动画设计者还要直接设定终端关节的方向。在这种情况下，我们假设相关的变量也包含在向量$\boldsymbol{x}$中。但是，为了简便起见，我们写出向量的表达式：

$$\boldsymbol{x}=(x_1,x_2,x_3)^T$$

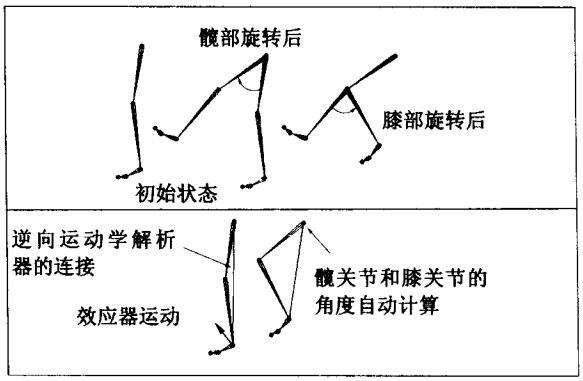


图16-16 正向运动学（上）要求动画设计者把所有关节放到正确的位置上。在逆向运动学（下）中，一些内部关节参数是基于终端效应器的期望运动计算的

因为$\boldsymbol{x}$中的每个变量都是α的函数，故可以写成向量公式$\boldsymbol{x} = \boldsymbol{F}(\alpha)$。如果把内部关节的参数改动一点$\delta\alpha$，那么终端效应器位置的改变$\delta\boldsymbol{x}$可以近似地写为

$$\delta\boldsymbol{x} = \frac{\partial \boldsymbol{F}}{\partial \alpha}\delta\alpha \tag{16-1}$$

这里$\dfrac{\partial \boldsymbol{F}}{\partial \alpha}$是偏导数的矩阵，称为雅可比矩阵：

$$\frac{\partial \boldsymbol{F}}{\partial \alpha} = \begin{bmatrix} \dfrac{\partial f_1}{\partial \alpha_1} & \dfrac{\partial f_1}{\partial \alpha_2} & \cdots & \dfrac{\partial f_1}{\partial \alpha_n} \\ \dfrac{\partial f_2}{\partial \alpha_1} & \dfrac{\partial f_2}{\partial \alpha_2} & \cdots & \dfrac{\partial f_2}{\partial \alpha_n} \\ \dfrac{\partial f_3}{\partial \alpha_1} & \dfrac{\partial f_3}{\partial \alpha_2} & \cdots & \dfrac{\partial f_3}{\partial \alpha_n} \end{bmatrix}$$

在时间的每个时刻，我们知道终端效应器的期望位置（由动画设计者设定），当然也知道终端效应器的当前位置。这两个位置相减，将得到期望的调整量$\delta\boldsymbol{x}$。当一个特定的内部参数改变，而其他的参数固定不变时，雅可比矩阵的元素与终端效应器的坐标的变化有关（见图16-17）。这些元素可以根据任何给定的骨架结构，利用几何关系来计算。式（16-1）中唯一的未知量是内部参数α的变化量。一旦解决了它们，可以更新$\alpha = \alpha + \delta\alpha$。这给出了由FK过程重定位骨架的所有必要信息。

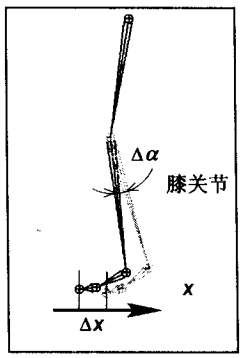


图16-17 偏导数$\partial x/\partial\alpha_{knee}$是$\Delta x/\Delta\alpha_{knee}$的极限。当所有的关节（除了膝关节）固定不变时，可以计算出效应器的位移量

令人遗憾的是，通常式（16-1）的系统不能进行解析求解，而且在大部分情况下无确定解，
363 即未知的内部关节参数α的数目超过了向量x中变量的数目。这意味着不同的骨架运动将导致终端效应器相同的运动。如图16-18所示的一些例子。获得这种系统特定解的方法有很多，包括那些考虑需要给一些现实中的关节加上自然约束的方法（例如：膝关节只能弯向一个方向）。还应该记住计算出的雅可比矩阵只对特定的结构有效，当骨架移动时，它必须被更新。图16-19展现了完整的IK框架。当然，对于IK根关节不必是整个层次的根结点，而且可以将多种IK求解器应用于骨架的独立部分。例如：可以对左右脚使用不同的求解器，而用另一个实现右手抓东西的动画，而每个部位都有它自己的根结点。

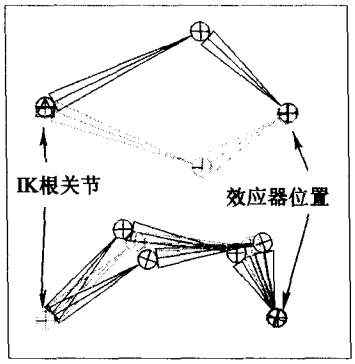


图16-18　多种内部关节的设置可以产生相同的效应器位置：（上）不相交的解，（下）连续解

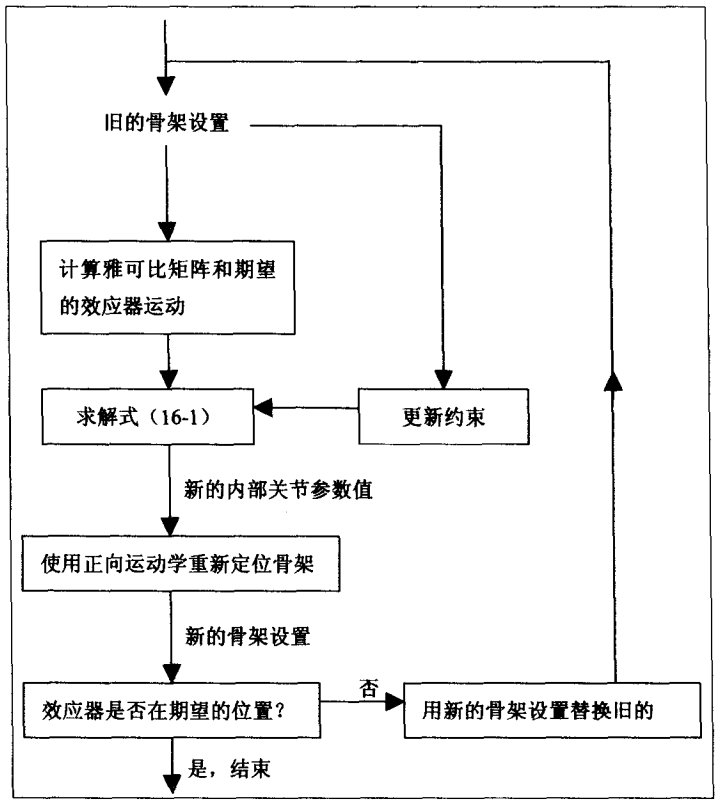


图16-19　逆向运动学算法流程图

FK和IK相结合的方法通常用来实现骨架动画。许多普通的运动（走路或跑圈、抓、伸手等）表现出许多众所周知的互动关节运动（mutual joint motion）模式。这样就能很快创建出看起来自然的运动，甚至可以使用这样的“剪辑”(clip)库。然后，动画设计者可以根据角色的实际参数，对通用的结果进行调整，也可以使其更加个性化。

当一个骨架的位置改变时，它就像将一种特别的变形器应用到角色的皮肤上。通过给每个皮肤顶点分配一个*刚性蒙皮*（rigid skinning）或多个*平滑蒙皮*（smooth skinning）关节作为驱动，运动被传递到表面（见图16-20）。在第一种情况下，皮肤顶点被简单地固定到相应关节的局部空间中，可以是空间中最近的关节，或者是用户直接选择的关节。然后，顶点重复这个关节做过的运动，它在世界坐标中的位置由标准的FK过程确定。虽然这很简单，但是刚性蒙皮使得在靠近关节的区域很难获取充分平滑的皮肤变形，或者实现像呼吸或肌肉运动这样更细微的效果。另外一种专门称为*屈肌*(flexors) 的变形器可以用来解决这个问题。在平滑蒙皮中，多个关节可以根据 364
动画设计者分配给的权值来影响皮肤顶点，对结果提供更细微的控制。位移向量$\boldsymbol{d}_i$可以由影响给定皮肤顶点的不同关节计算出（每个关节都用标准FK计算）。根据它们的权值w_i进行加权平均，计算出最终的顶点位移$\boldsymbol{d}=\sum w_i\boldsymbol{d}_i$。使用最普遍的是标准化权值($\sum w_i=1$)，但这并不是必须的。为达到期望的效果设置平滑蒙皮的权值并不容易，它要求动画设计者有相当的技巧。

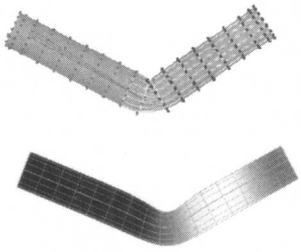


图16-20 上：刚性蒙皮给皮肤顶点分配一个具体的关节。属于肘关节的顶点是黑色的。下：平滑蒙皮可以混合多个关节的影响。肘关节的权值如图所示（颜色越浅，权值越大）。注意靠近关节的皮肤内部更平滑的皮肤变形

16.4.1 面部动画

骨架非常适合来创建角色身体的大部分运动，但是对于现实中的面部动画就不那么方便了。原因是人脸皮肤的运动是直接由附着的肌肉引起的，而不像身体的其他部分，肌肉运动的基本目的是驱动骨架中的骨骼，任何皮肤的变形只是间接的结果。面部解剖分析的结果是很多动态的面部表情。人们将其作为一种主要的交流手段。我们能很好地识别这种面部变化，并能很容易地注意到任何不自然的样子。这不仅对动画设计者提出了特殊的要求，也要求建立高分辨率的面部几何模型。如果要求有真实感效果，还需要精确的皮肤反射性质和纹理。

尽管有可能对面部模型一点一点地设置面部的关键姿态，然后在两者之间进行插值，或者使用基于物理的方法直接模拟底层肌肉结构的行为（见下面的16.5节），但是还有很多更专业的高级方法。特定面部的静态形状可以由一组相对较小的称为*构象参数conformational parameter*的集合来刻画（总体比例、眼睛到前额的距离、鼻子的长度、下巴的宽度等），这些参数可以对普通的面部模型进行变形，使之具有个性化的特征。另外可以用*表情参数*(expressive parameter)集合来描绘动画中面部的动态形状。例如：头部的刚性转动、眼睛张开的宽度、从静态位置开始运动的某个特征点，等等。仔细选择这些参数，就能够通过适当组合不同的参数得到大部分有趣的表情。因此，可以通过标准的关键帧技术来产生面部动画。为了实现更高层的控制，可以使用表情参数产生一组与常见情绪（紧张、悲伤、高兴、愤怒、惊讶等）相关的表情。然后，融合关键的姿态 365

就得到了“轻度悲伤”或“又惊又怒”的面部姿态。相同的技术可以用来表现嘴唇的同步动画。但是，在这种情况下，关键姿态与不同的音素（phoneme）有关。人脸运动编码系统（Facial Action Coding System，FACS）（Eckman & Friesen, 1978）不是用一组有序的静态表情来描绘一个动态表情，而是将动态的面部表情直接分解为一组基本运动，称为运动单元(Action Units, AUs)。这组运动单元基于广泛的心理学研究，它包括如扬起眉毛、皱起鼻子、拉长嘴唇等运动。可以通过组合运动单元来合成一个必要的表情。

16.4.2　运动捕捉

即使有了以上描述的方法，从头开始创建具有真实感的角色动画依然是一项令人生畏的任务。所以，人们很自然地将注意力转向了这样的技术，即记录现实世界中角色的运动，然后将记录应用于计算机生成的角色。这种*运动捕捉*（motion capture, MC）技术主要有两类：电磁式（electromagnetic）和光学式（optical）。

在电磁式运动捕捉中，电磁传感器直接测量出三维空间中的位置（可能还有方向），它提供实时的捕捉结果。这个技术的缺点包括巨大的仪器花费、周围金属物体可能造成的干扰，以及传感器和电池大小。太大的传感器或者电池在进行大幅度的运动时，会成为一种阻碍。在光学运动捕捉中，使用小的有色标记代替传感器，从而使受到的干扰更小。图16-21显示了这种系统的一个操作。最基本的方案是使用两台校准的摄影机记录运动，然后使用简单的三角测量法确定标记的三维位置。可以使用更先进的计算机视觉算法精确地跟踪视频中的多个标记，但是它的计算代价很高，因而在大多数情况下，处理过程不是实时完成的。通常光学跟踪的稳健性要小于电磁方法。在某些帧中某个指定的标记可能被遮挡，标记可能被误识，以及图像中的噪声，这些都是必须要处理的问题，而这些不过是其中的一小部分。我们可以用更多的摄像机从不同的角度来观察运动，以同时提高精度和稳健性，但是这种方法更加昂贵，而且处理数据的时间更长。随着可用计算能力的增长和更好的计算机视觉算法的出现，光学运动捕捉变得更有吸引力了。因为标记不容易受影响，光学方法适用于精细的面部运动捕捉，而且它也可以应用在其他物体上，而不只是人——例如：动物，甚至是风中的树枝。

366

通过附加在表演者身上的一些传感器或标记，可以记录下一组与时间相关的三维位置点集。这些跟踪位置一般选在关节附近。当然，它们仍位于皮肤表面，并不在实际的骨骼关节处。因此，需要一些额外的处理过程将记录位置转换到那些实际的骨骼关节上。例如：在肘或脚踝的两个相反面上放置两个标记，通过平均这两个标记的位置，系统可以更好地得到关节位置。若没有这种额外的处理，由于关节位置的偏移、固有的噪声和测量精度不足，就会出现有噪声的走样。例如：由于运动中物理设备精度不够，当走路或抓取东西时，角色的肢体可能并没有紧密地接触到它们想要触摸的物体，于是骨架可能会产生像滑步（foot-sliding）（滑冰）之类的问题。大部分问题可以利用逆向运动学技术纠正。逆向运动学技术可以明确地影响要求的肢体终端行为。

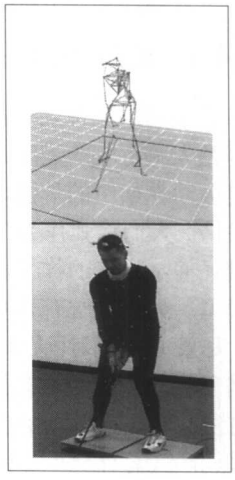


图16-21　光学运动捕捉：可以从附加在表演者身体上的标记中提取骨架运动（图片经Motion Analysis公司授权使用）

现在修改过关节的位置可以直接应用到计算机生成角色的骨架上。此过程假定角色的实际大小与表演者的一致。将记录的运动重定位到一个不同的角色上，更一般的说法是编辑运动捕捉的数据，要求必须要满足一些必要约束（如保持脚贴着地面或不允许肘部向后弯），并且修改后的运动要保持完全自然的效果。一般说来，与原来的运动相比，期望的改变越大，保持结果质量

的可能性就越小。解决这个问题的一种有趣的方法是记录一个巨大的运动集合，然后把从库中的剪辑片段合成在一起，以获得期望的运动。虽然这是当前非常活跃的研究领域，但是运动捕捉技术的一个主要缺点仍然是很难根据动画设计者的需要，来调整记录的运动。

16.5 基于物理的动画

我们的周围是一个被物理定律支配着的世界。许多物理定律都可以被形式化成偏微分方程，或在一些更简单的情况下，形式化成一些普通微分方程组。计算机最初的应用之一就是（现在仍是）求解这些方程。因此，很自然地会尝试使用过去几十年中发展起来的数字化技术来获取计算机动画所需要的真实运动。

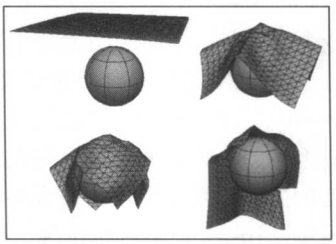


图16-22 真实感织物仿真通常用基于物理的方法实现。在这个例子中，碰撞和重力产生了对织物的作用力

由于其高复杂性和巨大的计算代价，只有在没有其他技术或者其他技术不能产生足够真实结果的情况下，才使用基于物理的动画技术。主要的例子有流体动画（包括由相同方程描述的许多气态现象——烟、云、火等）、织物仿真（如图16-22所示的例子）、刚体运动和弹性物体的精确变形等。在每一个实例中，控制方程和常用的数值分析方法的细节都是不同的，但是许多基本的思想和难点在很多应用中都是适用的。普通微分方程（ODE）和偏微分方程（PDE）已经有许多数值分析求解方法，但是其相关的详细讨论远远超出了本书的范围。为了让读者能初步了解基于物理的方法及其相关问题，在这里简要地提及有限差分法——概念上最简单且最流行的算法系列之一。它已经应用于动画中遇到的大部分（甚至全部）微分方程中。

367

该方法的关键思想是把微分方程用其离散相似体——差分方程来替代。为此，用一个有限点集来表示感兴趣的连续区域，从这个点集中就可以得到解。在最简单的情况下，点定义在均匀的矩形栅格中，如图16-23所示。原来普通微分方程或偏微分方程中的每个导数都可以用栅格点上的函数值来近似。一种方式是从栅格中相邻点的函数值中减去指定点的函数值：

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \approx \frac{\Delta f}{\Delta t} = \frac{f(t+\Delta t)-f(t)}{\Delta t} \text{或} \frac{\partial f(x,t)}{\partial x} \approx \frac{\Delta f}{\Delta x} = \frac{f(x+\Delta x,t)-f(x,t)}{\Delta x} \tag{16-2}$$

368

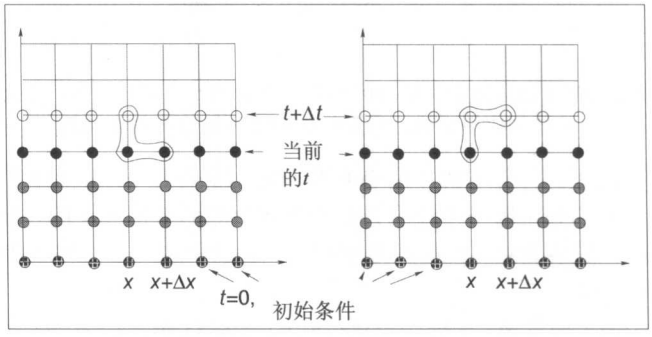


图16-23 包含微分 $\partial f/\partial x$ 和 $\partial f/\partial t$ 的方程的两种不同方案：（左）显式的方案只是通过当前的（黑色圆圈）和过去时间可能的（灰色圆圈）已知量来表示未知量（空心圆圈）；（右）隐式的方案将已知和未知量合到一个方程中，使得必须系统地求解所有的方程。对于这两种方案，要结束计算过程，必须要有右边界值

当然，这些表达式不是唯一的方法。例如，可以使用 $f(t-\Delta t)$ 替换上面的 $f(t)$，然后除以 $2\Delta t$。对于包含时间导数的方程，通过解未知量 $f(t+\Delta t)$ 的差分方程组，能够在沿以 Δt 为时间步长的时间序列中，传播未知函数的值。开始过程需要一些初始条件，即未知函数在 $t=0$ 时的值。在特定的问题中可能需要其他信息，如在区域边界上的值。

在当前时刻，如果已经得到所有其他出现的值，在相应的差分方程 $f(t+\Delta t)$ 中唯一的未知量可以由其他已知量来表示。这种计算 $f(t+\Delta t)$ 的简单方法称为显式方案。而隐式方案混合当前和未来时刻的值，可能会使用下列式子作为 $\frac{\partial f}{\partial x}$ 的近似：

$$\frac{f(x+\Delta x,t+\Delta t)-f(x,t+\Delta t)}{\Delta x}$$

在这种情况下，在每一步都必须解一个代数方程组。

差分方案的选择会显著地影响算法的所有方面。其中最明显的就是精度。在极限 $\Delta t\to 0$ 或 $\Delta x\to 0$ 中，式（16-2）的表达式是精确的。但是对于有限步长，某些方案允许对导数作更好的近似。不同差分方案的稳定性与数值误差随时间增长的速度有关。在实际中误差总是会出现，对于稳定的方案，误差的增长是有界的。而对于不稳定的方案，误差呈指数增长，并很快覆盖搜索的解（见图16-24）。当解中出现一些误差是可以容忍的（事实上，在动画中很少需要物理和工程上的精度要求），但不稳定的结果是完全没有意义的，认识到这一点很重要，因此应该避免使用不稳定的方案。一般说来，显式方案是不稳定的，或者当增大步长时会变得不稳定，而隐式方案是无条件稳定的。隐式方案允许更大的步长（或更小的步数），这就是它受欢迎的原因，虽然它每一步必须解一个代数方程组。显式方案很有吸引力是因为当稳定条件得到满足时它是非常简单的。为一个具体的问题设计好的差分方案和相关算法并不容易，而对于大部分标准情况，建议使用已有的方法。有很多文献都对其中的技术细节进行了详细讨论。

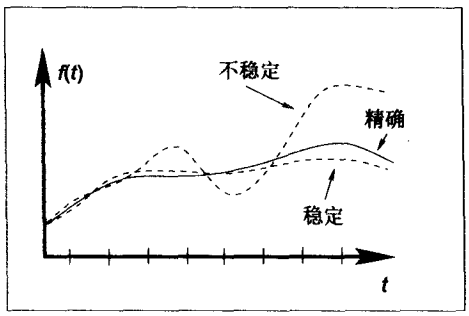


图16-24　不稳定的解开始可能会与精确解相近，但是随着时间的增加，它会程度不定地偏离精确解。对一个具体的应用来说，稳定解的精度可能仍然是不够的

应该记住在许多情况下，计算方程中所有必要的项本身就是一个困难而耗时的任务。例如：
369 在刚体或织物仿真中，大部分作用于系统的力是由物体之间的碰撞引起的。因此，在动画的每一步，必须解决纯几何的，但是非平凡的碰撞检测问题。在这些条件中，较少需要对作用力求解的方案可能会节省大量的计算。

虽然，求解适当依赖于时间的方程得到的解给出了非常真实的运动，但是这种方法有它的局限性。首先，很难控制基于物理的动画的结果。一旦设定初始条件，方程的基本数学性质就确定了，解也就唯一确定了。这留给动画设计者很少的输入空间。如果结果不满足某些要求，我们将只有很少的选择。它们不过是初始条件的调整，改变系统的物理性质，甚至是通过引入虚拟项修改方程组本身，使解“趋向”动画设计者期望的方向。做这些改变需要很强的技巧，也需要理解

基本物理和理论上的数值方法。缺少这些知识，就可能会破坏基于物理的动画所产生的真实感或出现严重的数值问题。

16.6 过程技术

想象一下人们可以写出（并可以用计算机实现）一个数学函数，它能够根据动画设计者给出的引导精确地输出期望的运动。当涉及的“函数”是求解特定微分方程的过程，而“引导”是一组初始和边界条件，或额外的方程项等时，上述基于物理的技术可以看成是这种方法的特例。

然而，如果只关心最终的结果，不一定要采用基于物理的方法。例如：在湖面上的一个简单的恒定振幅波可以用函数$f(x,t)=A\cos(\omega t-kx+\phi)$直接创建。其中频率$\omega$是常数，波向量$k$和相位$\phi$用来获得在时间$t$时二维点$x$的位移。一组具有随机相位和适当选择的振幅、频率和波向量的波可以产生一个非常真实的水面动画，而不需要显式地解任何流体动力学方程。这表明简单的数学函数也可以产生非常有趣的模式或对象。这样的函数，大部分是基于点阵噪声的，在第11章已经做了描述。根据加入时间的函数，能够实现某些复杂现象的动画，这比基于物理的技术更容易且代 370
价更小，而且能够使结果保持非常高的视觉质量。如果*noise*(*x*)是基本模式产生函数，那么通过在点阵中移动参数的位置，可以建立一个依赖时间的变量。最简单的情况是匀速运动：*timenoise*(*x*, *t*) = *noise*(*x* + *vt*)，但是点阵上可能有更复杂而且更常见的运动。图16-25显示了这样一条路径：一条螺旋形的路径。另一种方法是改变用来生成噪声函数的参数。如果外观随时间产生强烈变化——例如：云变得更加紊乱，那么这是一种特别合适的方法。在这种方式下，利用生成静态函数的函数可以模拟云层的动态过程。

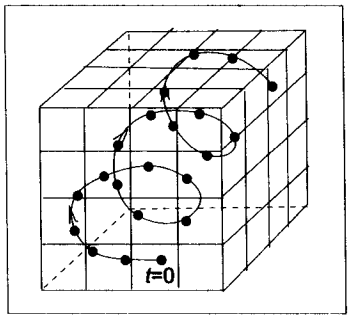


图16-25 穿越立方体的路径定义了过程噪声，通过在路径上移动可以实现相应模式的动画

对于一些过程技术，时间依赖性是一个更必要的部分。最简单的细胞自动机（cellular automata）是在二维矩形栅格上操作的，栅格中的每个位置（细胞）存储一个二进制值。为了建立一个时间变化模式，要反复地应用用户提供的规则来修改这些值。规则一般包括当前值和细胞周围邻居值的一些条件集合。例如：1970年由英国数学家John Conway发明的很受欢迎的二维生命游戏（Game of Life）的细胞分裂规则是：

(1) 一个死细胞（即在指定位置的二进制值是0）如果恰好有三个活的相邻细胞，则变成活细胞（即它的值被设为1）。

(2) 一个有两个或三个活细胞邻居的活细胞仍然是活的。

(3) 除了以上情况，一个细胞死亡或保持死亡状态。

这些规则一旦应用在所有栅格位置上，就将建立一个新的模式，并且将开始一个新的进化周期。图16-26中是三个简单瞬间图，它们表现了在不同时刻活细胞的分布情况。三维栅格上有更复杂的同步操作，可能有浮点值操作，可以用来建立云和其他气体现象或生物系统的动态模型。 371

这种方法本来就是为此发明的（注意术语）。仔细选择几条规则就会产生复杂性令人吃惊的模式。但是如何写出这些规则来产生期望的行为通常并不是显而易见的。过程技术有一个共同的问题是：关于如何创建新过程或调整已存在参数，基本上没有什么引导，有也是有限的。因此，为了充分发挥过程方法的潜力，通常需要利用试错法（“依靠经验”）进行大量的调节和学习。

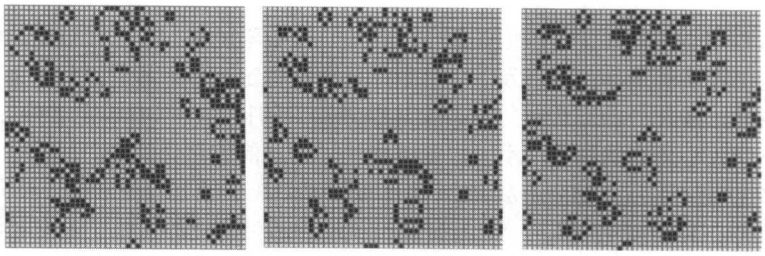


图16-26　生命游戏自动机进化的几个（非连续的）阶段。黑色的表示活细胞。稳定的物体、震荡器、旅游模式、和其他许多有趣的结构都可以用很简单的规则来产生。图片由Alan Hensel用程序生成

另一个有趣的方法称为L系统（L-systems，以其最初发明者Astrid Lindenmayer的名字命名）。这种技术原来也是用来描述生物学对象的。该方法基于文法或重写符号串的递归规则。有两种符号：终止符代表用来表示文法的元素。根据它们的含义，文法可以描绘出树木和灌木、建筑和整个城市，或者程序设计和自然语言的结构。在动画中，L系统普遍用来表现植物，相应的终止符指向一个几何建模系统：将一片叶子（或一根树枝）放在当前位置——这里使用符号@，并且只画一个圆圈，将当前位置向前移动几个单位（符号*f*），将当前方向沿着世界坐标系的*Z*轴转动60度（符号+），弹出（符号[）或推入（符号]）当前的位置/方向，等等。辅助的非终止符（用大写字母表示）只有语义上的意义而没有任何直接的含义。它们最终将用终止符来重写。从特定的非终止符（开始符号*S*）开始，不断并列地将文法规则应用到当前的串中，即，替换当前出现的所有非终止符获得新的串，直到得到一个只包含终止符的串，或者没有可能替换的为止。然后，使用这个建模指令串输出实际的几何图形。例如，下面一组规则（产生式）：

$$S \to A$$
$$A \to [+B]fA$$
$$A \to B$$
$$B \to fB$$
$$B \to f@$$

可能产生以下序列的重写步骤，如图16-27所示。

$$S \mapsto A \mapsto [+B]fA \mapsto [+fB]f[+B]fA \mapsto$$
372
$$[+ff@]f[+fB]fB \mapsto [+ff@]f[+ff@]ff@$$

正如上面所看到的，相同的非终止符通常有许多不同的产生式，所以相同的文法可以生成许多不同的对象。可以根据靠近正在被替换符号位置的符号（上下文敏感的）来选择使用哪条规则，或者按对每条规则指定的概率进行随机的选择（随机L系统）。更复杂的规则可以通过和环境的交互来建模，如：剪枝成特别的形状，或者使用关联到符号的参数来控制相应的几何命令。

L系统已经捕获了植物随时间改变的拓扑结构：重写过程中获得的每一个中间串都可以认为是植物“幼年”的样子（见图16-27）。对于更强烈的变化，不同的产生式可以在不同的时间发挥作用，这允许植物结构在生长过程中发生巨大的变化。例如：一棵小树会产生大量的新枝，而一棵老树只适度地长出新枝。

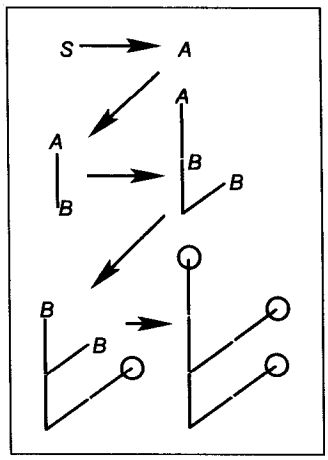


图16-27 使用一个简单L系统的连续推导步骤。大写字母表示非终止符，它表示相应的非终止符将被扩展的位置，不是实际输出的一部分

使用L系统已经可以生成非常真实的植物模型。但是，和大部分过程技术一样，我们需要有一定经验，有目的地使用现有的L系统，而写出新的文法实现一些期望的效果并不是一件容易的事情。

16.7 对象群

要实现多个对象的动画，当然可以简单将上述的标准技术应用在每个对象上。这种方法对数量适中的独立对象是很合理的，假设其中对象的期望运动已经预先知道了。但是，在许多情况下，动态环境中的某种协同动作是必须的。如果只涉及很少的对象，那么动画设计者可以根据某种高层的目标，利用基于人工智能（AI）的系统来自动地确定每个对象的当前任务，设计必要的运动，并执行这些设计。许多现代游戏使用这样的自主对象（autonomous object）来创建自动的怪物或玩家的伙伴。

有趣的是，当群中的对象数目从几个增长到几十，几百，甚至几千时，为了使群作为一个整体，表现出看起来象是目的驱动的协同运动，群中的个体成员只能拥有非常有限的“智能”。这表明群集（flocking）是紧急的行为。它是群成员只和最靠近它们的几个邻居进行有限交互作用产生的结果（Reynolds,1987）。任何观察过鸟群或鱼群迷人的同步运动的人都会熟悉群集。这种技术也可以用来控制在地面上移动的动物群，甚至是人群。 373

在任何指定时刻，群中成员的运动当应用于群集时通常被称为boid，它是对一些通常有矛盾的趋势平衡的结果。每个boid 都有自己的速度向量（见图16-28）。首先，有外部物理作用力F作用于boid，如重力或风。作用力下新的速度可以直接由牛顿定律来计算：

$$v_{new}^{physics} = v_{old} + F\Delta t / m$$

其次，一个boid应该反作用于全局环境和其他群成员的行为。碰撞规避（collision avoidance）是这种相互作用的主要结果之一。对于群集，至关重要的是每个群成员的视野非常有限，因此它只能知道它当前位置的邻域发生的事情。为了避开环境中的对象，最简单但不完备的策略是在每个对象周围建立一个有限范围的排斥力场。这将产生第二个期望的速度向量$v_{new}^{col_avoid}$，也由牛顿定律给出。通过同步地应用不同的操控行为产生一些额外的期望速度向量v_{new}^{steer}，可以建立其他群成员相互作用的模型。最普通的模型是：远离邻居以避免拥挤，操控向聚集同伴方向运动以确保群

集的凝聚性，以及调整boid的速度，使之与邻居的平均前进速度保持一致。最后，通常应用某些
374 特定的期望速度向量v_{new}^{goal}以达到需要的全局目标。这可以是空间中沿着某条路径的向量。它们跟随一些专门指定的群集首领，或是只是表示群体成员紧急迁移欲望。

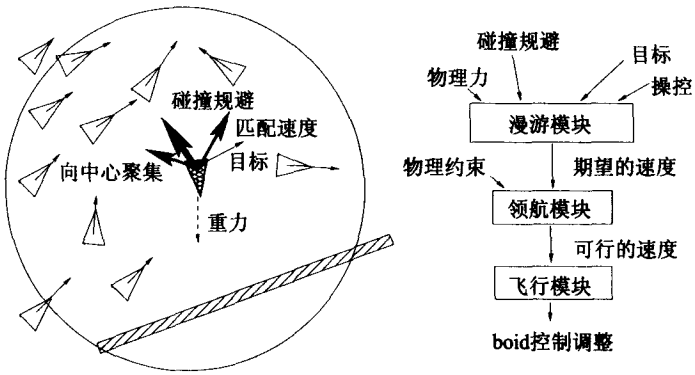


图16-28　（左）群集的个体成员（boid）会受到重要性不同的驱动力作用（用线条宽度表示），它们必须要合成一个速度向量。一个boid只知道到它周围的有限邻域（圆圈）。（右）boid的控制一般有三个单独的模块

一旦确定了所有的v_{new}，最终期望的向量是根据其优先级协商出的。碰撞规避和速度匹配一般有更高的优先级。使用加速度分配策略来代替简单的平均期望速度向量，因为后者会导致驱动力丢失和不自然的“无处移动”（moving nowhere）行为。可以将总量固定的加速度用于boid，加速度的每个小部分按照优先级顺序分给各个驱动力。如果全部可用的加速度用完，那么低优先级的驱动力将对运动产生比原来更小的影响或被完全忽略。希望是一旦当前最重要的任务（在大部分情况下是碰撞规避）完成了，就应该尽快关注其他任务。注意要满足真实对象的物理限制也很重要。例如：把过高的加速度或速度调整到某个真实值。根据群集成员的内部复杂性，动画的最后阶段可能是将协商得到的速度向量转化为一个特殊的参数集（如鸟的翅膀的位置，空中飞机模型的方向，腿的骨骼结构等）。这些参数用来控制一个boid的运动。图16-28（右）显示了一个实现了群集的系统。

一个更简单，但是很有用的群控制方法是*粒子系统*（particle systems）（Reeves,1983）。系统中的粒子数目一般要比群体中boid数目大得多。粒子数目可以是几万到几十万，甚至更多。而且，在动画过程中，随着在每个阶段新粒子的出生和一些旧粒子的消亡，粒子的精确数目是可以变动的。粒子彼此之间一般完全独立，忽视它的邻居，并且只通过受到的外力以及和物体的碰撞来与环境发生作用，而没有像群集那样使用碰撞规避。在动画的每个阶段，系统首先按照初始参数创建新粒子，终止旧粒子，然后根据牛顿定律计算必要的作用力，并更新剩下粒子的速度和位置。

粒子系统的所有参数（粒子的数目、粒子的生命周期、初始速度和粒子的位置等）通常直接在动画设计者的控制之下。粒子系统的主要应用包括模拟焰火、爆炸、喷射的液体、烟和火，或者其他模糊物体和没有清晰边界的现象。为了获得真实的效果，在所有的参数中引入一些随机性
375 是很重要的。例如：在每个阶段有随机数量的粒子出生（或消亡），其产生的速度是根据某些分布决定的。除了设置合适的初始参数外，控制粒子系统运动的一般做法是在空间中创建一个特殊的作用力模式——例如：一旦粒子到达某些特殊的位置，就将它吹向一个新的方向或添加一个吸引中心（图16-29）。我们应该记住，尽管粒子系统有很多优点：易于实现、易于控制，但是对于相同的现象，粒子系统一般不能提供像基于物理的仿真所能达到的真实程度。

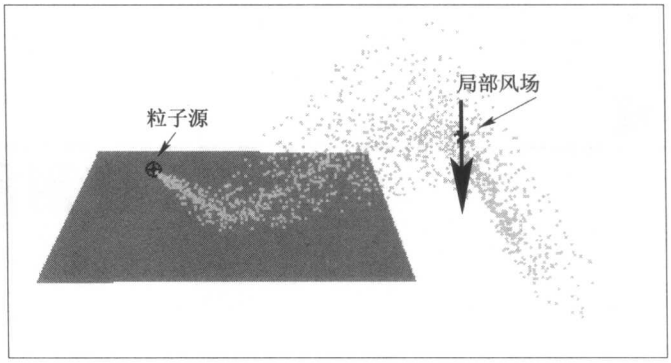


图16-29 粒子从一个方向发射出来后，与一个物体发生碰撞，接着，一旦它们清除了障碍就被一个局部的风场吹到下面去

文献注记

这一章，我们集中介绍了三维动画中使用的技术。当然也有很多算法用于实现二维动画和由计算机图形学绘制系统产生的图像的后处理。这些技术包括整理扫描输入的画家绘画、特征抽取、二维中间动画的自动生成、配色、图像变形、增强和合成，以及其他许多技术。

计算机动画最重大的发展之一是高级动画系统不断增强的处理能力和实用性。虽然它们的具体特征、内部结构、用户界面的细节和价格可能不同，但是大部分这些系统都不仅广泛支持动画的实现，而且也支持建模和绘制，这可以将它们转化为完整的产品平台。使用这些系统来产生静态图片也很普遍。例如，在这一节中的许多图像是利用Maya软件产生的，该软件由Alias公司慷慨捐赠。 376

大规模的动画作品是一个非常复杂的过程。它一般需要跨部门甚至是跨公司的许多不同背景的人共同努力。为了建立更好的协同，需要建立特定的产品流水线，从绘制故事和角色的草图开始，记录必要的声音，建模，并且为动画角色提供服饰。一旦开始实际的动画制作，通常要回头修改原来的设计、模型和服饰，以解决发现的所有运动问题和外观问题。接着设置必要的光照和材质性质，然后才可以开始绘制。在最复杂的工程中，大量的后处理和场景复合将不同来源的图像集成在一起，完成最后的作品。

在结束本章前，我们提醒读者，在计算机动画领域里，对于好的故事、富于表现力的角色和其他艺术因素来说，任何技术上的精巧完备都是次要的。因为，大部分因素都很难或不可能简单地量化。可以肯定地说白雪公主和七个小矮人可以与绿色的怪物和驴共同出现在屏幕上，因为大部分观众对角色和故事更感兴趣，而不是它们是否由计算机（或用何种精确的方式）帮助创建的。 377

第17章

使用图形硬件

Peter Willemsen

本书大部分的重点都放在计算机图形学的基础理论,而不是实现细节上。本章采用了一个略微不同的讲解路线，把使用图形硬件的细节和对该硬件编程相关的实际问题混合在一起进行介绍。

尽管如此，本章并不会讲解OpenGL™、其他的图形应用程序接口（API）甚至晦涩难懂的图形硬件编程的细节。本章的目的在于介绍一些基本概念和思想方法，这些是使用图形硬件编写程序的时候必备的。

17.1 什么是图形硬件

图形硬件描述的是进行快速绘制所必需的硬件单元，它使用基于光栅的专用硬件体系结构，能够快速将三维物体绘制成计算机屏幕显示的像素。使用这个术语是为了引起对执行这些计算所必需的物理单元的感性认识。换句话说，我们正在探讨的是当前许多显卡上的芯片、晶体管、总线和处理器。正如在本章即将看到的，当前的图形硬件非常善于处理三维物体的描述并将其转化为显示器上已着色的像素。

对于图形硬件来说有一件事是肯定的：随着新的扩展和特征的不断加入，它发展得非常快！
379 对于这种快速的发展步伐，可以解释为视频游戏产业的发展及其经济的驱动。本质上这就意味着新出现的图形卡提供了更好的性能和处理能力。这样，图形硬件就被应用到需要很多三维图形处理的任务中。例如，研究者正在图形硬件上进行计算实现光线跟踪（Purcell，Buck，Mark，& Hanrahan, 2002），甚至解决了模拟湍流的Navier-Stokes方程（Harris，2004）。

大部分图形硬件设计为要完成一组固定的操作。这些操作被组织成流水线，流水线把顶点和像素依次放入各个阶段。流水线所固有的功能确保了基本的着色。光照和纹理能很快执行——我们常称之为实时图形学（real-time graphics）。

实时图形学 实时图形学，通常意味着图形相关的计算执行得足够快，以至于能立即显示出结果。能以60 Hz的速度执行操作就认为是实时的。一旦显示刷新率（帧率）降低到15 Hz以下，这一速率就会认为是交互性的，而不是实时的，但是这种差别并不重要。因为计算速度很快，如果有更多的时间，用来绘制图形的公式通常可以近似地表示要描述的物体。

图17-1阐明了一个实时图形学流水线。下面是有关流水线需要注意的一些地方：

- 用户程序（或者说应用程序）把数据以图元的形式提供给图形硬件，例如用来描述三维几何模型的点、线或多边形。同时也提供用于表面纹理映射的图像或者位图。
- 以每个顶点为基础对几何图元进行处理，并从三维坐标变换为二维屏幕坐标。
- 屏幕对象先是被传送到像素处理器进行光栅化，然后在被输出到帧缓冲器之前对每个像

素进行着色，最后输出到显示器。

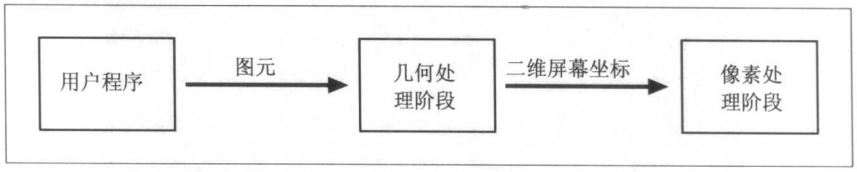


图17-1 基本的图形硬件流水线由几个阶段组成，三维数据转换成二维屏幕对象，为像素处理阶段的光栅化和着色作准备

17.2 从硬件角度描述几何模型

作为一个图形程序员，需要关心的是与三维物体相关的数据是如何被送到图形硬件的缓存中的。不幸的是（也可能是幸运的），程序员不能完全地控制这一过程。有许多方式可以把数据输送到图形硬件上，每一种方法都有自己的优点，在这部分中将讨论这些内容。可以使用API为显卡编写程序，但是任何一种API都不能提供相同的方式把数据加载到图形硬件的存储器中。在下面的例子中是用伪代码表示的，它基本上是基于OpenGL™的C语言函数的，但从语义上讲，这些例子也可以适用于其他的图形API。 380

大部分的图形硬件都使用一组具体的几何图元。这些图元类型减少了图元的复杂性，提高了图形硬件上的处理速度。图元越简单处理速度就越快。需要注意的是，图元类型应该是通用的，以模拟从简单到复杂的几何模型。在常见的图形硬件上，图元类型被限制为下面的一种或几种：

- **点**：单个的顶点通常用来表示点或粒子系统。
- **线**：成对的顶点通常用来表示线、轮廓或者高亮边缘。
- **多边形**：（例如三角形、三角形条带、索引三角形、索引三角形条带、四边形、一般的凸多边形等）用于描述三角形网格、几何模型表面和其他实体，比如球、圆锥、立方体或圆柱体。

图元 这三种图元（点、线和多边形）是唯一能使用的图元！甚至当创建基于样条的表面时，比如NURBs（非均匀有理B样条），也会通过图形硬件把表面转换成三角形图元。

点绘制 最初看来，点图元和线图元在使用中好像会受到限制，但是研究者已经能够使用点来绘制非常复杂的几何模型（Rusinkiewicz & Levoy，2000；Dachsbacher，Vogelgsang & Stamminger，2003）。

这三种图元作为构造要定义的绝大部分几何模型的基本构建模块。（图17-2显示了一个三角形网格的例子），使用这些图元，就能够利用一种图形API构建几何模型的描述，并把几何模型发送给图形硬件进行绘制。例如，为了把直线的描述发送给图形硬件，可以使用下面的代码： 381

```
beginLine();
  vertex( x1, y1, z1 );
  vertex( x2, y2, z2 );
endLine();
```

在这个例子中，完成了两个任务。第一，声明一种图元类型，并调用beginLine()函数激活。然后调用endLine()函数使线形图元被禁用。第二，通过调用vertex函数，把这两个函数之间声明的所有顶点都直接复制到图形卡中。

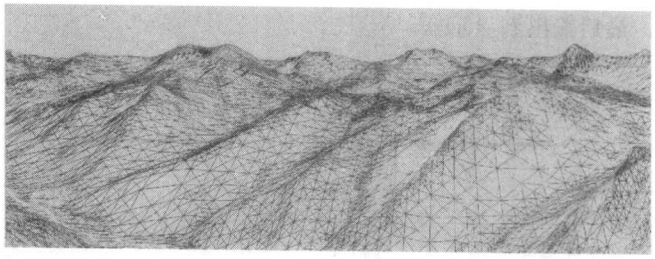


图17-2　几何模型的组织将影响应用程序性能。Little Cottonwood Canyon地形数据集的线框描述显示了数万个三角形，它们组成了一个三角形网格，以实时的速度运行（图像用VTerrain Project地形系统绘制，经Ben Discoe授权使用）

另外一个例子是创建一组组合成条带的三角形（见图17-3），可以使用下面的代码：

```
beginTriangleStrip();
  vertex( x0, y0, z0 );
  vertex( x1, y1, z1 );
  vertex( x2, y2, z2 );
  vertex( x3, y3, z3 );
  vertex( x4, y4, z4 );
endTriangleStrip();
```

在这个例子中，图元类型TriangleStrip被激活，定义三角形条带的顶点集被复制到图形卡的存储器去处理。注意在描述几何模型时指令的顺序是重要的。在三角形条带的例子中，相邻三角形之间的连接性含在顶点顺序中。顶点(v_0,v_1,v_2)构成了三角形t_0，顶点(v_1,v_3,v_2)构成了三角形t_1，顶点(v_2,v_3,v_4)构成了三角形t_2。

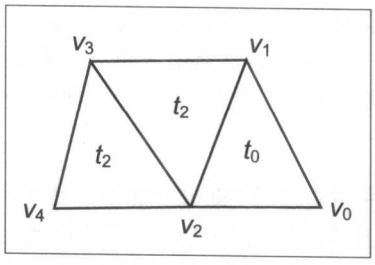


图17-3　一个由5个顶点定义的3个三角形组成的三角形条带

从这些简单例子中学习到的关键之处是：为了在图形硬件上进行绘制而定义的几何模型是一种图元类型和一组顶点定义的。前面的例子都是比较简单的，它们直接把顶点集送到图形硬件上。然而实际中，需要明确地确定怎样将数据送到图形硬件上，后面马上对这些问题进行讨论。

当几何模型传送到图形硬件时，可以为每个顶点指定额外的数据。这些额外的数据对于定义状态属性是很有用的，它能表示顶点的颜色、顶点上的法线方向、顶点上的纹理坐标或者其他每个顶点的数据。比如，为了设置三角形条带每个顶点的颜色和法线状态参数，可以使用下面的代码：

382

```
beginTriangleStrip();
  color( r0, g0, b0 ); normal( n0x, n0y, n0z );
  vertex( x0, y0, z0 );
  color( r1, g1, b1 ); normal( n1x, n1y, n1z );
  vertex( x1, y1, z1 );
  color( r2, g2, b2 ); normal( n2x, n2y, n2z );
  vertex( x2, y2, z2 );
  color( r3, g3, b3 ); normal( n3x, n3y, n3z );
  vertex( x3, y3, z3 );
```

```
    color( r4, g4, b4 ); normal( n4x, n4y, n4z );
    vertex( x4, y4, z4 );
endTriangleStrip();
```

其中每个顶点的颜色和法线方向都在定义顶点之前被指定。在这个例子中每个顶点有唯一的颜色和法线方向。函数color使用一个RGB三元组设置活动的颜色状态。每个顶点的法线方向状态通过函数normal设置。函数color和函数normal都影响图形硬件当前的绘制状态。任何在设置状态属性后定义的顶点都将绑定到这些状态属性上。

现在到了提及图形硬件的时候了，图形硬件维护了一个相当精细的状态参数集合，这个集合决定了顶点和其他组元如何绘制。一些状态被绑定到顶点上,比如颜色、法线方向和纹理坐标，而另一个状态可能影响像素级的绘制。在任何特定的时刻，图形状态都描述了一个巨大的内部硬件参数集。当编写三维应用程序的时候，考虑图形硬件的这个特征显得尤为重要。正如猜想的那样，频繁地对图形状态进行修改至少会在某种程度上影响性能。但是，除了使图形状态的改变最小，编程时需要仔细考虑的地方还有很多。应该尽量使状态的改变最小化，但是不可能把所有的几何模型都组合在一起以完全减少状态在上下文中的切换。利用一种数据结构能帮助最小化状态的改变，尤其在静态场景中，这就是场景图数据结构。在绘制任何几何模型之前，场景图都能重新组织几何模型及其相关的图形状态，尽量使状态的改变最小化。场景图在第13章中描述。

```
color( r, g, b );
normal( nx, ny, nz );
beginTriangleStrip();
  vertex( x0, y0, z0 );
  vertex( x1, y1, z1 );
  vertex( x2, y2, z2 );
  vertex( x3, y3, z3 );
  vertex( x4, y4, z4 );
endTriangleStrip();
```

383

在TriangleStrip中的所有顶点都具有相同的颜色和法线方向，所以这些状态参数可以在定义顶点之前设置。这把上面的函数调用和内部图形状态的改变降至最低。

有许多方面可以影响图形程序的性能，但其中最主要的是几何模型怎样组织以及它是否存储在图形卡的高速缓冲存储器中。在目前给出的代码中，几何模型被送到图形硬件中，这常常称为立即模式（immediate mode）绘制。在定义顶点之后，它们直接被送到图形硬件中。立即模式绘制的主要缺点是应用程序的每次迭代都要把几何模型送到图形硬件里。如果几何模型是静态的（即它不会发生改变），这时每次重画一帧时都不需要重发数据。在这种或其他一些情况下，更合适的做法是把几何模型存储到图形卡的内存中。

在计算机中的图形硬件通过数据总线（例如PCI、AGP或PCI-Express总线）连接到系统的其他部分。当向图形硬件发送数据的时候，它是由机器中的CPU通过总线发送的，最后存储在图形硬件的内存里。如果用一些大型的三角形网格表示复杂的几何形体，通过总线传送数据可能会对程序性能产生极大影响。就像前面的例子那样，几何模型在以立即模式绘制的时候体现得尤为明显。

组织几何模型有许多不同的方式；有一些有助于降低通过图形总线发送几何模型时所需的总的带宽。一些可能的组织方法包括：

- **三角形**：三角形用三个顶点定义。通过这种方式创建的三角形网格要求单独定义网格中的每个三角形，这其中会有许多的顶点被重复定义。对于一个包含 m 个三角形的三角形网格来说，将有 $3m$ 个顶点送到图形硬件中。
- **三角形条带**：三角形被组织成条带的形式。前三个顶点定义了条带中的第一个三角形，每添加一个顶点都会增加一个三角形。假如把 m 个三角形组织成一个三角形条带来构建

三角形网格，首先应该给图形硬件发送三个顶点，然后每添加一个顶点就会在条带中多
384 出一个三角形，条带中顶点的总数是 $m+2$。

- **索引三角形**：三角形顶点被排成一个顶点数组，另一个单独的数组用到顶点数组中的索引定义三角形。只要用很少的函数调用就能把顶点数组发送到图形卡中。
- **索引三角形条带**：和索引三角形一样，三角形顶点被存储在一个顶点数组里。不过，三角形被组织成条带，通过索引数组定义条带布局。这是定义三角形网格最紧凑的组织结构，因为它把三角形条带的优点和顶点数组的紧凑性结合在了一起。

不同的组织结构，通过索引三角形或索引三角形条带使用顶点数组，都是提高应用性能的好的选择。对组织紧密的封装也意味着需要更少的函数调用。一旦顶点和索引都存储到一个数组里，只需要几个函数调用就能把数据传送到图形硬件中，而在前面例子的伪代码中，每个顶点都需要一个函数调用。

现在你可能会疑惑在使用顶点数组的时候，像颜色、法线或者纹理坐标等的图形状态是怎样定义的。在本章前面立即模式绘制的例子中，用交叉存储相关顶点的图形状态显然是基于函数调用的次序。当使用顶点数组的时候，图形状态可以交叉存储在顶点数组里，或者定义在传送到图形硬件中的单独的数组中。

尽管几何模型在发送到图形硬件中时能被有效地组织，但是如果能在应用程序运行期间把几何模型存储到图形硬件的内存中，就可以获得更好的性能。对于当前的图形硬件来说有一些遗憾，那就是许多描述图形硬件内存和高速缓冲存储器结构的规范说明通常不被公开。然而幸运的是，有许多使用图形API的方式，允许程序员把几何模型放到图形硬件内存中以使应用程序运行得更快。

在图形硬件高速缓存中存储几何模型和图形状态通常使用的两种方法是创建*显示列表*和*顶点缓冲对象*。

显示列表编辑了一个简洁的列表表示几何模型及其相关状态，并把列表存储在图形硬件的内存里。显示列表的好处是它们是通用的，善于把静态的几何表示及其相关的图形状态存储到硬件
385 中。当不断地改变几何模型和图形状态时显示列表就不能很好地工作，因为每次更改显示列表，都要对显示列表重新编辑并再次存储到图形硬件内存中。

```
displayID = createDisplayList();
color( r, g, b );
normal( nx, ny, nz );
beginTriangleStrip();
  vertex( x0, y0, z0 );
  vertex( x1, y1, z1 );
  ...
  vertex( xN, yN, zN );
endTriangleStrip();
endDisplayList();
```

在上面的例子中，创建了一个显示列表，它包含了一个三角形条带的定义，以及相关的颜色和法线信息。在函数调用createDisplayList和endDisplayList之间的命令给出了定义一个显示列表需要的要素。显示列表通常在应用程序的初始化阶段创建。显示列表创建之后，存储在图形硬件的内存中，以后可以通过分配给列表的标识号访问。

```
// draw the display list created earlier
drawDisplayList(displayID);
```

当要绘制显示列表内容的时候，使用一个专门的函数调用通知图形硬件访问显示列表标识符所指的内存并显示列表内容。

应用程序运行期间，令一种把几何模型存储到图形硬件上的方式是使用顶点缓冲对象（vertex

buffer object, VBO)。顶点缓冲对象是图形硬件的高性能内存中专用于缓冲器，用于存储顶点数组及其相关的图形状态。同时顶点缓冲对象还可以提供从应用程序到图形硬件内存的映射，它允许快速地访问和更新顶点缓冲对象的内容。

顶点缓冲对象的主要优点是它能提供到图形硬件内存的映射。与使用立即模式绘制或显示列表相比，使用顶点缓冲对象可以在对应用程序的性能影响最低的情况下修改几何模型。倘若几何模型的某些部分在应用程序的迭代中发生改变或者组织几何模型的索引经常发生改变，这时顶点缓冲对象的映射就更有用了。

创建顶点缓冲对象的方式和建立索引三角形、索引三角形条带的方式非常相似。首先，在图形卡上创建一个缓冲对象，为包含三角形网格顶点的顶点数组分配空间。然后，顶点数组和索引数组被复制到图形硬件上。当要绘制几何模型的时候，使用顶点缓冲对象标识符通知图形硬件画出几何模型。如果在应用程序中已经使用了顶点数组，那么为了使用顶点缓冲对象而对代码进行的修改应该尽可能保持最小限度的改变。

386

最佳的组织：许多研究正努力寻找优化三角形网格的方式，以最大程度上提高图形硬件的性能。如果想进一步研究三角形网格的组织对性能的影响，建议先看一下SIGGRAPH 1999上关于网格局部性优化的文章（Hoppe，1999）。

17.3 几何模型转化为像素

当几何模型被放到图形硬件内存中后，在图形处理阶段，每个顶点都必须计算光照并变换为屏幕坐标。在图17-1所阐述的固定功能图形流水线中，顶点从模型坐标系变换到屏幕参考坐标系。这个过程和相关的矩阵在第7章和第12章中进行了讲解。变换需要的模型观察矩阵和投影矩阵使用图形API提供的函数定义。

在每一个顶点上都要进行光照计算。根据全局明暗处理参数，三角形表面可以具有平面明暗处理的效果或者漫反射明暗处理（Gouraud 明暗处理）的效果，后者是通过在三角形表面上对三角形顶点的颜色进行线性插值实现的，它会产生更加平滑的外观。每个顶点上的颜色是根据指定的材质性质、场景中的灯光以及各种光照参数计算的。

固定功能的图形流水线中的光照模型有利于顶点的快速光照，它在提高速度和精确光照之间作了一个折衷。结果是，固定功能结构不支持Phong明暗处理的表面。

特别是图形硬件中建立的漫反射明暗处理算法通常不能计算出合适的光照，因为光照仅仅在每个顶点上计算。例如，当到光源的距离很近的时候（与进行明暗处理的面片的大小相比），面片的光照是不正确的。图17-4说明了这种情况。尽管三角形离光源很近，但是它的中心却并不是很亮，因为远离光源的顶点光照用于对面片的明暗处理插值。

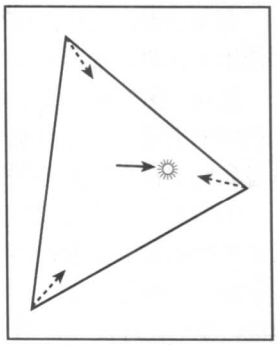


图17-4 到光源的距离相对于三角形的大小来说很小

387

利用固定功能的流水线，这个问题只能通过增加几何模型网格密度来补救。这个方法可以，但是它在实时图形中很少使用，因为一旦为了更精确的光照而增加几何模型，就会降低绘制的速度。

然而，对于当前的硬件来说，在不增加物体几何复杂度的情况下对光照进行更好的近似的问题已经得到了解决。具体地说就是用自己的程序替代嵌入在图形硬件内的固定功能的例行程序。这些小程序运行在图形硬件上并执行图形流水线中几何处理和像素处理阶段的一部分功能。

17.3.1 设计流水线

最近图形消费硬件组织的改变已经对游戏开发者、图形研究者及许多其他人产生了很大的影响。你很有可能已经听说过GPU程序设计、图形硬件程序设计甚至是着色器程序设计。这些术语和变化是因为消费硬件的变化而产生的，现在可以对图形硬件绘制流水线的方式进行编程。

具体地说，这种改变已经明显地呈现了图形硬件流水线的两个具体的特征。现在程序设计者已经有能力通过编写顶点着色器（vertex shader）和片段着色器（fragment shader）（有时也称为顶点程序（vertex program）和片段程序（fragment program））修改硬件处理顶点和像素着色的方式。顶点着色器是这样的程序：它通常在几何处理阶段执行顶点和法线的变换、纹理坐标生成和各顶点的光照计算。而片段着色器是这样程序：在图形流水线的像素处理阶段执行计算，并最终确定如何对像素着色，如何应用纹理以及是否绘制像素。这些小的着色程序由应用程序发送到图
388 形硬件中（见图17-5），但它们在图形硬件中执行。这种编程能力意味着需要有一台多处理器的机器。这样就可以很好地考虑一下图形硬件，因为它意味着在某些应用程序中能够使用图形硬件处理器减轻CPU的负荷。图形硬件处理器通常称为GPU。GPU表示图形处理单元（Graphics Processing Unit），并且强调现在的图形硬件单元包含一个专门用于图形相关计算的独立处理器。

定义 片段是一个术语，它描述了图形硬件处理之前与像素相关的信息。这个定义包括了许多数据，它们可能要用于计算像素颜色，例如像素的场景深度、纹理坐标或模板信息。

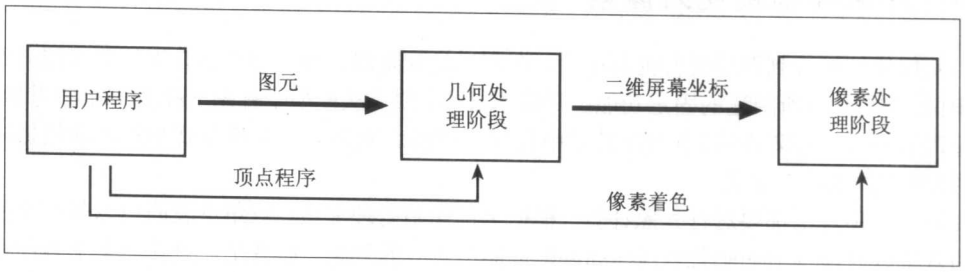


图17-5 可编程图形硬件流水线。用户程序给硬件提供图元、顶点程序和片段程序

有趣的是，现代的GPU要比现代的CPU包含更多的晶体管。在这一时期中，GPU将大部分晶体管用于计算，而将一小部分用于存储或高速缓存管理操作。

然而，图形硬件并不能总要按照这种模式发展。因为计算正向三维图形靠拢，这就意味着可以在GPU上进行与计算机图形无关的计算。但是在GPU上编程的方式与在通用的CPU上不同，在考虑如何解决问题和为图形硬件编程序时需要对以前的方式稍微做些修改。

GPU是一种流处理器，它在三维向量操作中优势明显，比如向量乘、向量加、点积以及其他表面光照和纹理映射所需要的基本操作。作为流处理器，顶点和片段处理单元都有能力同时处理多个图元。在这一点上，GPU是一个SIMD（Single Instruction, Multiple Data，单指令多数据）处理器，并且在片段处理器硬件特定的实现中，一次最多能处理16个像素。当为这些处理单元编写程序时，可以认为所有的计算是并行在数据上执行的，至少从概念上可以这样认为。换句话说，顶点着色程序将针对所有顶点同时运行。接下来的片段着色程序将在所有的片段上同时执行。值得注意的是，当顶点和片段计算同时发生时，流水线单元的执行仍然以同样的次序发生。

顶点和片段着色器的工作方式很简单。写一个顶点着色程序和一个片段着色程序，并发送到图形硬件中。这些程序可以用于具体的几何模型，当处理几何模型时，顶点着色器用于对顶点进行变换和光照计算，而片段着色器则在像素级上对几何模型执行最后的明暗处理。正如能把不同的图像映射到不同的几何模型面片上一样，也可以在程序中为不同的物体编写不同的着色器程

序。着色器程序是图形状态的一部分，所以必须要关注，根据当前正绘制的几何模型，着色器程序是如何进行来回交换的。 389

相关历史 流水线程序设计并不完全是最近才开始的。PixelFlow体系结构和来自于UNC的着色语言（Molnar, Eyles, & Poulton, 1992; Lastra, Molnar, Olano, & Wang, 1995; Olano & Lastra, 1998）是最早为提高编程设计灵活性而设计的图形硬件体系结构之一。为了提供自定义的着色技术，其他的研究包括了着色树（Cook, 1984）、RenderMan（Pixar, 2000）、使用OpenGL™的加速多通道绘制（Peercy, Olano, Airey, &Ungar, 2000）以及其他实时着色语言（Proudfoot, Mark, Tzvetkov, & Hanrahan, 2001; McCool, Toit, Popa, Chan, & Moule, 2004）。

不过细节变得有些复杂。顶点着色器通常会执行两个基本的动作：设置顶点颜色和把顶点变换为屏幕坐标，这通过把顶点分别与模型观察矩阵和投影矩阵相乘得到。透视分割和裁剪的步骤不在顶点程序中进行。顶点着色器也常常用来为片段着色器作准备。尤其是当拥有一些顶点属性时，比如纹理坐标或其他依赖应用的数据，顶点着色器计算或修改这些属性，然后发送到片段处理阶段，以备片段着色器使用。开始可能会觉得很奇怪，不过顶点着色器能够用来控制顶点的位置。这对于完全在GPU上生成仿真的海浪运动常常是很有用的。

在片段着色器中，要求程序输出片段颜色。这可能涉及查询纹理值和以某些方式把纹理值与其他数值进行合并，这些数值是通过在每一个像素上执行光照计算而获得。或者它也可能涉及删除某个片段，阻滞对它进行完全地绘制。因为片段着色器中的操作是在片段级上进行的，所以可编程图形硬件真正的能力就是在片段着色器中。固定功能流水线和可编程流水线的关键区别之一就在于这个附加的处理能力。在固定功能流水线中，片段处理使用三角形顶点间的光照插值计算片段颜色。而用可编程的流水线，每个片段的颜色可以独立计算。例如，在图17-4所示的情况下，三角形面片的Gouraud明暗处理不能给出一个合理的结果，因为仅仅在顶点上进行光照计算：与三角形中心相比，它们离光源更远。在片段着色器中，光照方程在每一个片段上计算，而不是在每一个顶点上，这会使面片的绘制更加准确。

17.3.2 基本执行模型

当编写顶点和片段着色器时，关于顶点和片段程序如何执行以及如何访问GPU上的数据而言，还有一些重要的东西要去理解。因为这些程序完全在GPU上运行，所以需要解决的第一个问题就是着色器将要使用哪些数据以及着色器怎样获得数据。着色器程序使用的数据类型有几个相关特征。下面的术语用来描述这些数据的概念上的特征，它们主要来自OpenGL™ 着色语言框架。不同的着色语言框架中的概念是相同的。在自己写的着色器中，变量使用下面的术语之一来描述其特性： 390

- **属性**：属性变量表示每个顶点上频繁变化的数据，通常和与每个顶点相关的图形状态的变化联系在一起。例如，法线向量或纹理坐标被认为是属性数据，因为它们是和每个顶点相关的图形状态的一部分。
- **一致性变量**：一致性变量表示在着色器程序执行期间不能改变的数据。尽管如此，在一个着色器的各个执行段之间一致性变量可以被应用程序修改。这也为应用程序提供了一种把数据传送到着色器的方式。一致性数据常常表示和应用程序相关的图形状态。例如，模型观察矩阵和投影矩阵可以通过一致性变量访问。应用程序中光源的信息也能够通过一致性变量获得。在这些例子中，当着色器正在执行的时候数据不能被更改，但能在应用程序的下一个迭代之前更改（例如，光线可以移动）。
- **可变的**：可变数据用于在顶点着色器和片段着色器之间传递数据。数据被认为是可变的

原因是数据由顶点着色器在顶点级上写入，但由片段着色器读取，作为在面片上进行插值时图元相邻顶点的值。

用这三个特性定义的变量可以是内建变量或者是用户自定义变量。除了访问内建图形状态，属性变量和一致性变量也是把用户自定义数据传送到顶点和片段程序的方式之一。可变数据是把数据从顶点着色器传送到片段着色器的唯一方式。图17-6利用着色器的输入和输出阐明了顶点和片段处理器的基本执行过程。

另一种把数据传送到顶点和片段着色器的方法是使用纹理映射作为数据的来源和接收。如果只把纹理图作为应用到几何模型表面的图像，你可能会感到很惊奇。纹理图之所以重要是因为它们允许访问图形硬件的内存。当编写在CPU上运行的程序时，可以控制应用程序需要的内存并在必要时直接访问它。在图形硬件上，内存不能以同样的方式访问。实际上，不能直接分配和收回
391 通用内存块，所以这个特殊的方面通常需要我们稍微改变一下思考问题的方式。

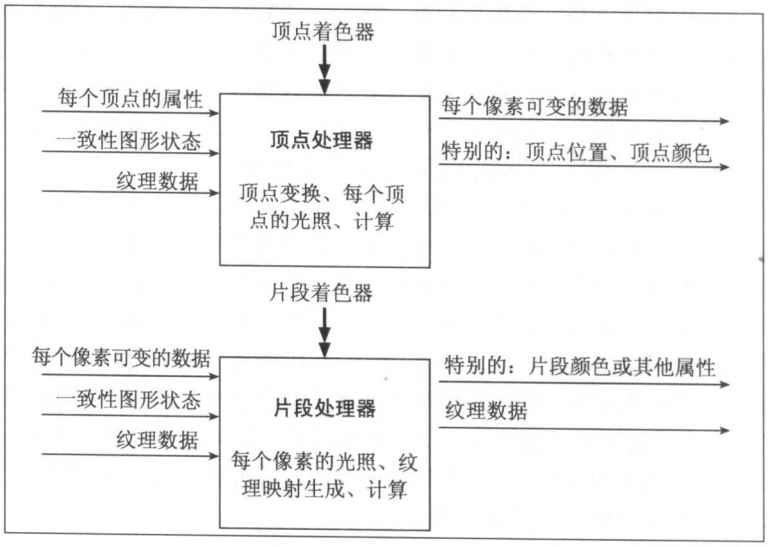


图17-6　着色器程序的执行流程。输入，如每个顶点的属性、与图形状态相关的一致性变量、可变数据和纹理图等提供给着色器处理器的顶点和片段程序中。着色器输出的变量专门用于后面部分图形流水线中

尽管如此，可以使用图形API在图形硬件上创建、删除和控制纹理图。换句话说，对于着色器使用的一般性数据，可以创建包含那些数据的纹理图，然后使用纹理访问函数访问纹理图中的数据。从技术上讲，顶点和片段着色器都能访问纹理。然而实践中，当前并不是所有的图形卡都支持从顶点着色器访问纹理。一个利用纹理图作为数据源的例子就是为凹凸映射。凹凸映射使用了一个法线图，法线图定义了法线向量三角形表面上是如何改变的。凹凸映射片段着色器能在法线图的“纹理数据”中访问法线向量，并在特殊片段上把它用在着色计算中。

必须要注意放入到纹理图中的数据类型。并不是所有数值数据类型都能得到很好的支持，直到最近才出现支持16位浮点纹理的图形硬件。而且，在GPU上执行的计算不能处理双精度的数。如果应用程序对数字精度要求很高，就需要仔细思考这些问题以决定使用图形硬件计算是否有用。

注解　在本章中使用的着色器语言示例都是用GLSL（OpenGL™ Shading Language）给出的。选择这种语言是因为它是由OpenGL™体系设计委员会（Architecture Review Board）开发的，而且当发布OpenGL™ 2.0版本时，它可能会成为OpenGL™的标准着色语言。在本书写作的过程中，利用更新的图形硬件驱动，GLSL可以在当前大部分图形卡上使用。

那么这些着色程序看起来什么样子呢？编写顶点和片段着色器的一种方式是使用汇编语言指令。例如，使用着色器汇编语言执行一个矩阵乘法的程序是：

```
DP4 p[0].x, M[0], v[0];
DP4 p[0].y, M[1], v[0];
DP4 p[0].z, M[2], v[0];
DP4 p[0].w, M[3], v[0];
```

392

在这个例子中，DP4指令是一个四元组点乘函数。它把点积存储在第一个寄存器中，在后两个寄存器之间执行点乘。在着色器程序设计中，寄存器存储与齐次坐标对应的x、y、z、w项或者RGBA元组里的r、g、b、a项。所以在这个例子中一个简单的矩阵乘法

$$\boldsymbol{p} = \boldsymbol{M}\boldsymbol{v}$$

用四个DP4指令来计算，每条指令计算最后结果中的一个元素。

幸运的是，我们不必一定用汇编语言编程序,高级语言也能够编写顶点和片段着色器。对于图形硬件的可编程特征，NVIDIA的Cg、OpenGL™着色语言（GLSL）和微软的高级着色语言（HLSL）都提供了类似的接口，允许用程序访问图形硬件。使用GLSL的表示法，上面的矩阵乘法可以写成

```
p = M * v;
```

其中$\boldsymbol{p}$和$\boldsymbol{v}$是顶点数据类型，$\boldsymbol{M}$是矩阵数据类型。正如这里所看到的，比起汇编语言来使用高级语言的一个优点就是程序员能够使用多种不同的数据类型。在所有这些语言中，都有存储向量和矩阵的内建数据类型，以及建立结构需要的数组和构造。许多不同的函数也能被内建到这些语言中，帮助计算三角函数值（sin、cos等）、最小值和最大值、指数函数（log2、sqrt、pow等）以及其他的数学或几何函数。

17.3.3 顶点着色器示例

用顶点着色器能控制顶点的光照和变换。顶点着色器也用于为片段着色器的执行创造条件。顶点着色器中一个有趣的地方是仍然能够使用几何模型的缓存机制,比如显示列表或VBO，因此当使用顶点着色器在GPU上作运算时可以提高性能。例如，如果顶点表示粒子，并能使用顶点着色器模拟粒子的运动，那么就几乎能脱离CPU而进行计算。由于CPU和GPU之间的数据传递而产生的性能瓶颈将会在最大程度上弱化。在顶点着色器出现之前，粒子运动的计算是在CPU上执行的，并且在应用程序的每次迭代中每个顶点会被再次送到图形硬件中。能在图形硬件内存中的顶点上执行计算是性能上一个很大的提高。

393

一种最简单的顶点着色器把顶点转换成裁剪坐标，并把朝前的颜色分配给顶点的颜色属性。

```
void main(void)
{
    gl_Position = gl_ModelViewProjectionMatrix *
                  gl_Vertex;
    gl_FrontColor = gl_Color;
}
```

在这个例子中，gl_ModelViewProjectMatrix是GLSL运行环境提供的一个内建的一致性变量。变量gl_Vertex和gl_Color是内建顶点属性。专有输出变量gl_Position和gl_FrontColor被顶点着色器用来设置变换的位置和顶点的颜色。

有一个更有趣的着色器，它实现了第9章中给出的表面明暗处理方程，给出了使用Phong明暗

处理算法对于每个顶点进行明暗处理的效果。

```
void main(void)
{
    vec4 v = gl_ModelViewMatrix * gl_Vertex;
    vec3 n = normalize(gl_NormalMatrix * gl_Normal);
    vec3 l = normalize(gl_LightSource[0].position - v);
    vec3 h = normalize(l - normalize(v));

    float p = 16;
    vec4 cr = gl_FrontMaterial.diffuse;
    vec4 cl = gl_LightSource[0].diffuse;
    vec4 ca = vec4(0.2, 0.2, 0.2, 1.0);

    vec4 color;
    if (dot(h,n) > 0)
        color = cr * (ca + cl * max(0,dot(n,l))) +
                cl * pow(dot(h,n), p);
    else
        color = cr * (ca + cl * max(0,dot(n,l)));

    gl_FrontColor = color;
    gl_Position = ftransform();
}
```

394

从这个着色器给出的代码中，应该能获得对着色器程序设计的感性认识，可以看到它类似C语言程序设计。在上述着色器中完成以下几个任务。首先，创建了一组变量保存计算Phong明暗处理所必需的向量：***v***、***n***、***l***、***h***。注意在顶点着色器中的计算是在眼睛空间中执行的。这么做有很多原因，一个原因是着色器要访问的光源位置已经被变换到眼睛坐标系中。当创建着色器时，决定使用的坐标系将很可能依赖于执行的计算类型，这是需要考虑的一个很重要的因素。同样，注意例子中内建函数和数据结构的使用。尤其是在着色器中使用的几个函数：normalize、dot、max、pow和ftransform，这些函数由着色器语言提供。另外，和材质、光照相关的图形状态通过内建的一致性变量gl_FrontMaterial和gl_LightSource[0]访问。材质和光源的漫反射分量通过这些

395

变量的diffuse成员来访问。顶点的颜色使用公式（9-8）计算并存储在专用输出变量gl_FrontColor里。顶点位置通过函数ftransform进行变换，用这个函数便于执行对模型矩阵和投影矩阵的乘法运算。图17-7显示了在不同的镶嵌球体上运行顶点着色器的结果。因为计算是在每个顶点上进行的，要在球体上产生正确的Phong高光需要大量的几何模型。

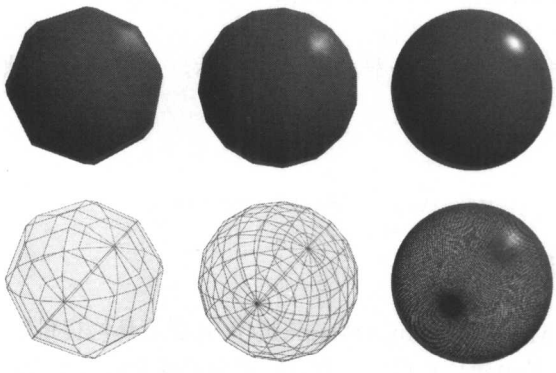


图17-7　每个球体仅使用一个顶点着色器计算Phong明暗处理。因为计算是基于每个顶点偏差进行的，只有在用于建立球体模型的几何模型的数量大大增加后，Phong高光看起来才是正确的（另见彩图VIII）

17.3.4 片段着色器示例

编写片段着色器的方式与非常相似于顶点着色器，为了强调这一点，下面将用一个片段着色器实现第9章中的公式（9-8）。为此，首先需要编写一个顶点着色器来为片段着色器创造条件。

这个例子所需的顶点着色器是相当简单的，但是仍然使用可变变量把数据传送到片段着色器中。

```
varying vec4 v;
varying vec3 n;

void main(void)
{
    v = gl_ModelViewMatrix * gl_Vertex;
    n = normalize(gl_NormalMatrix * gl_Normal);

    gl_Position = ftransform();
}
```

回想一下，顶点着色器是基于每个顶点对可变变量进行设置的。但是当可变变量在片段着色器中被访问的时候，它们在三角形或几何图元上将会改变（例如被插值）。在这种情况下，眼睛空间v中的顶点位置和顶点n的法线都在每个顶点上计算。顶点着色器执行的最后的计算是把顶点变换成裁剪坐标，因为片段着色器将在每个片段上计算光照。在顶点着色器中设置朝前的颜色是没有必要的。

片段着色器程序使用Phong明暗处理模型在每一个片段上计算光照。

```
varying vec4 v;
varying vec3 n;

void main(void)
{
    vec3 l = normalize(gl_LightSource[0].position - v);
    vec3 h = normalize(l - normalize(v));

    float p = 16;
    vec4 cr = gl_FrontMaterial.diffuse;
    vec4 cl = gl_LightSource[0].diffuse;
    vec4 ca = vec4(0.2, 0.2, 0.2, 1.0);

    vec4 color;
    if (dot(h,n) > 0)
       color = cr * (ca + cl * max(0,dot(n,l))) +
                    cl * pow(dot(h,n),p);
    else
       color = cr * (ca + cl * max(0,dot(n,l)));

    gl_FragColor = color;
}
```

396

第一个要注意的是这个例子中的片段着色器代码和17.3.3小节给出的顶点着色器代码的相似之处。主要的不同是在可变变量v和n的使用上。在片段着色器中，观察向量和法线值是在模型表面上在相邻顶点之间插值得到的。结果显示在图17-8中。我们立即就能注意到四边形模型上的Phong高光，该模型仅仅包含了四个顶点。因为着色计算在片段级上进行，使用带有插值数据（即可变数据）的Phong方程，所以用很少的几何模型就得到了更一致而准确的Phong明暗处理效果。

397

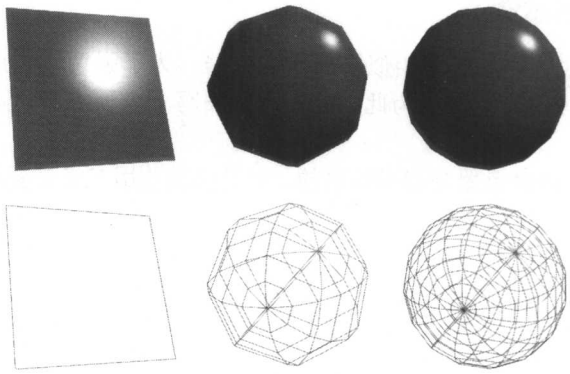


图17-8　运行17.3.4节的片段着色器得到的结果。注意在最左边的模型的确出现了Phong高光，这个模型是用一个多边形表示的。实际上，因为光照是在片段上而不是在每个顶点上计算，所以较粗糙的镶嵌球体模型也出现了适当的Phong明暗处理效果（另见彩图IX）

17.3.5　GPU上的通用计算

在学习了顶点和片段着色器的实例之后，你可能想知道是否能编写程序在GPU上执行其他类型的计算。很显然，回答是肯定的，因为有各种在GPU上进行程序设计的语言，很多问题能被编码在GPU上运行。尽管如此，仍然要记住几个重要的事实。首先，当前图形硬件上处理的浮点数不是双精度的。其次，可能需要把问题变换成适合图形相关的框架的形式。换句话说，需要使用图形API设置问题，使用纹理图作为数据而不是传统的存储器，并编写顶点和片段着色器去构架和解决问题。

正如已经声明的，GPU仍可能是一个具有吸引力的计算平台，因为在GPU中执行计算所用的晶体管的比率要比在CPU中得高得多。在许多情况下，算法在GPU上运行得要比在CPU上快。而且，GPU执行单指令流多数据流计算，特别是在片段处理级。实际上，可以把片段着色器上的计算看作是一个普通的指令`foreach`在高度并行地执行，在一组图元上同时执行操作。

在GPU上执行通用计算方面已经有了大量的研究，这就是常常提到的GPGPU。研究者正使用GPU模拟云的动态运动（Harris，Baxter，Scheuermann, & Lastra，2003）、实现光线跟踪（Purcell et al., 2002；Carr，Hall, & Hart，2002）、计算辐射度（Coombe，Harris, & Lastra，2004），使用水平截集进行三维分割（A.E.Lefohn Kniss，Hansen, & Whitaker 2003），或者解决Navier-Stokes方程（Harris，2004）。

通用计算通常在GPU上使用多个绘制“通道”执行，并且由于片段处理器使用数据高度并行设计，大部分计算都用片段处理器完成。每一个通道称为一个内核，它完成计算中的一个部分。内核是工作在数据流上的，多个内核连在一起形成全部的计算。第一个内核完成计算的第一部分，
398 第二个内核在第一个内核数据的基础上进行工作，依此类推直到完成计算。在这种程序设计风格中，在GPU上使用数据和数据结构的方式不同于传统的程序设计，必然要再做一点思考。幸运的是，最近的研究为在GPU程序设计中建立高效的数据结构提供了一些抽象和信息（A. Lefohn, Kniss, & Owens, 2005）。

使用GPU进行通用设计必然要求懂得怎样使用图形硬件。例如，绝大部分执行GPGPU的应用程序都要绘制一个四边形或者一组四边形，这一过程是顶点和片段着色器对几何模型进行操作实现的。几何模型不必是可见的或者不需要绘制到屏幕上，但是应该允许进行顶点和片段操作。正因如此，学习在硬件上进行通用计算是比较困难的。幸运的是，最近的研究正在使面向GPU的

接口更像传统的程序设计。*Brook for GPUs*（Buck et al., 2004）正是这样一个系统：它提供了一个类C的接口支持在GPU上的流水计算，这将使更多的人能充分利用现代图形硬件上的计算功能。

常见问题

- **如何调试着色器程序？**

在大部分的平台上，顶点和片段着色器的调试并不简单。通常，对于调试图形应用程序很少有实时的支持，而对着色器程序进行实时的调试更不可能。然而，这一点正在改变。在最新版本的Mac OS X、Linux和Windows中，已经有对着色器设计的支持。对着色器程序进行调试的好办法就是使用一种图形硬件开发厂商提供的着色器开发工具。

文献注记

要学习更多涉及图形硬件程序设计的技术细节，有许多不错的资源。*OpenGL™ Programming Guide*（Shreiner, et al., 2004）一书应该是学习的一个好起点。*OpenGL™ Shading Language*（Rost, 2004）和*The Cg Tutorial*（Fernando & Killgard, 2003）这两本书详细讲述了如何用着色语言编程。399
在GPU Gems系列丛书（Fernando，2004；Pharr & Fernando，2005）中讲述了有关顶点和片段处理器程序设计更高级的技术信息和实例。在GPGPU.org网站（http://www.gpgpu.org）上可以了解到更多有关GPU上通用计算的信息。

习题

1. 与在CPU上执行操作相比，在GPU上执行会有多快？编写一个程序，用参数控制在GPU中处理的数据量，即从不使用着色器程序到所有的计算都在着色器程序中执行。当完全在GPU上执行计算时，应用程序的性能会发生怎样的改变？
2. 是否存在某些特定长度的三角形条带，比其他长度的效率更高？试确定使性能最好的三角形条带的最大长度。在图形硬件的内存或缓存结构上，它能给你带来什么样的启示？ 400

第18章

构建交互式图形应用程序

Kelvin Sung

本书大部分章节讨论的都是计算机图形学领域的基本算法。本章将这些算法集成到应用程序中。这是一个很重要的主题，因为仅有基本的图形学算法知识，在实际的应用程序中实现这些算法的时候，并不能总是得到最佳的实现。

本章将从一个简单的示例开始：一个允许用户（在重力的影响下）模拟击球的程序。用户可以创建不同大小和不同初始速度的球，射击该球，然后检验该球的抛物线自由落体运动。需要的基本概念有：网格结构，用来表示球（球体）；纹理映射；照明；明暗处理，使球的外观具有一定美感；球的轨迹变换；光栅化技术，用来生成球的图像。

为了实现这个简单的击球程序，还需要以下知识：

- 图形用户界面（GUI），以便进行有效的用户交互。
- 软件体系结构和设计模式，用来制作一个易于维护和扩展的实现框架。
- 应用程序接口（API），用来选择适当的底层支持，以避免大量不必要的代码。

为了按照上述三个方面的要求建立应用程序，需要完成以下步骤：

- 分析交互式应用程序。
- 掌握不同的编程模型，并识别这些模型的功能组件。
- 定义组件的交互行为。
- 设计解决方案的组件集成框架。
- 在现有不同API集合的基础上演示实现示例。

我们将以击球程序作为示例，并从具体细节规范的精炼开始。为简单起见，本示例将被限制在二维空间，以避免三维空间的复杂性。显然，这个简单的程序并不是复杂，也不代表实际的应用程序。但是，规范稍微精炼后，这个示例包含所有基本组件，以及较复杂的真实交互系统的行为特性。

新的概念将不断添加进来，增加这个简单示例的复杂性，直到获得一个适合于建立一般交互式图形应用程序的软件体系结构。我们将分析结果的有效性，并讨论如何将这个简单示例的经验教训应用到其他熟悉的真实应用程序中去（例如PowerPoint、Maya等）。

18.1 击球程序

击球程序包含以下元素和行为：

- **球（对象）**：用户可以在屏幕上任意位置单击鼠标左键拖出一个新球（圆）（见图18-1）。拖出一个球的操作包括：
  - （A）初始鼠标单击位置指定圆心。
  - （B）按下鼠标键并移动鼠标，就是拖动操作。
  - （C）拖动时当前鼠标的位置允许指定半径和初始速度。半径R（以像素为单位）为到圆

402

心（在步骤(A)定义）的距离。从当前位置到圆心的向量就是初始速度V（以像素/秒为单位）。一旦创建好了，球就会以指定的初速度开始滑动。

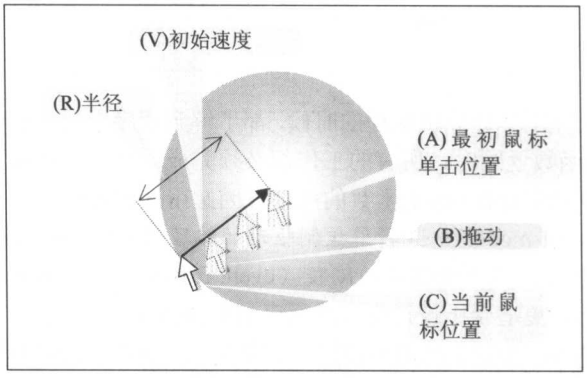


图18-1 拖出一个球

- **HeroBall（Hero/活动对象）**：用户可以单击鼠标右键，选择一个球作为当前的HeroBall。HeroBall的速度可以通过滑动条（下面讨论）来控制，滑动条会显示其速度。（最近创建的球被默认为当前的HeroBall。）在空白区域单击鼠标右键表示当前没有HeroBall。
- **速度滑动条（GUI元素）**：用户监视并控制两个滑动条（分别对应*x*方向和*y*方向的大小）来改变HeroBall的速度。在没有HeroBall的时候，滑动条的值是不明确的。
- **模拟**：
  - **球的滑行/碰撞（对象固有的行为）**：球知道如何根据当前的速度滑行，并且可能与另一个球相撞。为了简化，假定所有球的质量都一样，并且所有碰撞都是完全弹性的。
  - **重力（球的外部作用）**：由于重力的作用，球的速度不断变化。
  - **状态条（应用程序状态显示）**：通过分析状态条中的信息，用户可以监视应用程序的状态。在击球应用程序中，状态条会更新显示当前屏幕上球的数量。

403

击球应用程序启动时屏幕是空的。用户按下鼠标并拖动就可以创建不同半径和速度的新球。球一旦滑出屏幕就会被清除。为了避免不必要的细节，模拟过程中没有绘制运动轨迹和速度向量。注意状态条通过两种方式将当前状态告知用户：滑块的位置和显示的数值（见图18-2）。

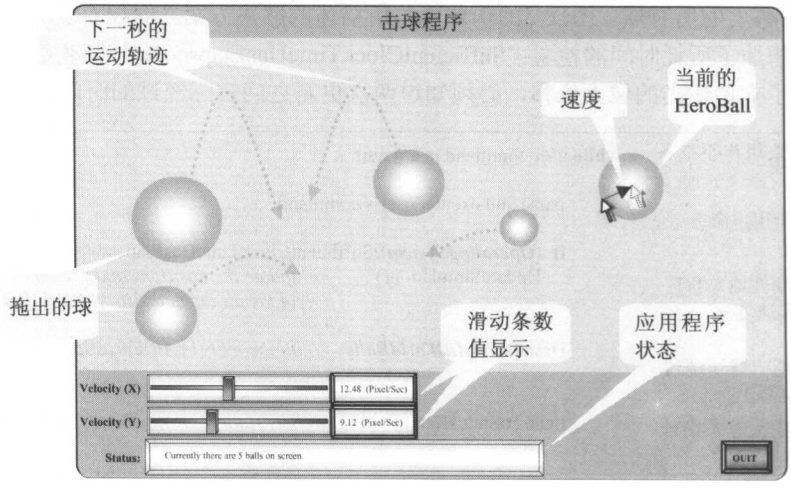


图18-2 击球程序

至此，我们介绍了一个简单图形交互应用程序的行为。本章的剩余部分将介绍与这类应用程序的实现有关的概念。

18.2　编程模型

我们当中的很多人在刚学计算机编程的时候，都了解到程序应该从*main*()函数开始，以*main*()的返回而结束，*main*()函数返回时，所有的工作都必须完成，然后终止程序。由于在整个程序的生命周期中，所有控制都保持在*main*()函数的内部，因此用这种方法求解问题的模型被称为内部控制模型（internal control model），或者是控制驱动的编程（control-driven programming）。稍后我们将看到，另一种方式，即事件驱动的编程（event-driven programming）或称外部控制模型
404 （external control model），更适合于设计交互式程序的解决方案。

本节将先在比较熟悉的控制驱动的编程模型的基础上，阐明二维击球程序的解决方案。然后分析该方案，识别其缺点，描绘外部控制模型或事件驱动编程方法的动机。

采用的伪代码类似于C++/Java。假设已经具备了来自操作系统的典型功能（*OperatingSystem::*）以及图形用户界面API的功能（*GUISystem::*）。伪代码主要用来帮助分析解决方案的基本控制结构（即if/while/case）。因此，应用程序的操作细节以及特定于图形操作的细节都被特意屏蔽了。例如，省略了如何*UpdateSimulation*()的细节。

18.2.1　控制驱动的编程

控制驱动编程的主要优势在于可以相当直观地将解决方案的口头描述翻译为程序的控制结构。在这种情况下，我们的解决方案可以描述为：

只要用户不想退出（A）；
解析并执行用户的命令（B）；
更新球的速度和位置（C）；
然后绘制所有的球（D）；
405 最后在获取另一个用户命令之前，将当前应用程序的状态显示到状态条中，告诉用户正在发生的事情（E）。

图18-3给出了将这一口头描述的解决方案翻译成简单编程结构的结果。集合AllWorldBalls用来表示当前屏幕上的所有球。图18-3的伪代码与所描述的解决方案之间仅有的另一个差别是，在步骤（C）中增加了所耗时间的检验：SufficientClockTimeHasElapsed（注意速度是按照像素/秒来定义的）。为了支持适当的像素位移，必须知道两次更新之间实际经过的时间。

```
（A）只要用户不准备退出    while user command is not quit
（B）解析用户命令              parse and excute  user's command
（C）周期性地更新球的位置和速度   if (OperatingSystem::SufficientClockTimeHasElapesd)
                                  UpdateSimulation()      // update the positions and velocities
                                                          // of the all the balls (in AllWorldBalls set)
（D）将所有球绘制到计算机屏幕上   DrawBalls(AllWorldBalls)  // all the balls in AllWorldBalls set
（E）设置状态条中球的数目         EchoToStatusBar()       // Sets status bar: number of balls on screen
```

图18-3　来自描述方案的编程结构

由于增加了解析执行用户命令的额外细节，所以必须扩展解决方案。在图18-4中，修订后的解决方案展现了中央解析切换语句（B）的详细情况，以及对用户可以发布的三条命令的支持情况：定义新的HeroBall（B1）、选择HeroBall（B2）、使用滑动条调整当前HeroBall的速度（B3）。未定义的用户动作（例如，未按键时的鼠标移动）被简单地忽略了（B4）。

请注意HeroBall的创建涉及三个用户动作：按下鼠标（B1）、随后拖动鼠标（B1-1）、最后松开鼠标（B1-2）。对这一操作的解析是通过多次连贯的外层while循环（A）来实现的：第一遍循环，创建新的HeroBall（B1），在后面的循环中执行实际的拖动操作（B1-1）。假设在没有按下鼠标键（B1）的时候，从不会调用鼠标拖动（B1-1），那么在拖动操作期间HeroBall就一直是有定义的。

动作LeftMouseButtonUp（B1-2）是一个隐含的动作，在最初的规范中没有定义这一动作。在我们的实现中，选择这一隐含动作来激活插入操作，将新的HeroBall插入到AllWorldBalls集合中。这样在用户完成拖动操作之前，HeroBall一直不是AllWorldBalls集合的元素。这一延迟确保了*UpdateSimulation*()过程更新AllWorldBalls集合中所有球的时候（C），HeroBall的速度和位置不受影响。这意味着用户可以利用该时间拖出一个新的HeroBall，而不用担心该球在释放鼠标键之前会自由降落。绘制操作（D1）的简单修正确保了HeroBall在插入AllWorldBalls集合之前的正确绘制。

当在支持用户交互的上下文中分析这一解决方案时，必须关注效率问题以及可能增加的复杂性。

```
main()  {
(A)     while ( GUISystem::UserAction != Quit )  {
(B)         switch (GUISystem::UserAction)  {
(B1)定义新的HeroBall
                // Begins creating a new Hero Ball
                case GUISystem::LeftMouseButtonDown:
                    HeroBall = CreateHeroBall()      // hero not in AllWorldBalls set
                    DefiningNewHeroBall = true
(B1-1)对拖动的支持
                // Drags out the new Hero Ball
                case GUISystem::LeftMouseButtonDrag:
                    RefineRadiusAndVelocityOfHeroBall()
                    SetSliderBarsWithHeroBallVelocity()
(B1-2)隐含的动作
                // Finishes creating the new Hero Ball
                case GUISystem::LeftMouseButtonUp:
                    InsertHeroBallToAllWorldBalls()
                    DefiningNewHeroBall = false
(B2)选择当前的HeroBall
                // Selects a current hero ball
                case GUISystem::RightMouseButtonDown:
                    HeroBall = SelectHeroBallBasedOnCurrentMouseXY()
                    if (HeroBall != null)
                        SetSliderBarsWithHeroBallVelocity()
(B3)设置HeroBall的速度
                // Sets hero velocity with slider bars
                case GUISystem::SliderBarChange:
                    if (HeroBall != null)
(B4)忽略未定义的操作
                        SetHeroBallVelocityWithSliderBarValues()
                // Ignores all other user actions e.g. Mouse Move with no buttons, etc
                default:
            }  // end of switch(userAction)
(C)         // Move balls by velocities under gravity and remove off-screen ones
            if ( OperatingSystem::SufficientClockTimeHasElapesd)
                UpdateSimulation()
(D)         DrawBalls(AllWorldBalls)
(D1)绘制新的HeroBall
                // Draw the new Hero Ball that is currently being defined
            if (DefiningNewHeroBall)
                DrawBalls(HeroBall)
(E)         EchoToStatusBar()    // Sets Status Bar with number of balls currently on screen
        }  // end of while(UserAction != Quit)
    }  // end of main() function. Program terminates.
```

图18-4 基于控制驱动编程模型的编程方案

1. 效率问题

通常用户是以活动爆发的方式与应用程序交互的——一系列连续动作之后会空闲一段时间。这可以通过以下事实来解释：作为用户，我们通常在应用程序中执行某些任务，然后花一些时间来分析结果。例如，在使用文字处理器的时候，常见的工作模式就是在爆发式的输入/编辑操作之后停下来阅读一段时间（此时没有输入动作）。在我们的示例应用程序中，可以预期用户拖出
406 ~ 407
一些圆，然后观察这些圆的自由降落。在*main*()函数中通过持续的while循环登记注册用户命令，意味着在用户没有执行任何操作的时候，程序仍然在活跃地运行而浪费机器资源。在活动爆发期间，用户每秒钟最多可能产生成百上千的输入动作（例如，每秒钟内的鼠标像素移动）。如果将这一速率与常见的每秒高达10^9的CPU指令性能相比，会在巨大的差异中发现，即使是在活动爆发期间，解析用户命令的切换语句（B）大部分时间都花费在default情况中，没有做任何事情。

2. 复杂性问题

注意整个解决方案都在*main*()函数内。这意味着所有相关的用户操作都必须经过用户命令解析语句（B）的解析。在现代多个程序共享的窗口环境下，用户执行的很多动作实际上都不是特定于某个应用程序的。例如，用户如果单击鼠标左键并在程序窗口的绘图区域拖动，那么击球程序就应该响应用户操作，拖出一个新的HeroBall。然而，如果用户在程序窗口的标题区域执行这样的操作，应用程序则应该将这些操作转发给GUI/操作/窗口系统，并开始移动整个应用程序窗口的坐标。体验过窗口环境的用户都知道，这种非特定于应用程序的操作有很多，并希望所有应用程序都遵循这些操作（比如，图标化、改变大小、提升或降低窗口，等等）。按照图18-4给定的解决方案，对于想要遵循的用户操作，都必须在解析切换语句（B）中包括一种相匹配的支持情况。这种需求将迅速增加解决方案的复杂性，并成为交互式应用程序的实现负担。

一个高效GUI系统在默认情况下应该是空闲的（不占用机器资源），并且只在交互活动（比如，用户输入动作）出现时才活跃起来。此外，为了将交互式应用程序集成到成熟的多程序窗口环境中，让支持GUI的系统自动照顾标准的用户动作，是至关重要的。

18.2.2 事件驱动的编程

事件驱动的编程使用GUI系统定义的*MainEventLoop*()纠正了所关心的效率和复杂性问题。对于事件驱动的程序，*MainEventLoop*()取代了*main*()函数，因为所有程序都在这个函数里面启动和
408
终止。与控制驱动编程的*main*()函数一样，*MainEventLoop*()函数返回时，所有工作都应该完成了，然后程序终止。*MainEventLoop*()函数定义了所有事件驱动编程方案的中央控制结构，一般不能通过用户应用程序来改变它。这样，整个应用程序的控制实际上是在用户程序代码的外面。由于这一原因，事件驱动编程也被称为外部控制模型。

图18-5描绘了一种常见的*MainEventLoop*()实现方案。在这种情况下，我们的击球程序成为用户应用程序，建立在*MainEventLoop*()函数的基础之上。在结构上，*MainEventLoop*()非常类似于图18-4中的*main*()函数：在一个持续的循环（B）中包含一个中央解析切换语句（D）。这两个函数之间的重要区别有：

- （A）*SystemInitialization*()：事件驱动的程序是在*MainEventLoop*()函数内启动和终止的。*SystemInitialization*()是一种从*MainEventLoop*()内部调用用户程序的机制。用户程序需要实
409
现*SystemInitialization*()来初始化应用程序的状态，并注册事件服务例程（见（D）的讨论）。
- （B）持续的外层循环：由于这是所有事件驱动应用程序共享的一般控制结构，所以没有任何终止循环的条件。用户程序应该重载适当的事件服务例程，并在服务例程内部终止程序。
- （C）停止并等待：*MainEventLoop*()并不是主动登记注册用户动作（那样会浪费机器资源），而是终止整个应用程序的处理，等待异步操作系统调用，在相关用户动作出现的时候重

新激活应用程序的处理过程。

- （D）事件与中央解析切换语句：包含在该语句中的是所有用户可能执行的动作/事件（情况）。与每一事件（情况）相关联的是一种默认的行为和一个触发器。该触发器允许用户应用程序重载这种默认的行为。在*SystemInitialization*()期间，用户应用程序可以通过触发重载来注册事件的替代服务例程。

为了开发事件驱动的解决方案，我们的程序必须先向GUI系统注册事件服务例程。然后，将整个程序的解决方案建立在等待并服务用户事件的基础上。控制驱动编程的解决方案建立在*main*()函数控制结构的算法组织基础之上，而事件驱动编程的解决方案却是建立在事件规范的基础之上。事件规范能够将应用程序转变到指定的状态上。这就是设计编程方案的不同范例。这里关键的区别在于程序员没有明确控制事件的算法组织：事件应该在什么条件下出现，何时出现，或者是以什么样的频率出现。

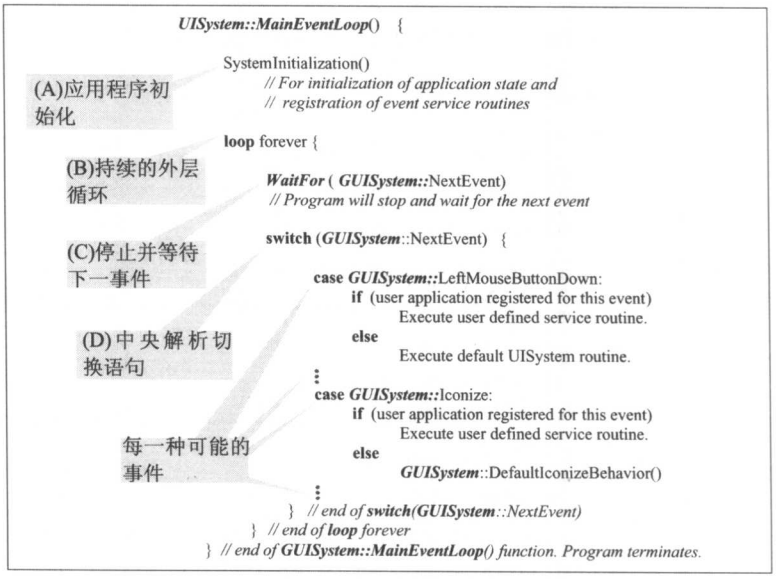


图18-5 默认的*MainEventLoop*函数

图18-6的程序实现了击球程序的鼠标左键操作。可用看到在系统初始化期间（A），程序定义了一个适当的应用程序状态（A1），并注册了鼠标左键（LMB）的按下、拖动、弹起事件（A2）。同时还定义了相应的事件服务例程（D1、D2和D3）。在每个事件服务例程结束的时候，重新绘制了所有球，以确保用户一直能够看到最新的显示。请注意这里没有任何组织初始化过程和服务例程的控制结构。这就是事件驱动的程序：整个控制结构都定义在解决方案外部的*MainEventLoop*函数中。

图18-7展示了如何将图18-6中的程序连接到GUI系统中预先定义好的*MainEventLoop*()。*MainEventLoop*()调用在我们的解决方案中定义的*SystemInitialization*()函数（A）。按照描述，在初始化之后，整个程序基本上就是三个服务例程（D1、D2和D3）。但是我们对这些例程的调用没有任何控制。实际上，是由用户动作触发这些事件，然后驱动这些例程。这些例程反过来改变应用程序的状态。按照这种方式，事件驱动编程的解决方案就是建立在事件（LMB事件）规范的基础之上。事件规范将应用程序转变到指定的状态上（AllWorldBalls和HeroBall）。由于*MainEventLoop*()内部解析用户命令的切换语句（图18-7中的D），对于每个可能的用户动作都包含了一种情况以及相应的默认行为，所以我们的解决方案没有增加复杂性就遵循了环境中非特定于应用程序的操作（如，图标化、移动，等等）。

410

```
(A) System Initialization:
    (A1): Define Application State:
          AllWorldBalls: A set of defined Balls, initialze to empty
          HeroBall: current active ball, initialize to null

    (A2): Register Event Service Routines
          Register for: Left Mouse Button Down Event
          Register for: Left Mouse Button Drag Event
          Register for: Left Mouse Button Up Event
                              // We care about these events, inform us if these events happen

(D) Events Services:
    (D1): Left Mouse Button Down// service routine for this event
          HeroBall = Create a new ball at current mouse position
          DrawAllBalls(AllWorldBalls, HeroBall) // Draw all balls (including HeroBall)

    (D2): Left Mouse Button Drag // service routine for this event
          RefineRadiusAndVelocityOfHeroBall()
          DrawAllBalls(AllWorldBalls, HeroBall) // Draw all balls (inlucding HeroBall)

    (D3): Left Mouse Button Up   // service routine for this event
          InsertHeroBallToAllWorldBalls()
          DrawAllBalls(AllWorldBalls, null)      // Draw all balls
```

图18-6　一个简单的事件驱动程序的规范

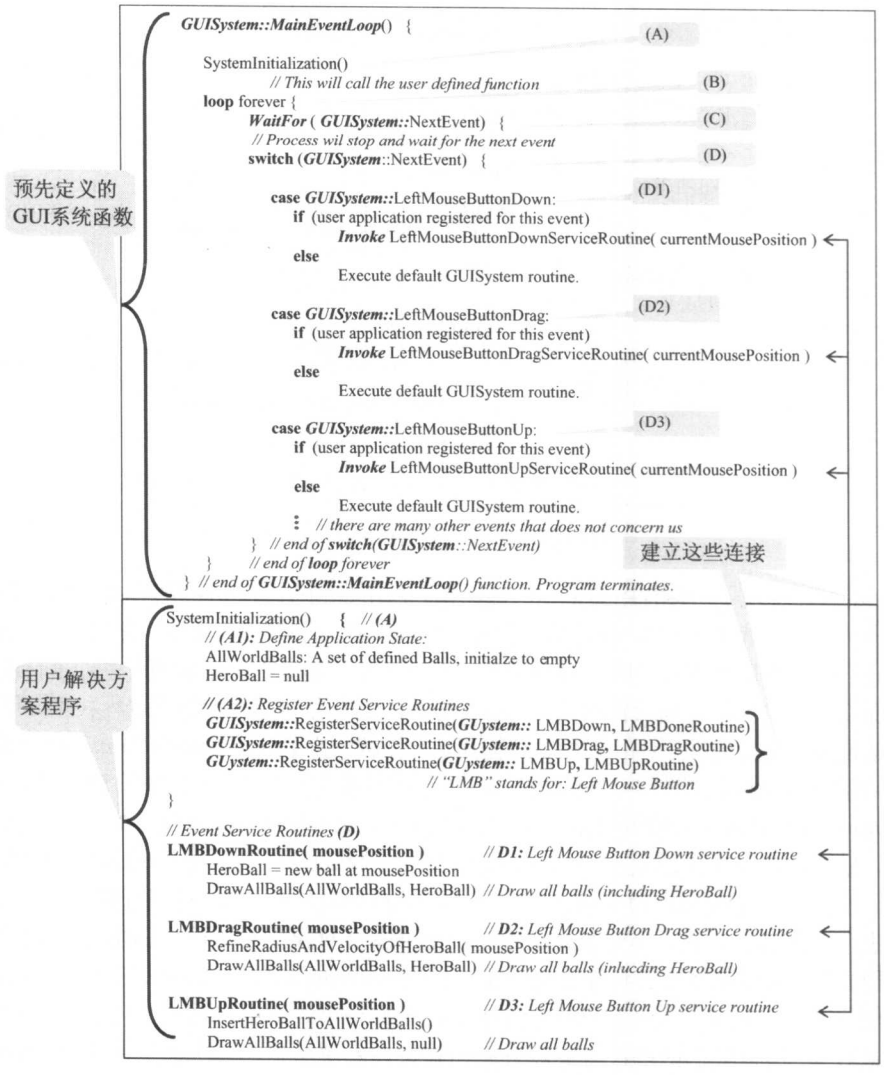


图18-7　解决方案与*MainEventLoop*的连接

在事件驱动编程的上下文中，可以将事件看作异步通知，表明发生了某些所关注的事情。该通知的信使就是底层的GUI系统。接收事件的机制是通过重载相应事件服务例程来实现的。

由于这些原因，在讨论事件驱动编程时，总有一个GUI支持系统。这一GUI系统一般称为图形用户界面（Graphics User Interface，GUI）应用编程接口（Application Programming Interface，API）。这样的GUI API实例包括Java Swing Library、OpenGL Utility ToolKit(GLUT)、The Fast Light Toolkit(FLTK)、Microsoft Foundation Classes (MFC)，等等。

根据上面的讨论，可以看出设计并开发事件驱动程序解决方案的核心是注册适当的事件服务。在开发完整的击球程序解决方案之前，需要花一些时间来理解事件。 411

图形用户界面（GUI）事件

通常，应用程序会收到很多由用户、应用程序本身或者是GUI系统产生的事件。本节将描述这些事件来源，以及应用程序在服务这些事件时的作用。

S1：用户。这些事件是由用户在输入设备上执行的动作触发的。注意输入设备包括实际的硬件设备（如鼠标、键盘等）以及软件模拟的GUI元素（如滑动条、组合框等）。通常，用户执行动作有两个完全不同的原因： 412

- **S1a：特定于应用程序的。**这些输入动作是应用程序的组成部分。在应用程序的屏幕区域单击拖动创建一个HeroBall，就是这种在硬件输入设备上执行的操作实例。改变滑动条控制HeroBall的速度，就是在软件模拟的GUI元素上执行的操作实例。这些操作以及所产生的事件都是特定于应用程序的。
- **S1b：一般的。**这些输入动作是由操作环境定义的。例如，用户在窗口的标题栏单击鼠标并拖动，将移动整个应用程序窗口。这类事件的服务要求应用程序与GUI系统之间的协作。在分析S3c中源自GUI系统的事件时，将更详细地讨论这类事件的服务。

注意用户动作的含义是上下文敏感的（context sensitive），依赖于动作的执行场所：是在应用程序的屏幕区域单击并拖动，还是在滑动条上，或者是应用程序窗口的标题栏上。在任何一种情况下，都是由底层的GUI系统负责解析上下文关系，并确定哪个应用程序元素接收某个特殊事件。

S2：应用程序。这些事件是由应用程序定义的，通常取决于一些运行时的条件。在运行期间，如果条件满足，GUI支持系统就会触发事件，并将满足的条件传递给应用程序。一个简单的例子就是周期性的报警。现代的GUI系统通常允许应用程序定义定时器事件（timer event）（有时为多个）。一旦定义了，GUI系统将在定时器到期的时候，触发一个事件，唤醒应用程序。我们将看到，定时器事件是支持实时模拟的基本要求。因为应用程序（我们的程序）请求产生这些类型的事件，所以我们的程序只是负责服务这些事件。应用程序定义的事件与用户产生的事件之间重要区别是，应用程序定义的事件可以是自发产生的（spontaneous）：只要定义好了，即使用户没有做任何事情，也可能触发这类事件。

S3：GUI系统。这些事件源自GUI系统，是为了将状态信息传达给应用程序。产生这些事件的原因通常有三个： 413

- **S3a：内部GUI状态变化了。**这些事件发出信号高知内部GUI系统状态的改变。例如，GUI系统通常在应用程序的主窗口创建之前产生一个事件。该事件为应用程序提供了一个执行相应初始化工作的机会。在一些GUI系统中（比如MFC），*SystemInitialization*()的功能就是利用这类事件完成的：用户应用程序重载适当的窗口创建事件，初始化应用程序的状态。现在，一般的商用的GUI系统都定义了大量事件，发出信号通知应用程序中详细的状态变化，以支持各种不同的应用程序和需求。例如，为了创建应用程序的主窗口，GUI系统可以针对下列状态来定义事件：

  —在资源分配之前。

—在资源分配之后，但是初始化之前。
—在初始化之后，但是在开始绘制之前，等等。
一个GUI系统通常为这样的事件定义了有意义的默认行为。为了编写高效的基于GUI系统的应用程序，必须掌握不同的事件群组，并且只服务于适当的事件。

- **S3b：外部环境请求关注**。这些事件指出操作环境发生了变化，可能需要应用程序关注。例如，用户可能移动另外一个应用程序窗口，而覆盖了我们的应用程序窗口的一部分，或者是用户最小化了我们的应用程序窗口。GUI系统和窗口环境一般都有适当的针对这类事件的服务例程。应用程序将只在有特殊动作需要执行的时候，才选择服务这些事件。例如，在实时模拟程序中，如果应用程序窗口被最小化了，应用程序可能选择延缓模拟。在这种情况下，应用程序就必须处理（服务）最大化和最小化事件。
- **S3c：外部环境请求应用程序协作**。这些事件通常请求应用程序协作完成一般用户动作（general user action）的服务（见S1b）。例如，用户点击拖动应用程序窗口的标题栏，GUI系统将做出回应，让用户“拖动”整个应用程序窗口。这一“拖动”操作的实现实际上
414 是在计算机显示器上当前鼠标指针位置上不断地擦除并重画整个应用程序窗口。GUI系统完全知道应用程序窗口的外观（如窗口框架、菜单等），但是不知道应用程序窗口内容（比如，有多少个自由降落的球、球以什么样的速度滑行，等等）。在这种情况下，GUI系统将重新绘制应用程序窗口框架，并为应用程序产生Redraw/Paint事件，请求帮助完成用户“拖动”操作的服务。作为一个共享窗口环境中的应用程序，我们的程序应该尊重并服务这些事件。在这类事件中最常见的事件有：Redraw/Paint和Resize。Redraw/Paint是应用程序必须服务的最重要事件，因为它支持用户在共享窗口环境中可能执行的最常见的操作。Resize也是一个应用程序必须响应的重要事件，因为应用程序掌握了GUI元素的放置策略（例如，如果窗口大小增加了，那么在更大的窗口中该如何放置GUI元素）。

18.2.3　事件驱动的击球程序

在18.2.1节，介绍了击球程序控制驱动的编程方案。该方案建立在各种条件（控制）下应该采取的适当动作的描述基础之上：

```
只要条件满足，解析输入……
```

我们已经看到，对此稍加修改就能够详细描述解决方案的控制结构。
按照18.2.2节的讨论，设计事件驱动的编程方案，必须
(1) 定义应用程序状态。
(2) 描述用户动作如何改变应用程序状态。
(3) 将用户动作映射到GUI系统支持的事件。
(4) 重载相应的事件服务例程，实现用户操作。

415 18.1节中的规范详细描述了击球程序的行为。该描述的基础是常见输入设备上（如滑动条、鼠标）的动作，这些动作改变显示屏幕的外观。这样，18.1节的规范就描述了上述列表中第（2）项和第（3）项，但是没有明确定义应用程序的状态是什么。设计解决方案的工作就是导出隐含定义的应用程序状态，并设计适当的服务例程。

图18-8给出了事件驱动的编程方案。按照预期的方式，在*SystemInitialization*()中定义应用程序的状态（A1）。从18.1节的规范可以导出集合AllWorldBalls以及HeroBall。标志*Defining-NewHeroBall*是一种过渡的（临时的）应用程序状态，用来支持跨越多个事件的用户动作（单击并拖动）。使用过渡的应用程序状态，是一种常见的支持内部连续事件的方法。

```
SystemInitialization() {     // (A)
    // (A1): Define Application State
        AllWorldBalls: A set of defined Balls, initialze to empty
        HeroBall = null
        DefiningNewHeroBall = false

    // (A2): Register Event Service Routines
        // S1a: Application Specific User Events
            GUISystem::RegisterServiceRoutine(GUISystem:: LMBDown, LMBDownRoutine)
            GUISystem::RegisterServiceRoutine(GUISystem:: LMBDrag, LMBDragRoutine)
            GUISystem::RegisterServiceRoutine(GUISystem:: LMBUp, LMBUpRoutine)
            GUISystem::RegisterServiceRoutine(GUISystem:: RMBDown, RMBDownRoutine)
            GUISystem::RegisterServiceRoutine(GUISystem:: SliderBar, SliderBarRoutine)
        // S2: Application Define Event
            GUISystem::DefineTimerPeriod(SimulationUpdateInterval)
            GUISystem::RegisterServiceRoutine(GUISystem:: TimerEvent, ServiceTimer)
                                    // Triggers TimerEvent every: SimulationUpdateInterval period
        // S3c: Honor collaboration request from the GUI system
            GUISystem::RegisterServiceRoutine(GUISystem:: RedrawEvent, RedrawRoutine)
}

// Event Service Routines (D)
LMBDownRoutine( mousePosition )    // D1: Left Mouse Button Down service routine
    HeroBall = CreateHeroBall (mousePosition)
    DefiningNewHeroBall = true
    GUISystem::GenerateRedrawEvent

LMBDragRoutine( mousePosition )    // D2: Left Mouse Button Drag service routine
    RefineRadiusAndVelocityOfHeroBall( mousePosition )
    SetSliderBarsWithHeroBallVelocity()
    GUISystem::GenerateRedrawEvent      // Generates a redraw event

LMBUpRoutine( mousePosition )      // D3: Left Mouse Button Up service routine
    InsertHeroBallToAllWorldBalls()
    DefiningNewHeroBall = false

RMBDownRoutine ( mousePosition )   // D4: Right Mouse Button Down service routine
    HeroBall = SelectHeroBallBasedOn (mousePosition )
    if (HeroBall != null)  SetSliderBarsWithHeroBallVelocity()

SliderBarRoutine ( sliderBarValues )   // D5: Slider Bar changes service routine
    if (HeroBall != null)
        SetSliderBarsWithHeroBallVelocity( sliderBarValues )

ServiceTimer ( )                   // D6: Timer expired service routine
    UpdateSimulation( )                 // Move balls by velocities  and remove off-screen ones
    EchoToStatusBar( )                  // Sets status bar with number of balls on screen
    GUISystem:: GenerateRedrawEvent     // Generates a redraw event
    if (HeroBall != null)               // Reflect propoer HeroBall velocity
        SetSliderBarsWithHeroBallVelocity( sliderBarValues )

RedrawRoutine ( )                  // D7:Redraw event service routine
    DrawBalls(AllWorldBalls)
    if (DefiningNewHeroBall)
        DrawBalls(HeroBall)             // Draw the new Hero Ball that is being defined
```

A1：应用程序状态

A2S2：定义定时器事件

D1L3：强行启动一个重新绘制事件

图18-8 基于事件驱动编程模型的编程方案

图18-8展现了三种服务例程的注册（A2）：

- 用户产生的特定于应用程序的事件（S1a）。
- 应用程序定义的事件（S2）。
- GUI系统产生的请求协作的事件（S3c）。

定时器事件定义（A2S2）为应用程序建立一个周期性的警报，以此更新球体自由降落的模拟过程。用户产生的特定于应用程序的事件（D1-D5）的服务例程非常类似于图18-4呈现的控制

驱动解决方案中对应的case语句（B1-B3）。对此不要感到奇怪，因为我们是在同一规范的基础上实现完全相同的用户动作。*LMBDownRoutine*()的第3行（D1L3）表明，如果有必要，应用程序可以请求GUI系统初始化事件。在这种情况下，可以通知GUI系统应用程序需要重新绘制。注意在我们的程序中事件服务例程是很简单的函数。这意味着在D1L3处还可以直接调用*RedrawRoutine*()（D7）。区别在于，调用*RedrawRoutine*()将立即强行启动重新绘制，而请求产生重新绘制事件则允许GUI系统优化重新绘制的次数。例如，如果用户执行LMB单击并立即开始拖动，那么对于D1和D2的实现，GUI系统将在很短的时间内收集到很多*GenerateRedrawEnvent*请求，但是只会产生一个重新绘制事件。按照这种方式可以避免执行一些不必要的重新绘制。

416 ~ 417

为了实现平滑的动画效果，应该执行大约每秒20～40次的更新。由此可知*SimulationUpdateInterval*不应该超过50 ms，以使每秒种调用*ServiceTimer*()例程次数超过20。（请注意重新绘制事件是在*ServiceTimer*()例程结束的时候请求的）。这意味着，应用程序至少能保证每秒钟收到20个重新绘制事件。由于这一原因，D1和D2中的*GenerateRedrawEvent*实际上就不需要了。定时器事件的服务将保证屏幕显示一直都是最新的。

18.2.4　实现注释

事件驱动程序的应用程序状态必须持续整个程序的生命周期。就实现而言，这意味着，应用程序状态必须定义在运行时动态分配的变量基础上，而且要求这些变量驻留在堆内存中。这些变量不同于局部变量，局部变量驻留在栈内存中，持续时间不会跨越不同的函数调用。

将用户操作映射到GUI系统事件经常会导致隐含的（implicit）或者是未定义的事件。在我们的击球程序中，定义HeroBall的动作涉及鼠标左键的按下与拖动。当将这些操作映射到实现中的事件时（见图18-4和图18-8），我们发现还应该注意隐含的鼠标键弹起事件。另一个例子是HeroBall的选择动作：按下鼠标右键。在这种情况下，应用程序没有为鼠标右键的拖动和弹起提供服务，因此（对我们的应用程序而言）这些事件是未定义的。

418

当将用户动作（比如“拖出HeroBall”）映射到一组连续事件（比如，鼠标键按下、拖动、然后弹起）时，可以使用有限状态图来帮助设计解决方案。图18-9描绘了击球程序中定义HeroBall的有限状态图。鼠标左键按下事件使程序进入状态1（State 1）。在图18-8的解决方案中，*LMBDownRoutine*()实现了这一状态，并定义了HeroBall的中心。在这种情况下，状态之间的转移是由鼠标事件触发的。可以看到，在物理上不可能从状态2返回到状态1。但是需要处理从状态1直接转移到状态3的用户操作（按下鼠标键却没有任何拖动动作）。状态图能帮助我们分析可能的状态转移情况，并执行适当的初始化工作。

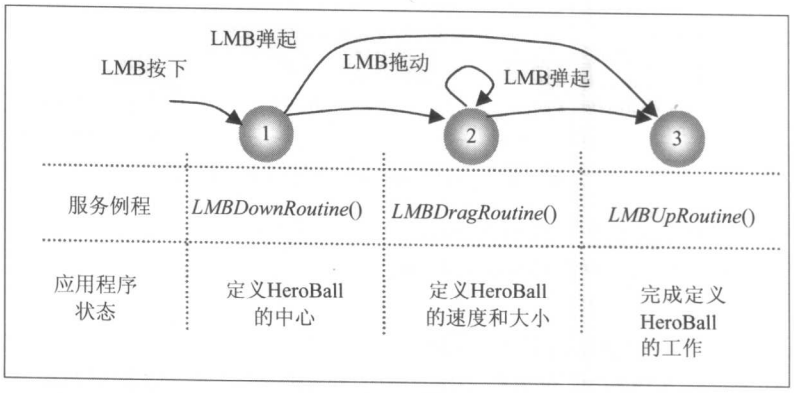


图18-9　定义HeroBall的状态图

事件驱动的应用程序通过物理的（如鼠标的点击）或者是模拟的GUI元素（如退出键、滑动条）与用户对接。GUI输入元素（如退出按钮）是指用户用来改变应用程序状态的人工对象（例如图标）。而GUI输出元素（比如状态条）则是应用程序向用户反馈当前应用程序状态的途径。这两种元素的信息都只沿着一个方向流动——或者从用户到应用程序（输入），或者是从应用程序到用户（输出）。当把GUI元素既当输入又当输出使用时，就需要特别小心。例如，用户在选择或定义一个HeroBall之后，滑动条就会反映自由降落的HeroBall的速度（输出），而与此同时，在任何时候用户都可以操纵滑动条改变HeroBall的速度（输入）。在这种情况下，GUI元素显示的状态与应用程序的内部状态连接在一起。应用程序必须确保这两个状态是一致的。请注意在图18-4展示的解决方案中，没有维护这种状态的一致性。在用户单击RMB（图18-4中的B2）选择HeroBall时，滑动条的值会适当更新。但是当球在重力作用下自由降落时，滑动条的值却没有更新。图18-8呈现的解决方案采用*ServiceTimer*()函数修改了这一问题。

事件服务例程是在程序中定义的函数。在出现相关事件的时候，将从*MainEventLoop*回调这些函数。由于这一原因，这些服务例程也称为**回调函数**（callback function）。应用程序向GUI系统注册回调函数，将函数的地址传递给GUI系统。这就是隐含在图18-7和图18-8中的注册机制。简单的GUI系统（如GLUT或FLTK）一般都支持这种注册机制。这种机制的优势是易于掌握、编程直观，而且占用的程序内存少。这种机制的主要不足是回调函数缺乏组织结构。

在商业GUI系统中有大量应用程序必须处理的事件。采用结构化的方式组织服务例程有助于提高GUI系统的编程性能。现代的商业GUI系统通常都采用面向对象的语言来实现（例如，MFC的C++、Java Swing的Java）。对于这些系统，很多事件服务注册都是作为GUI系统类的子类来实现的，覆盖了相应的虚函数。这样，事件服务例程就是按照GUI元素的功能特性来组织的。在18.4.1节介绍实现细节的时候，将介绍不同注册机制的详细情况。 419

在我们的程序中事件服务例程（或者是回调函数）只是函数。不过，这些函数还担当了一个重要的角色，那就是外部异步事件的服务器。下面是实现事件服务例程时应该考虑的一些准则：

(1) 事件服务例程应该只服务于触发事件，并且立即将控制返回给*MainEventLoop*()。这看起来好像是“没有头脑的人”。然而，由于我们熟悉控制驱动的编程，所以往往会企图在服务例程中使用控制结构来参与或者是登记注册随后的事件。例如，在服务鼠标左键按下事件的时候，我们知道接着将发生鼠标拖动事件。在分配并定义了圆心之后，就具备适当初始化了的数据，可以使用HeroBall对象。可能看起来比较容易的是，简单地采用一个while循环来登记并服务鼠标拖动事件。然而，对于其他可能发生的所有外部事件（比如定时器事件、外部重新绘制事件，等等）而言，这种服务例程内独占式的控制，不仅是一种不好的设计决定，而且还可能导致程序故障。

(2) 事件服务例程应该是**无状态的**（stateless），且调用之间应该是独立的。就实现而言，这实际上意味着事件服务例程不应该定义局部**静态变量**，以此来记录前一次调用的数据。因为无法控制事件在什么时候触发、触发多少次，当把这些变量当作数据或者是改变应用程序状态的条件时，很容易导致不必要的灾难性的解决方案复杂化。可以在应用程序状态中定义外部状态变量，记录必须持续多个事件服务的临时状态信息。图18-8中的*DefiningNewHeroBall*标记就是这样的例子。

(3) 不管常规的逻辑顺序如何，事件服务例程都应该检查调用条件。例如，虽然鼠标按下事件未发生时逻辑上不会发生鼠标拖动事件，但是实际上用户可能在应用程序窗口外面按下鼠标键，然后将鼠标拖到应用程序窗口内。在这种情况下，我们将接收到鼠标拖动事件而没有相应的鼠标按下事件。为此，鼠标拖动服务例程应该检查调用条件，确保适当的初始化工作已经完成。 420

注意，在图18-8中没有包含适当的调用条件检查。例如，*LMBDragRoutine*()没有验证已经调用了*LMBDownRoutine*()（即检验*DefiningNewHeroBall*标记）。在实际系统中，这可能导致程序故障或者是崩溃。

18.2.5　小结

本节讨论了编程模型或者是程序语句的组织策略。交互式应用程序不断地等待用户输入动作并做出反应，如果将程序语句的组织建立在设计控制结构的基础上，结果将使程序复杂效率低下。现有GUI系统分析所有可能的用户动作，设计与用户交互的控制结构，实现所有用户动作的默认行为，并在GUI的API中提供这种功能特性。开发交互式应用程序可以利用GUI API中已有的控制结构（例如*MainEventLoop*()），并（通过事件服务例程）修改用户动作的默认行为。为了与现有GUI API正确协同，组织程序语句的策略是，规定改变应用程序状态的用户动作。

我们已经掌握了如何组织程序的语句，下面就可以分析解决方案功能模块的组织策略了。

18.3　模型-视图-控制器体系结构

18.2.3节和图18-8所介绍的事件驱动击球程序，阐述了与用户交互时的编程能力和效率问题。在推导该模型的过程中，我们掩盖了解决方案所需要的很多支持函数（例如*UpdateSimulation*()）。本节将开发组织这些函数的策略。注意这里所关注的并不是这些函数的实现细节，而是将相关的
421 函数组织成组件。然后关注不同的组件如何协作支持应用程序的功能。按照这种方式，可以导出一个适合于实现一般交互式图形应用程序的框架。使用适当的框架来引导设计和实现，将使开发的程序更易于理解、维护、修改和扩充。

18.3.1　模型-视图-控制器框架

按照18.2节解决方案的开发经验，可以将交互式图形应用程序（interactive graphics application）看作是允许用户交互式地更新应用其内部状态的程序。这些应用程序使用计算机图形（比如圆）将内部状态（比如自由降落的球）实时可视化。模型-视图-控制器（modelview-controller）框架为这类应用程序的讨论提供了一个方便的结构。在MVC框架中，模型（model）就是应用程序的状态，视图（view）负责将模型呈现给用户，控制器（controller）则负责用户与模型的交互。在该框架内，图18-8的解决方案仅仅实现了一个控制器。本节将介绍MVC框架的另外两个组件，以及这些组件如何协同支持交互式图形应用程序。

图18-10展示了一个MVC框架的细节。该框架描述了一个常见的交互式图形应用程序的行为。击球程序将继续作为演示组件细节的实例。右上矩形框内是模型，右下的为视图。左边的矩形框就是控制器组件。这三个矩形框代表了应用程序开发人员必须开发的程序代码。两个虚线框分别代表外部图形API和GUI API。他们是构建系统时将要使用的外部库（external library）。比较流行的图形API有OpenGL、Microsoft Direct3D(D3D)、Java 3D，等等。18.2.2节提到了，流行的GUI API有GLUT、FLTK、MFC和Java Swing Library。

模型组件定义持久稳固的应用程序状态（例如AllWorldBalls、HeroBall等），并且实现应用程序状态的接口函数（例如*UpdateSimulation*()）。因为讨论的是“图形”应用程序，所以期望图形原语（例如CirclePrimitive）成为表示应用程序状态的组成部分。图18-10呈现了这一事实，其中应用程序状态（椭圆）部分覆盖了虚线框Graphics API。本节的剩余部分将交替使用术语模型和
422 持久稳固的应用程序状态。

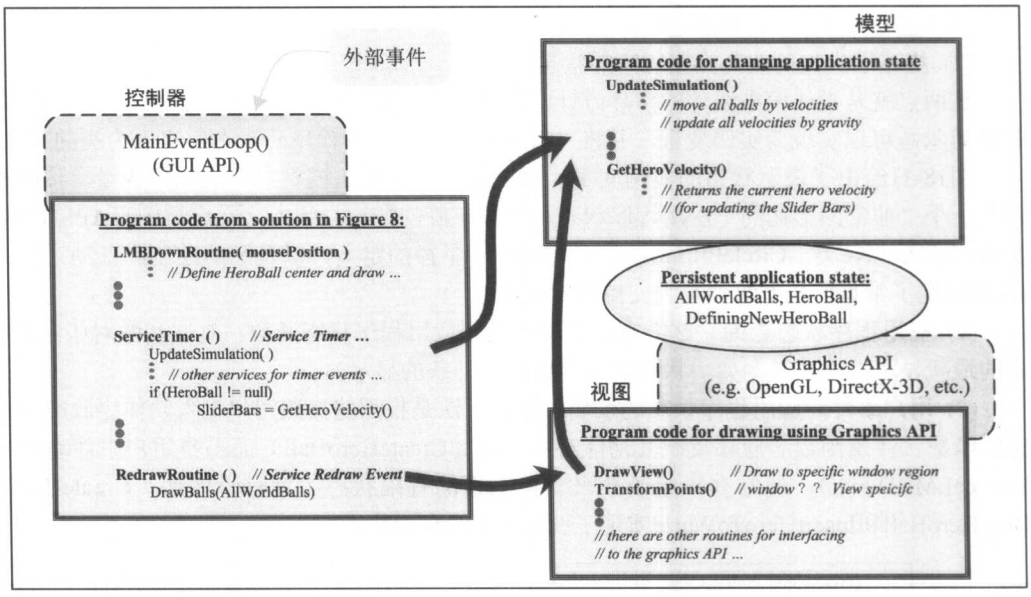


图18-10 交互式图形应用程序的组件

视图组件掌管应用程序窗口中绘图区域（drawing area）的绘图工作（drawing）（例如，绘制自由降落的球）。更具体地讲就是，视图组件负责初始化图形API变换，从而使模型的图形原语的绘制结果出现在适当的绘图区域。从视图指向模型组件的箭头表示必须由模型组件重新绘制应用程序的实际状态。只有模型组件知道整个应用程序状态的详细情况（比如，自由降落的圆的大小和位置），因此只有模型组件能够重新绘制整个应用程序。视图组件还负责将用户点击的鼠标位置变换到模型所能够理解的坐标系中（例如，单击鼠标拖出一个hero球）。

图18-10中左上的外部事件箭头表明，所有外部事件都是由*MainEventLoop*()处理的。相关事件将被转发给控制器组件中的事件服务例程。由于控制器组件负责与用户的交互，所以它的设计通常都是建立在事件驱动编程技术的基础上。18.2.3节和图18-8所介绍的解决方案就是实现控制器组件的例子。从控制器到模型的箭头指出大部分外部事件实际上都改变模型组件（比如，创建一个新的HeroBall，或者是更改当前HeroBall的速度）。从控制器到视图组件的箭头表明用户输入点的变换是由视图组件处理的。控制器通常返回的是设备坐标系中鼠标的点击位置，设备坐标系的原点在左上角。在应用程序模型中，使用原点在左下角的坐标系将更为方便。视图组件具备变换功能，能够执行必要的变换。 423

由于模型必须理解视图设置的变换，所以在同样的图形API基础上实现模型和视图组件就显得很重要。但是，这种底层支持API的共享并不意味着模型和视图是集成在一起的组件。与此相反，在后续章节的讨论中将看到，清楚地区分这两个组件并在它们之间建立良定义的接口，将是有利的。

18.3.2 将 MVC 应用于击球程序

有了所描述的MVC框架，并掌握了各个组件的职责，就可以扩展图18-8中的解决方案，并完成击球程序的设计。

1. 模型

模型就是应用程序的状态，因此也是应用程序的核心。在18.2.3节中介绍事件驱动程序的设计方法时，提到了如下两点：

(1) 定义应用程序的状态。

(2) 描述用户如何改变应用程序的状态。

这两点就是设计模型组件的指导方针。在面向对象的环境中，模型组件可以实现为类，应用程序的状态可以实现为实例变量，并将“用户如何改变应用程序的状态”实现为类的方法。

图18-11给出了表示状态的实例变量。这些变量通常都是模型组件的私有变量。按照预期，将有一个“非常图形化的”应用程序状态。为了正确支持这一状态，在底层图形API的基础上定义CirclePrimitive类。CirclePrimitive类支持圆心、半径的定义，以及圆的绘制和移动，等等。图
424 18-11展现了常见的模型组件必须支持的4类方法：

(1) **应用程序状态查询**。这类方法是那些返回应用程序状态函数。这些函数对于维护GUI元素的最新显示很重要（例如，状态显示和速度滑动条的显示）。

(2) **用户事件对应用程序状态的改变**。这些方法是指那些按照用户输入动作更改应用程序状态的函数。注意函数名应该反映其功能特性（例如CreateHeroBall）而不是用户事件动作（比如
425 ServiceLMBDown）。通常有一组函数来支持确定的有限状态的转移。例如，CreateHeroBall、DragHeroBall和InsertHeroToWorld实现了图18-9中的有限状态图。

```
class ApplicationModel    {                                          建立在图形API上的类
    private:
    // Application's private state
        vector<CirclePrimitive>   AllWorldBalls      // World balls, initially empty
        CirclePrimitive           HeroBall           // The Hero Ball
        bool                      DefiningNewHero    // If LMB drag is true

    public:                                                          1. 应用程序状态查询
        bool  IsDefinningHeroBall()
              // If LMB drag is true, same as if we are in
              // the middle of  defiing the a new hero ball
        bool  HeroBallExists()
              // Current hero ball is not null
        int   NumBallsOnScreen()
              // Number of balls currently on screen
        float HeroVelocityX()                                        2. 用户事件对应用程序状态的改变
              // Hero's velocity, x-component
        float HeroVelocityY()
              // Hero's velocity, y-component

        void  CreateHeroBall (mousePosition)
              // Creates new hero ball, with center at mousePosition
              // radius and velocity are initialized to zero
        void  DragHeroBallTo (mousePosition)
              // Refine radius and velocity of hero ball based on
              // hero's center and current mousePosition
        void  SetHeroBallVelocity (velocityX, velocityY)             3. 应用程序事件对应用程序状态的改变
              // Sets current hero ball velocity
              // if there is no current hero ball, nothing happens
        void  InsertHeroToAllWorld( )
              // Done defining HeroBall, insert into WorldBallSet
        void  SelectHeroBall (mousePosition)
              // Sets hero ball to be the one currently under
              // mousePosition sets to null if none exists
        void  UpdateSimulation( )                                    4. 应用程序状态的可视化
              // Move balls by their velocities, update velocity
              // by gravity and remove off-screen ones
        void  RedrawApplicationState( )
              // Draw all the freefalling balls (including the HeroBall)
              // to the desired region on the application window.
}
```

图18-11　击球程序的模型组件

(3) **应用程序（定时器）事件对应用程序状态的改变**。这是一个更新应用程序状态的函数，

其状态通常来源于有目的的、同步应用程序的定时器事件。对于击球程序而言，需要更新所有球的速度，按照更新的速度替换球的位置并计算球之间的碰撞，同时清除移到屏幕外面的球。

(4) **应用程序状态的可视化**。这是一个知道如何绘制应用程序状态的函数（比如在相应的位置上绘制必要数目的圆）。视图组件将在应用程序窗口上初始化一定的区域，建立变换，并调用该函数将结果绘制到初始化了的区域中。

重要的是记住用户的异步事件是在两个同步的应用程序定时器事件之间达到的。实际上，用户观察即时的应用程序状态（应用程序窗口中的图形），并产生异步事件，改变下一个模拟周期的应用程序状态。例如，用户看到HeroBall将要与另一个球相撞，而更改HeroBall的速度以避免在下一个模拟周期中可能发生的碰撞。这意味着，必须在同步定时器更新前，确保所有现存的异步用户事件都被处理了。此外，应用程序还应该提供持续的反馈，以确保用户观察到最新的应用程序状态。对于简单的、类似于击球程序的单用户应用程序，这种微妙的事件达到时间和处理顺序并不是一个问题。而在大规模的网络化的多用户交互系统中，输入事件和输出显示的延时可能很明显，因此*UpdateSimulation*()函数通常被分割成更新前、更新中和更新后三部分。

2. 视图

图18-12展现了ApplicationView类。该类支持两个主要的视图组件功能：坐标空间变换和重新绘制的初始化。在前面的讨论中提到，控制器负责调用*DeviceToWorldXform*()，将用户输入点传递给模型组件。Viewport（视点）类用来封装高度依赖于API的设备初始化以及变换过程。 426

```
class Viewport {
        private:
            // An area on application window for drawing.
            // Actual implemenation of the viewport is GraphicsAPI dependent.
        public:
            void        EraseViewport()
                          // Erase the area on the application window
            void        ActivateViewportForDrawing()
                          // All subsequent Graphics API draw commands
                          //  will show up on this viewport
}
class ApplicationView {
        private:
            // a view's private state information
            Viewport    TargetDrawArea
                          //  An area of the application main window that
                          // this view will be drawing to
        public:
            void        DeviceToWorldXform( inputDevicePoint, outputModelPoint)
                          // transform the input device coordinate point to
                          // output point in a coordinate system that the model understands

            void        DrawView( ApplicationModel TheModel)
                          // Erase and activate the TargetDrawArea and then
                          // Sets up transformation for TheModel
                          // calls TheModel.DrawApplicationState() to draw all the balls.
    }
```

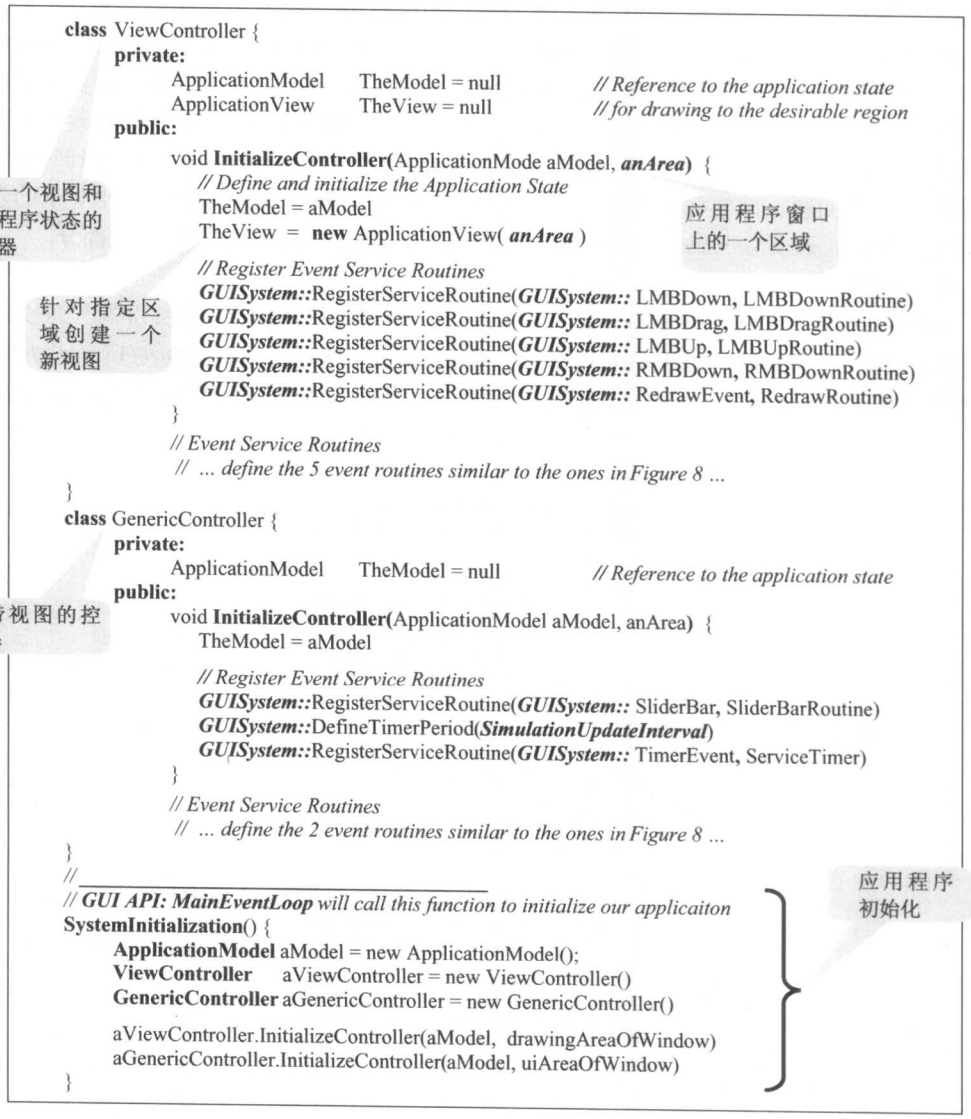
图18-12 击球程序的视图组件

3. 控制器

可以改进图18-8的解决方案，以便更好地支持击球程序的专用功能。在图18-2中描绘的应用程序窗口有两个独特的解释事件的区域：上面的应用程序绘制区，其中鼠标按键事件与HeroBall的定义/选择关联起来了；下面的GUI元素区，该区域不同GUI元素的鼠标按键事件有不同的含义（例如，滑动条上的鼠标按键事件将产生SliderBarChange事件，等等）。另外，上面的应用程序绘

制区也就是ApplicationView必须控制的ApplicationModel状态的绘制区。

图18-13引入了两种控制器类：ViewController和GenericController。每个控制器类都致力于接收应用程序窗口相应区域的输入事件。ViewController在初始化期间创建一个ApplicationView，这样该视图就被和控制器绑定在一起，以便在同一区域绘制ApplicationModel状态。此外，ViewController类还定义了合适的鼠标事件服务例程，以便支持与HeroBall的交互。
427 GenericController主要用来包含与应用程序状态进行交互的GUI元素。




```
class ViewController {
    private:
        ApplicationModel    TheModel = null          // Reference to the application state
        ApplicationView     TheView = null           // for drawing to the desirable region
    public:
        void InitializeController(ApplicationMode aModel, anArea) {
            // Define and initialize the Application State
            TheModel = aModel
            TheView = new ApplicationView( anArea )

            // Register Event Service Routines
            GUISystem::RegisterServiceRoutine(GUISystem:: LMBDown, LMBDownRoutine)
            GUISystem::RegisterServiceRoutine(GUISystem:: LMBDrag, LMBDragRoutine)
            GUISystem::RegisterServiceRoutine(GUISystem:: LMBUp, LMBUpRoutine)
            GUISystem::RegisterServiceRoutine(GUISystem:: RMBDown, RMBDownRoutine)
            GUISystem::RegisterServiceRoutine(GUISystem:: RedrawEvent, RedrawRoutine)
        }
        // Event Service Routines
        // ... define the 5 event routines similar to the ones in Figure 8 ...
}
class GenericController {
    private:
        ApplicationModel    TheModel = null          // Reference to the application state
    public:
        void InitializeController(ApplicationModel aModel, anArea) {
            TheModel = aModel

            // Register Event Service Routines
            GUISystem::RegisterServiceRoutine(GUISystem:: SliderBar, SliderBarRoutine)
            GUISystem::DefineTimerPeriod(SimulationUpdateInterval)
            GUISystem::RegisterServiceRoutine(GUISystem:: TimerEvent, ServiceTimer)
        }
        // Event Service Routines
        // ... define the 2 event routines similar to the ones in Figure 8 ...
}
//_____________________________________
// GUI API: MainEventLoop will call this function to initialize our applicaiton
SystemInitialization() {
    ApplicationModel aModel = new ApplicationModel();
    ViewController    aViewController = new ViewController()
    GenericController aGenericController = new GenericController()

    aViewController.InitializeController(aModel, drawingAreaOfWindow)
    aGenericController.InitializeController(aModel, uiAreaOfWindow)
}
```

图18-13　击球程序的控制器组件

图18-13的底部阐明了GUI API MainEventLoop仍将调用*SystemInitialization*()函数初始化应用程序。在这种情况下，创建了一个ViewController实例和一个GenericController实例。ViewController被初始化为监控应用程序窗口绘图区域的鼠标按键事件（例如，单击LMB定义HeroBall），而GenericController则被初始化为监控GUI元素状态的改变（例如，LMB拖动滑动条）。注意，定时器事件的服务是全局性的，在整个应用程序范围内都有效，应该只在一个控制器中定义（任何一个都行）。

实际上，GUI API MainEventLoop会根据事件的上下文将事件分派给各个控制器。事件上下文通常是按照鼠标指针位置或者是当前GUI元素的焦点来定义的（即，哪个元素是活跃的）。应用程序负责为每一个将要从GUI API直接接收事件的窗口区域创建一个控制器。 428

18.3.3 利用 MVC 扩充击球程序

18.3.2节介绍的解决方案有一个有趣的特性，那就是模型组件不知道视图或控制器组件的任何信息。这种干净的接口允许我们通过插入额外的视图/控制器对来扩充解决方案。

例如，图18-14展现了图18-2中给出的击球程序的一种扩充。在UI（用户界面）区域紧邻退出（quit）按钮的地方有一个小视图。小视图与原始大小的视图完全一样，除了其所覆盖的应用程序窗口区域更小之外。

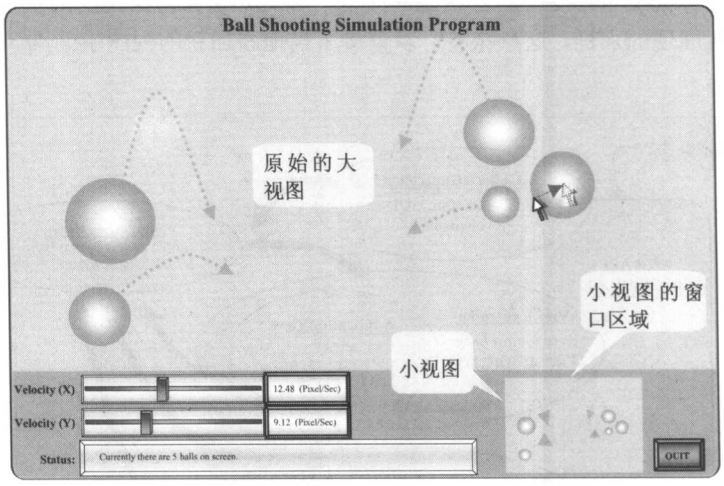


图18-14 拥有大视图和小视图的击球程序

图18-15展现了使用MVC设计方案实现小视图的过程，即针对所期望应用程序窗口区域创建一个新的ViewController实例（ViewController将创建一个附加的ApplicationView）。注意，GenericController的窗口区域实际上包含了这个小的ViewController的区域。当在该区域触发用户事件时，“顶层”控制器（可见的那个）将接收到事件。初始化之后，新的小视图在行为方式上将与原始大小的视图完全一样。 429

```
//_____________________________
// GUI API: MainEventLoop will call this function to initialize our applicaiton
SystemInitialization() {
    ApplicationModel   aModel = new ApplicationModel();
    ViewController     aLargeViewController = new ViewController()
    GenericController  aGenericController = new GenericController()

    aLargeViewController.InitializeController(aModel, drawingAreaOfWindow)
    aGenericController.InitializeController(aModel, uiAreaOfWindow)

    ViewController aSmallViewController = new ViewController()
    aSmallViewController.InitializeController(aModel, smallViewDrawingArea)
}
```

新的ViewController实例（以及ApplicationView实例）

图18-15 击球程序中小视图的实现

为了简化，图18-14显示了两个同样的视图/控制器对。一般，创建新的视图/控制器对是为了呈现应用程序状态不同的可视化效果。例如，稍微修改一下视图组件的变换功能，图18-14中的

大视图可以配置为缩放视图（zoom view），而将小视图配置为工作视图（work view）。缩放视图可以缩放不同的区域（比如HeroBall周围的区域），工作视图可以呈现整个应用程序空间（例如所有自由降落的球）。

图18-16展现了图18-15中的解决方案的组件，以及这些组件的交互作用。可以看到模型组件支持所有视图和控制器组件的操作，但是它并不了解这些组件。这种独特的简单接口具有下列优势：

(1) 简单。模型组件是应用程序的核心，通常也是最复杂的组件。让该组件的设计独立于任何特定的控制器（用户输入/事件）或视图（具体的绘制区域），可以避免一些不必要的麻烦。

(2) 可移植性。控制器组件通常将用户动作翻译成具体模型函数的调用。这种翻译功能的实现一般比较简单，并且特定于底层GUI API。让模型脱离高度依赖于API的控制器，有利于解决方案向其他GUI平台的移植。

430 (3) 可扩充性。模型组件支持内部状态的改变，并且知道如何绘制内部状态的内容。就像图18-15和图18-16所展现的那样，这意味着可以直接了当地添加新的视图/控制器对，增加应用程序的交互活动。

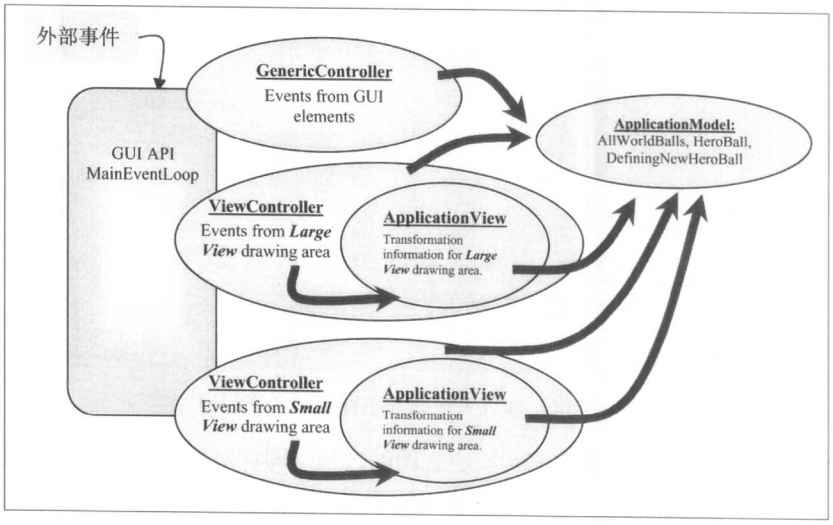


图18-16　带有小视图的击球程序的组件

18.3.4　MVC 组件之间的交互作用

MVC框架是一个描述一般交互式系统的工具。该框架的一个优势就是能够直接了当地支持多个视图/控制器对。每个视图/控制器对都以同样的方式分担各自的职责：视图呈现模型，控制器允许事件（用户产生的事件或者是其他事件）更改模型组件。

对于类似于图18-16所描绘的、拥有多个视图/控制器对的应用程序，用户可以通过三个控制器中的任何一个更改模型组件。此外，应用程序本身也能够改变模型状态。但是所有的组件都必须确保呈现给用户的内容是一致的最新的。例如，在用户拖出一个新的HeroBall的时候，大视图组件和小视图组件都必须显示该球的拖动过程，而GenericController组件则必须确保滑动条正确地显示这个暗中定义的HeroBall的速度。在传统的MVC模型中，组件之间的一致性是通过精心设
431 计的协议来维持的（比如，通过观察者设计模式来维护）。尽管传统的MVC模型工作得很好，但是精心设计的协议要求所有组件都可以通信，或者是在模型组件中跟踪所有变动。

在我们的例子，以及大多数现代的交互式图形系统中，都是应用程序定义定时器事件来进行模拟计算。为了获得平滑的模拟结果，一般在实时响应的阈值（比如20～50毫秒）内触发定时器

事件。在服务定时器事件时，应用程序可以抓住机会维护所有组件之间状态的一致性。例如，在图18-18的*ServiceTimer*()函数中，可以根据当前HeroBall的速度来更新速度滑动条。实际上，在每个定时器事件的服务期间，应用程序都将最新的模型信息推送给所有组件，并强制各组件刷新向用户呈现的内容。在这种方式下，组件之间的通信协议就不是那么重要了。所有组件都引用模型，应用程序中的每个视图/控制器对都不需要了解别的视图/控制器对的情况。在两个周期性的定时器事件之间，用户的异步事件可以改变模型。这些改变只在模型组件内部进行，应用程序中的其他组件不需要知道这些更改。在周期性的定时器服务期间，除了计算模型的模拟更新外，所有组件都会调查模型获取最新的状态信息。例如，在用户单击鼠标左键并按下拖动的时候，将在模型组件中定义一个新的HeroBall。在此期间，大视图组件和小视图组件都不会显示新的HeroBall，速度滑动条也不会显示新HeroBall的速度。这些组件将只在应用程序定时器事件的服务期间获取并显示最新的HeroBall的信息。由于定时器事件每秒触发30多次，所以用户在所有组件中都将一直观察到平滑的最新的应用程序状态。

18.3.5 应用 MVC 概念

MVC框架适合于一般的交互式系统。就像本节所介绍的那样，使用MVC框架的交互式系统能够清晰地定义组件的行为。此外，在组件之间有了明确定义的接口之后，使用额外的视图/控制器对来扩充系统就变得比较简单了。

交互式系统并不一定要是一个精心设计的应用软件。例如，滑动条就是一个功能完备的交互式系统。模型组件包含当前值（通常是一个浮点数），视图组件将该值呈现给用户，而控制器则允许交互式地更改该值。常见的视图组件通常画一个矩形图标（横条和滑块）表示模型组件中的当前值，而控制器组件一般支持鼠标的按下和拖动事件，以便交互式地更改模型组件中的值。有了这个概念，使用额外的视图/控制器对来扩充系统就比较简单了。例如，在我们的击球程序中，滑动条有一个附加的视图控件来显示模型的数值。在这种情况下，没有为这个数值视图定义互补的控制器组件。如果有互补的控制器组件，用户将能够键入数值。

432

18.4 实现示例

图18-17展现了18.3.3节所介绍的解决方案的两种实现。左边的版本是基于OpenGL和FLTK的，右边的版本则基于D3D和MFC。本节将介绍这两种实现的详细情况。所要学习的经验教训是：（a）一个合适的MVC解决方案框架应该独立于任何一种实现；（b）一种设计好的实现应该在任何合适的API上都是可实现的，或者是易于移植到任何合适的API上。

在分析每种实现的详细情况之前，需要掌握一些现代的GUI API和图像API的使用。

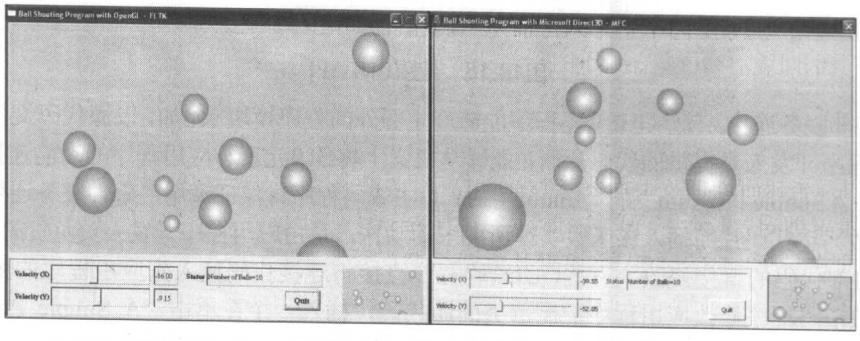


图18-17 分别使用OpenGL+FLTK和D3D+MFC的击球程序

18.4.1 使用 GUI API

建立应用程序的图形用户界面（GUI）涉及两个截然不同的步骤。第一个步骤是设计用户界
433 面系统的布局。在这个步骤中，应用程序开发人员在代表应用程序窗口的区域中放置GUI元素（比如，按钮、滑动条等）。GUI元素通常是二维图形对象（artifact）（例如，具有三维效果的表示滑动条的图标）。这一步骤的目标是安排好这些对象，达到用户友好性和最大的可用性（例如，滑动条的位置、颜色和大小应该怎么样才最好，等等）。建立应用程序GUI的第二个步骤是在语义上将GUI元素连接到应用程序的功能上（例如，在拖动滑动条的时候更新HeroBall的速度）。在这个步骤中，应用程序开发人员需要编写所需功能的代码（例如，改变HeroBall速度的代码）并向屏幕上的图形对象（比如滑动条）登记这一代码。这正好是18.2.2节所描述的事件服务注册。

现代的GUI API支持使用GUI编码程序（builder）建立图形用户界面。GUI 编码程序是一种交互式的图形编辑器，使用户能够交互式地放置并操作GUI元素的外观。此外，GUI编码程序还能够帮助应用程序开发人员合成或者是产生服务例程，并将那些服务例程连接到GUI元素所产生的事件上。

图18-18阐述了GUI编码程序将前端图形用户界面（图的左边）连接到用户开发的程序代码（图的右边）的机制。在图18-18的中间显示的是GUI Builder，即带花纹的椭圆。指向左边应用程序
434 （A Simple Program）的箭头表明，应用程序开发人员使用GUI Builder设计了该应用程序的布局（比如按钮或状态显示区域的位置）。从GUI Bulider指向MainEventLoop和Event ServiceLinkage模块的箭头表明：GUI编码程序能够产生程序设计代码，登记事件服务。在图18-18中，有两条虚的连接线，将鼠标和GUI按钮元素通过MainEventLoop模块与事件服务连接模块以及应用程序控制器模块连接起来了。这两个连接分别代表了两种不同的使用GUI API支持事件服务的机制：

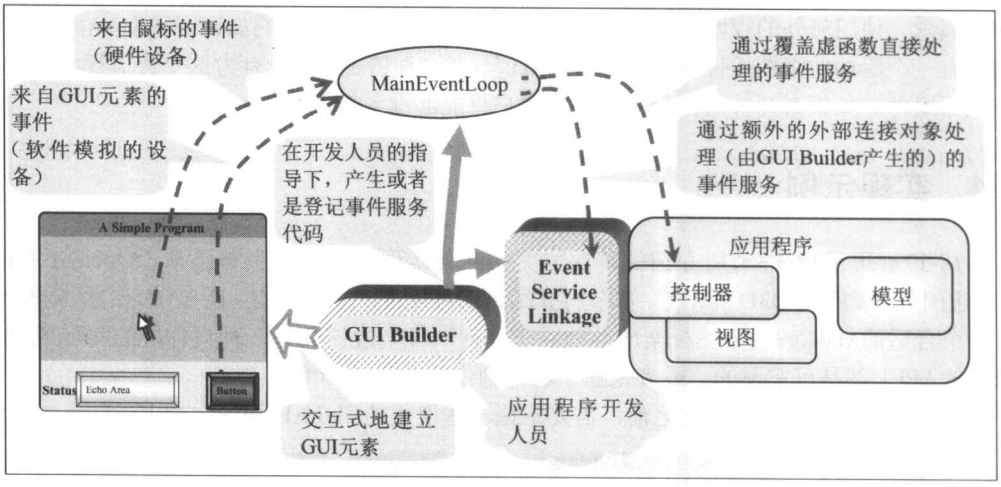


图18-18　使用GUI API

(1) 外部服务连接。有些GUI编码程序能够产生额外的程序模块（比如，以源代码文件的形式），使用应用程序开发人员提供的代码片断，能够从语义上将GUI元素与应用程序的功能连接起来。例如，单击“A Simple Program”的“Button”时，GUI编码程序将保证调用“事件服务连接”模块中的一个函数。将代码片断插入该函数实现所要求的动作，是由应用程序开发人员负责的。

(2) 内部代码的直接修改。有些GUI编码程序直接将连接代码插入到应用程序的源代码中。例如，GUI编码程序修改应用程序控制器类的源代码，插入一个在单击“A Simple Program”的“按钮”时将要调用的新函数。注意，GUI编码程序只是插入一个空函数；应用程序开发人员仍

然需要填写这个新函数的具体细节。

外部服务连接机制的优势在于，GUI编码程序只需要了解很少的应用程序源代码。这提供了一个简单而灵活的开发环境，开发人员能够以任何合适的方式，自由组织源代码的结构、变量名，等等。然而，外部产生的程序设计模块意味着松散的集成环境。例如，为了修改“A Simple Program”的“button”行为，应用程序开发人员必须调用GUI编码程序，修改代码片断，并重新生成外部程序模块。与此相反，内部代码的直接修改机制提供了更好的集成环境。在该环境下，GUI编码程序直接修改应用程序的源代码。然而，为了支持适当的“直接代码修改”，GUI编码程序必须熟悉应用程序的源代码系统（比如，源代码的组织方式、文件名、变量名，等等），而且还经常对其施加严格的约束。

18.4.2 使用图形API

图18-19阐述了一种理解现代图形API的方式，那就是将API看作底层图形硬件的函数接口。 435
比较方便的是，可以认为该函数接口包含两个阶段：图形硬件环境（Graphics Hardware Context，GHC）和图形设备环境（Graphics Device Context，GDC）。

1．**图形硬件环境（GHC）**。图18-19右边的垂直椭圆描绘了这一阶段。可以将GHC看作一种配置，该配置包装了硬件视频显示卡。应用程序为每一个唯一的硬件视频卡配置（比如，帧缓冲区或者是z缓冲区的深度，等等）创建一个GHC。很多图形设备环境（如下所示）都可以连接到每个GHC，以便支持同一应用程序多个屏幕区域的绘图。

2．**图形设备环境（GDC）**。在图18-19中这一阶段被描绘为圆柱形管道（pipe）。图中的多个管道表明应用程序可以创建多个连接到同一GHC的GDC。通过每个GDC，应用程序可以绘制应用程序窗口中截然不同的区域。为了正确支持这一功能特性，每个GDC都表示一个完整的绘制状态（rendering state）。绘制状态包含影响图像最终外观的所有信息。其中包括原语属性、照明参数、坐标变换，等等。原始属性有颜色、大小、模式等等。照明参数包括灯光位置、灯光颜色、表面材质性质等。图形API通常使用一系列二个或者三个的矩阵处理器来支持坐标变换。在图 436
18-19中GDC管道内部的“M”框就是矩阵处理器。每个矩阵处理器都有一个变换矩阵，通过该矩阵对输入向量进行变换。由于这些处理器是串联操作的，所以联合起来能够实现多级坐标空间变换（例如，对象坐标到世界坐标，世界坐标到视觉坐标系，视觉坐标系到投影空间）。应用程序必须给这些矩阵处理器加载适当的矩阵，实现所期望的变换。

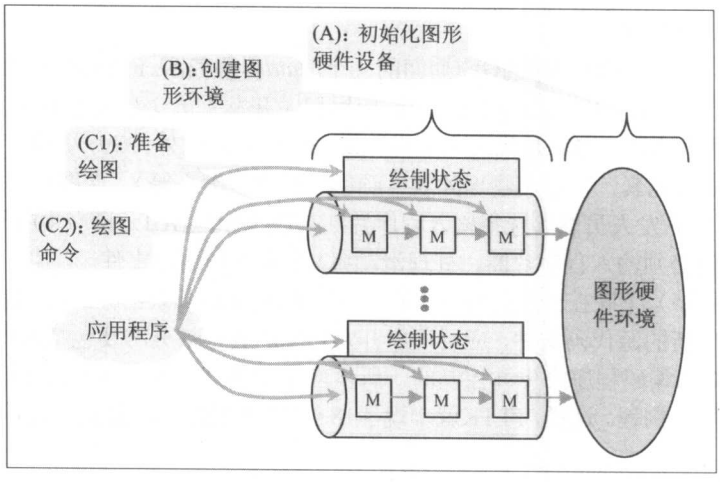


图18-19 使用图形API

了解了这一情况，就可以明白图18-19所阐述的内容。为了使用图形API，应用程序将：

（A）初始化一个或多个GHC。每个GHC代表一种独特的图形视频卡配置。一般的情况是，初始化并配置一个GHC，然后在整个应用程序内共享。

（B）创建一个或多个GDC。每个GDC都支持应用程序窗口中独特区域的绘制。例如，应用程序可能为应用程序内的每个视图组件创建一个GDC。

（C）使用GDC绘制。应用程序通过相应的GDC实现特定窗口区域的绘制。在图18-19中，应用程序设置绘制状态（C1），然后向GDC发布绘图命令（C2）。绘制状态的设置包括所有相关原语和照明属性的设置、适当的变换矩阵的计算，以及变换矩阵到矩阵处理器的加载。绘图命令通常是一系列顶点位置，以及解释这些顶点的指令（例如，两个顶点位置和一条指令，指出这两个点是一条直线的端点）。

实际上，现代的图形API是高度可配置的，并且支持很多抽象的编程模式。例如，Microsoft的Direct3D支持一种绘图模型，可以完全绕过矩阵处理器（比如，在顶点经过预先变换的时候）。

18.4.3　实现细节

图18-20展现了18.3.3[1]节所介绍的解决方案的具体实现的设计情况。此处，MainUIWindow表示整个击球程序。该对象包含GUI元素（滑动条、退出按钮，等等）、模型（应用程序的状态）
437 以及两对视图/控制器实例（分别针对LargeView和SmallView）。

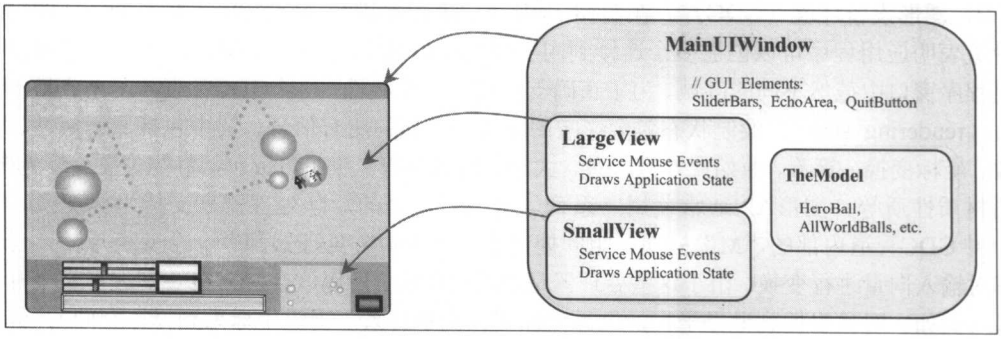


图18-20　双视图击球程序的实现

1. OpenGL和FLTK

图18-21给出了构建击球程序的GUI期间的一个*Fluid*屏幕截图。Fliud是FLTK的GUI编码程序。在图18-21的右下角，可以看到（A）Fluid允许应用程序开发人员交互式地放置以图形表示的GUI元素（三维外观的图标）；（B）是代表应用程序窗口的区域；此外（C），应用程序开发人员还可以交互式地选中GUI元素，指定其物理外观（颜色、形状、大小等）。在图18-21的左下角可以看到，（D）应用程序开发人员能够选择输入程序片断，服务相应GUI元素产生的事件。在本例中，可以看到开发人员必须输入程序片断，处理滑动条X方向上的速度事件。注意，这一程序片断与Fluid（GUI编码程序）关联在一起，独立于程序源代码系统的其余部分。就GUI布局设计总的情
438 况而言，Fluid产生新的源代码文件，应用程序开发环境的其余部分将要包括这些文件。因为这些源代码文件是由GUI编码程序产生并控制的，所以应用程序开发人员必须调用GUI编码程序才能更新或维护事件服务例程。这样，FLTK就实现了18.4.1节介绍的外部服务连接。在我们的实现中，

1. 本节源代码的网址为http://faculty.washington.edu/ksung/fcg2/ball.tar.zip。

我们指示Fluid创建了一个UserInterface类（.h和.cpp文件），然后将该类与应用程序开发环境的其余部分集成。

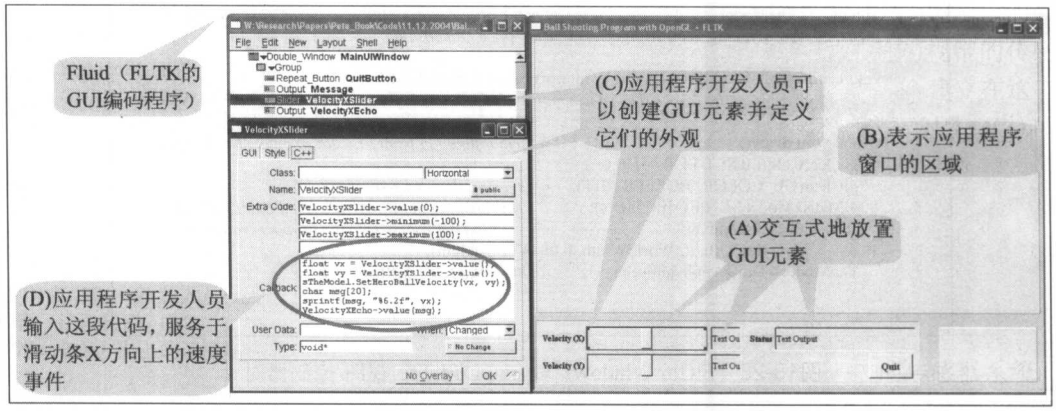


图18-21 Fluid：FLTK的GUI编码程序

图18-22展现了使用OpenGL和FLTK实现的MainUIWindow。这里，图形操作是通过OpenGL执行的，用户界面操作则由FLTK支持。按照介绍，MainUIWindow中的UserInterface对象由Fluid创建，用来服务GUI事件。TheModel就是图18-11描述的应用程序状态。两个FlGlutWindow对象建立在预先定义好的FLTK类的基础上，那是为了专门支持OpenGL绘图而设计的。MainUIWindow的构造器表明鼠标事件服务是通过回调机制注册的。按照18.2.4节的讨论，FLTK（Fast Light Toolkit）是一种轻量级的GUI API。在此，我们看到了使用回调注册机制接收用户事件的例子。

```
// Forward declaration of mouse event service routines
    void ServiceMouse(int button, int state, int x, int y); // service mouse button click
    void ServiceActiveMouse(int x, int y);                   // service mouse drag
class MainUIWindow {
    UserInterface       UI;          // This is Linkage Code generated by Fluid (GUI Builder)
                                     //      This object services events geneated by GUI elements
    Model               *TheModel;   // The application State (Figure 11)
    FlGlutWindow        *LargeView;  // These are View/Controller pairs that understand graphics
    FlGlutWindow        *SmallView;  //      outputs (GDC) and mouse events (controller)

    MainUIWindow(Model *m)    {                          // The constructor
        TheModel = m;                                    // Sets the model ...
        LargeView = new FlGlutWindow(TheModel);          // Create LargeView
        LargeView->mouse = ServiceMouse;          // callback functions for service mouse events
        LargeView->motion = ServiceActiveMouse;
        // Create SmallView ... exactly the same as LargeView (not shown)
        glutTimeFunc( // set up timer and services )    // Set up timer ...
    }
};
```

图18-22 基于OpenGL和FLTK的MainUIWindow

FlGlutWindow是FLTK预先定义的Fl_Glut_Window类对象（见图18-23）。该类专门用来支持OpenGL绘图。每个FlGlutWindow对象实例都包含一个控制器（比如，接收鼠标事件）和一个图形设备环境（GDC）。从中可以看到在调用TheModel重新绘制应用程序状态之前，*draw*()函数先设置了绘制状态（比如，清除颜色和矩阵的值），其中包括矩阵处理器的计算和编程（例如，GL_PROJECTION）。

```
// Fl_Glut_Window is a pure virtual class supplied by FLTK specifically for supporting
// windows with OpenGL output and for receiving mouse events.
class FlGlutWindow : public Fl_Glut_Window {
  FlGlutWindow(Model *m);              // Constructor
  Model    *TheModel;                  // The application state: initialized during construction time.
  float    WorldWidth, WorldHeight;    // World Space Dimension

  void HardwareToWorldPoint(int hwX, int hwY, float &wcX, float &wcY);
                                       // Transform mouse clicks (hwX, hwY) to World Cooridnate (wcX, wcY)

  virtual void draw() {                // virtual function from Fl_Glut_Window for drawing
    glClearColor( 0.8f, 0.8f, 0.95f, 0.0f );
    glClear(GL_COLOR_BUFFER_BIT);      // Clearing the background color
    glMatrixMode(GL_PROJECTION);       // Programming the OpenGL's GL_PROJECTION
        glLoadIdentity();              //      Matrix Processor to the propoer transfrotm
        gluOrtho2D(0.0f, WorldWidth, 0.0f, WorldHeight);
    TheModel->DrawApplicaitonState();  // Drawing of the application state
  }
};
```

图18-23　FlGlutWindow：OpenGL/FLTK 视图/控制器对

2．Direct3D和MFC

图18-24展现了击球程序构建期间MFC资源编辑器的屏幕截图，这个编辑器就是MFC的GUI编码程序。类似于Fluid（图18-21），在图18-24的中间可以看到，该资源编辑器也能够在表示应用程序窗口的区域（B）中，交互式地设计GUI元素的布局（A）。虽然这个GUI编码程序的接口操作不太相同，但是我们可以看到在（C）中，MFC资源编辑器也能够定义/修改GUI元素的外观。不过，与Fluid不同的是，MFC资源编辑器与整个开发环境是紧密集成的。在这种情况下，开发
440 人员可以通过继承或覆盖适当的服务例程来注册事件服务。MFC资源编辑器会自动将代码片断插入到应用程序源代码系统中。为了支持这一功能，应用程序源代码的组织是由GUI编码程序控制的，或者是与GUI编码程序共享。应用程序开发人员并不能完全自由地重命名文件或类，或者是重新组织实现源代码的文件系统结构。MFC对于事件服务连接采用了内部直接的代码修改机制，就像18.4.1节所描述的那样。

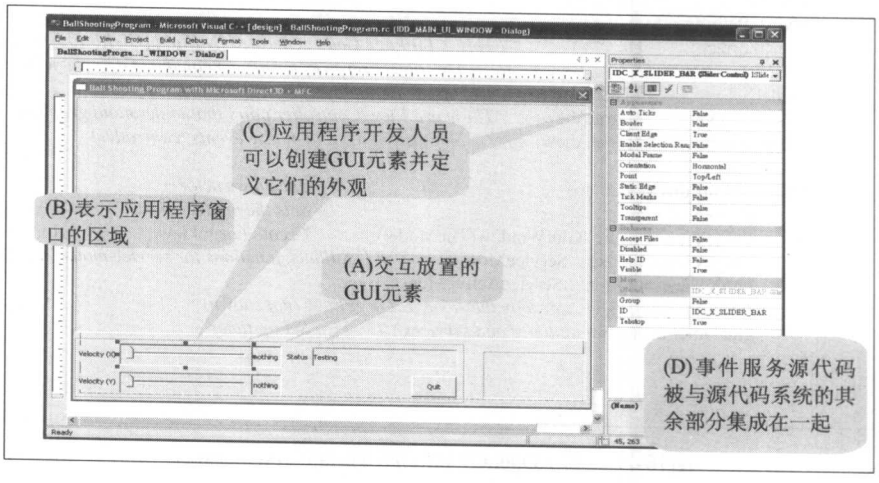


图18-24　MFC资源编辑器

图18-25使用了Microsoft Direct3D和MFC实现的MainUIWindow。在这一实现中，图形操作是通过Direct3D执行的，而用户界面操作则由MFC来支持。同样，TheModel为图18-11所描述的应用程序状态。*LPDIRECT3D9*是图形硬件环境（GHC）接口对象。该对象是在MainUIWindow的构造函数（这里没有显示该函数）中创建并初始化的。两个CWndD3D对象主要用来支持Direc3D绘

图。注意，图18-25和图18-22之间的一个主要区别在于对GUI元素的支持。在图18-25中，GUI元素对象（比如 XSlider）以及相应的服务例程（比如*OnHScroll*()）被集成到MainUIWindow内部。与此不同，图18-22所示的解决方案将GUI元素归入到一个单独的对象中（比如UserInterface对象），采用回调事件服务注册机制。在18.2.4节提到，MFC是一个大型商业GUI API，很多事件服务的注册都建立在面向对象的函数覆盖基础上（例如，*OnHScroll*()和*OnTimer*()函数）。

```
class MainUIWindow : public CDialog {
    Model          *TheModel;    // The application State (Figure 11)
    LPDIRECT3D9    TheGHC;       // This is the Graphics Hardware Context
    CWndD3D        *LargeView;   // These are View/Controller pairs that understand drawing
    CWndD3D        *SmallView;   //     with D3D (GDC) and UI element events (controller)

    CSliderCtrl    XSlider, YSlider;   // These are the GUI elements
    CStringEcho    StatusEcho;

        void    OnTimer();             // Override the Timer service function
        void    OnHScroll( ...);       // Override the Scroll bar service function
};
```

图18-25 基于Microsoft Direct3D和MFC的MainUIWindow

CWndD3D是MFC CWnd类的子类（见图18-26）。CWnd是针对一般的MFC窗口设计的基类。作为这个基类的子类，CWndD3D能够支持所有默认的与窗口相关的事件（比如鼠标事件）。 441
LPDIRECT3DDEVICE9是D3D图形设备环境（GDC）接口对象。*InitD3D*()函数创建并初始化GDC对象，并将该对象连接到*LPDIRECT3D9*(GHC)。这样，子类CWndD3D就是一个基本的视图/控制器对：支持通过D3D GDC绘图的视图功能，以及通过MFC输入的控制器功能。函数*RedrawView*()类似于图18-23中的*draw*()函数。在调用模型进行绘图之前，需要先建立绘制状态（例如，bgColor和矩阵），其中包括矩阵处理器的编程（例如，D3DTS_WORLD）。

```
// CWnd is the MFC base class for all window objects. Here we subclass to create a D3D output
// window by including a D3D Graphics Device Context.
class CWndD3D: public CWnd {
  LPDIRECT3DDEVICE9  D3DDevice;   // This is the D3D Graphics Device Context (GDC)
  Model              *TheModel;   // The application state
  void InitD3D(LPDIRECT3D9);      // Create D3DDevice (GDC) to connect to GHC

  void RedrawView()  {            // Draws the Application State
        // Compute world coordinate to device transform
        D3DMATRIX transform = ComputeTransformation();
        D3DDevice->SetTransform(D3DTS_WORLD, &transform);
              // Programming the D3D_WORLD matrix with the computed transform matrix

        D3DDevice->Clear( bgColor, D3DCLEAR_TARGET);
        D3DDevice->BeginScene();
          TheModel->DrawApplicationState(D3DDevice);
        D3DDevice->EndScene();
        D3DDevice->Present();
  }

  void HardwareToWorldPoint(CPoint hwPt, float &wcX, float &wcY);
                        // Transform mouse clicks (hwPt) to world coordinate (wcX, wcY)

  void OnLButtonDown(CPoint hwPt);  // Override mouse button/drag service functions
  ⋮
};
```

图18-26 CWndD3D：Direct3D/MFC视图/控制器对

总之，图18-20代表18.3.3节所介绍的解决方案的一种实现。而18.4.3节给出了图18-20中实现方案的两个版本。虽然GUI编码程序、事件服务的注册，以及实际的API函数调用都不同，但是最终源代码的编程结构却惊人地相似。实际上，对于Model类，两个版本都采用了完全一样的源

代码文件。另外，虽然OpenGL和D3D的绘图函数*CirclePrimitive*不一样，但是其他的原语行为（比如，设置圆心/半径、滑行、碰撞，等等）却能够共享源代码文件。在此，再次重申，软件框架、
442 方案结构以及事件的实现在设计上应该独立于任何API。

18.5 结果的运用

事件驱动编程模型非常适合于设计并实现与用户交互的程序。此外，对于交互式图形应用程序而言，模型-视图-控制器框架是一种功能强大而方便的结构。开发击球程序解决方案的过程表明，事件驱动编程的知识有助于设计控制器组件（比如鼠标事件的处理等），计算机图形学知识有助于设计视图组件（比如圆的变换与绘制等），而模型组件则高度依赖于具体的应用程序（比如，自由降落并发生碰撞的圆，等等）。迄今为止，我们的讨论都是围绕一个简单的实例展开的。下面将分析MVC框架的适用性及其在实际应用程序中的实现。

18.5.1 示例1：PowerPoint

图18-27展现了如何在深入分析Microsoft PowerPoint[1]的过程中运用我们所学的知识。Microsoft PowerPoint是一个流行的交互式图形应用程序。图18-27的左边展示了一张使用PowerPoint创建幻灯片的屏幕照片。图18-27的右边则展示了如何运用实现框架来洞察PowerPoint应用程序。图18-27右边的MainUIWindow就是整个应用程序的GUI窗口。该窗口包含影响或显示
443 应用程序状态的GUI元素（比如，主菜单、状态区等）。可以将MainUIWindow看作是包含TheModel组件以及四个视图/控制器对的模块。

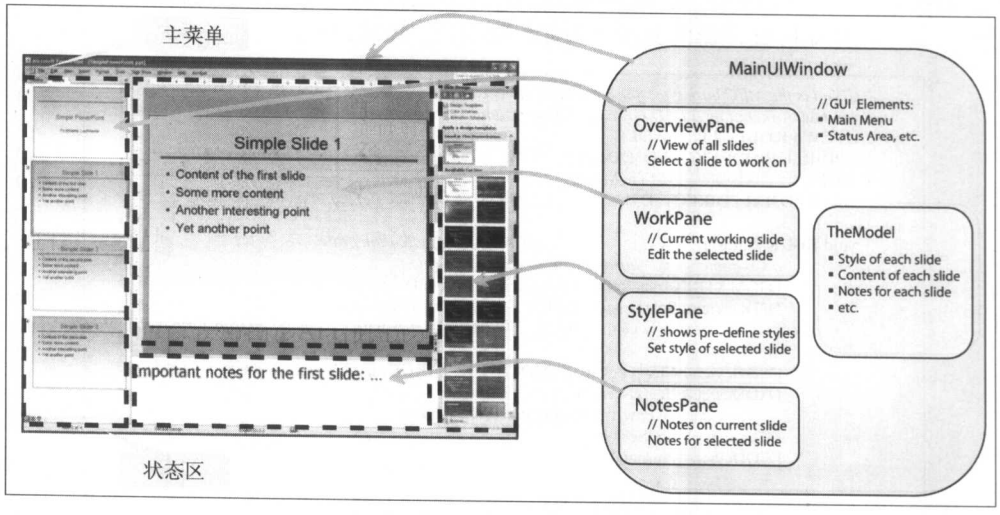


图18-27　通过MVC实现框架了解Powerpoint

注意，TheModel就是应用程序的状态，包含了所有用户交互创建的数据。在PowerPoint中，用户创建的是幻灯片集合，因此该模型包含了这些幻灯片的所有信息（例如，布局设计风格、幻灯片内容、与每个幻灯片关联的注释等）。按照这种方式来理解TheModel组件，那么应用程序的其余部分就可以看作是将TheModel（视图）呈现给用户、并允许用户改变TheModel（控制器）的便利工具。这样，这些便利工具正好就是视图/控制器对（例如，图18-16的ViewController组件）。

1. PowerPoint是Microsoft的注册商标。

图18-27中的每一个视图/控制器对（即OverviewPane、WorkPane、StylePane、NotesPane）都呈现TheModel组件的一个不同方面，并且支持对TheModel组件不同方面的更改：

- **OverviewPane**。这个视图组件显示用户创建的所有幻灯片中多张连续的幻灯片。控制器组件允许用户滚动浏览这些幻灯片，并选定一张进行编辑。
- **WorkPane**。该视图组件显示正在编辑的幻灯片的详细情况，而控制器组件则支持对该幻灯片内容的选择与编辑。
- **StylePane**。该视图组件显示当前正在编辑的幻灯片的布局设计。控制器组件允许为该幻灯片选择并定义新的布局设计。
- **NotesPane**。这个视图组件显示用户为当前编辑的幻灯片创建的注释。控制器组件支持对这一注释的编辑。

与大多数现代的交互应用程序一样，PowerPoint定义了一个应用程序定时器事件，支持用户定义的动画（例如，幻灯片切换之间的动画序列）。四个视图控制器对之间的一致性可以在应用程序定时器事件的服务期间维护。例如，用户使用StylePane改变TheModel组件中当前幻灯片的布局。在处理下一个定时器事件之前，OverviewPane和WorkPane都不会知道这一改变，因而为当前幻灯片显示的是过时的设计。在定时器事件的服务期间，MainUIWindow会强制所有视图/控制器对调查TheModel，并刷新它们的内容。按照18.3.4节的讨论，定时器事件每秒触发的次数通常超过30，因此用户检测不到暂时过时的显示，一直看到的都是一致的显示。这样，四个视图/控制器对只需要保持一个对模型组件的引用，而不需要了解任何其他组件。因此，应用程序中视图/控制器对的插入和删除都比较简单。 444

18.5.2 示例2：Maya

现在运用所学的知识来分析了解Maya[1]。Maya是一个交互式的三维建模/动画/绘制系统。图18-28的左边显示了一个简单的三维内容创建会话中的Maya屏幕照片。跟图18-27一样，图18-28的右边展示了如何运用实现框架来洞察Maya应用程序。同样，MainUIWindow是整个应用程序的GUI窗口，包含影响和显示整个应用程序状态的GUI元素、TheModel组件以及所有视图/控制器对。

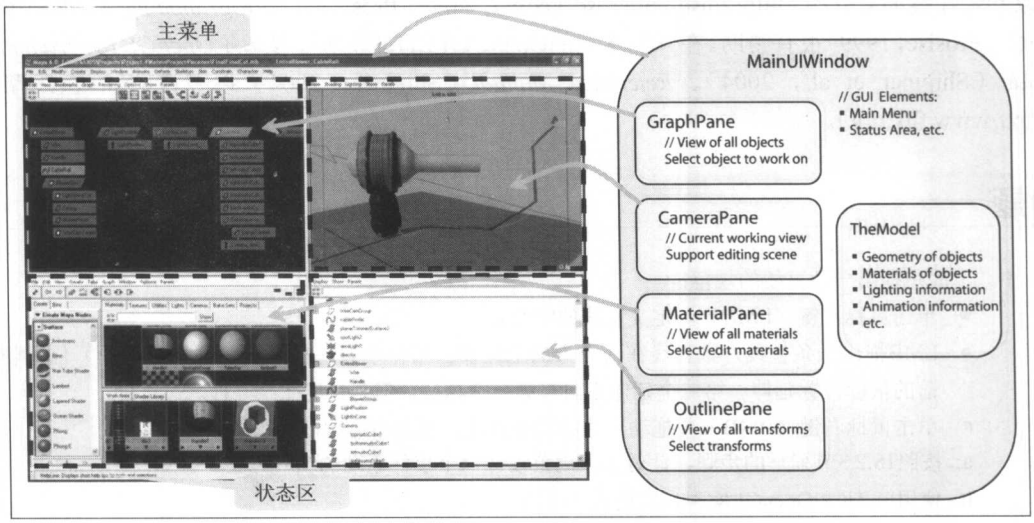


图18-28 通过MVC实现框架了解Maya

1. Maya是Alias的注册商标。

因为Maya是一个三维媒体创建系统，所以TheModel组件包含三维内容信息（例如，场景图、三维几何结构、材质性质、光照、照相机、动画等）。同样，MainUIWindow中其他组件是用来方
445 便用户查看或更改TheModel的。此处，四个视图/控制器对的功能是：

- **GraphPane**。该视图组件显示三维内容的场景图。控制器组件支持场景导航，以及图中布景结点的选择。
- **CameraPane**。该视图根据照相机的查看位置绘制场景图。控制器组件支持照相机的查看操作，以及场景中对象的选择。
- **MaterialPane**。该视图显示所有定义的材质。控制器组件支持对这些材质的选择与编辑。
- **OutlinePane**。该视图显示场景中所有变换结点。控制器组件支持对变换的操作（比如，创建或修改父子关系，等）。

同样，不同视图/控制器对之间的一致性可以通过应用程序定时器事件的服务来维护。

不用推测PowerPoint或者是Maya是否按照我们的框架实现。它们都是非常成熟的商业应用程序，其底层实现肯定也相当复杂。不过，根据本章所学的知识，我们可以开始掌握如何分析、设计并建立这样的交互式图形应用程序。记住，需要从本章学习的重要内容是，如何将交互式图形应用程序的功能组织成组件，并掌握组件之间的交互，这样就能更好地理解、维护、修改并扩充交互式图形应用程序。

文献注记

我最初是从SmallTalk（Goldberg & Robson,1989）（还可以参考SmallTalk的网站（http://www.smalltalk.org/main/））了解到模型-视图-控制器框架与事件驱动编程的。在*Design Patterns——Elements of Resuable Object-Oriented Design*（Gamma, Helm, Johnson & Vlissides, 1995）和*Pattern-Oriented Software Architecture*（Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996）两书中可以找到更多有关设计模式和软件体系架构的一般内容。对于有关实时图形系统实现方面的问题讨论，可参考*3D Game Engine Architecture*（Eberly，2004）。学习MFC和Direct3D可以参考在线的MSDN（http://msdn.microsoft.com）。此外，Prosise的书*Programming Windows with*
446 *MFC*（Prosise, 1999）很有帮助。在开发本章的OpenGL/FLTK程序时，参考了*OpenGL Programming Guide*（Shreiner et al.，2004）、*Reference Manual*（Shreiner，2004）以及FLTK的在线帮助（http://www.fltk.org）。

习题

1. 下面是拖出一条直线的规范：
   - 单击鼠标左键（LMB），定义直线的中心。
   - LMB拖出一条直线，该直线在两个方向上延伸。第一个方向从中心（LMB单击的）位置朝当前的鼠标位置延伸。第二个延伸方向与第一个相反，但是长度完全一样。
   - 单击鼠标右键（RMB）并拖动，将移动该直线，使其中心跟随当前的鼠标位置。

   a. 按照18.2.3节列举的步骤，针对上述规范设计一个事件驱动的编程方案。

   b. 使用FLTK和OpenGL实现自己的设计方案。

   c. 使用MFC和Direct3D实现自己的设计方案。

   注意在这个案例中，有用的应用程序内部状态信息（直线的中心位置）与绘图表示需求（直线的端点）并不完全相同。在定义应用程序状态的时候，需要注意什么是需要存储的最重要的最方便

的信息，以便支持指定的功能。

2. 给习题1定义的直线，确定一个速度，速度值等于该直线的斜率：一旦确定了，直线将沿斜率所确定的方向滑行。将直线的长度作为速度使用（注意较长的直线要比短的直线滑得快）。
3. 下面是拖出一个矩形的规范：
   - 单击LMB，定义矩形的中心。
   - LMB拖出一个矩形。该矩形从中心向四周扩展，其中矩形一角的位置一直跟随鼠标的当前位置。 447
   - RMB单击并拖动，移动该矩形，使其中心跟随当前的鼠标位置。

   a. 按照18.2.3节列举的步骤，针对上述规范设计一个事件驱动的编程方案。

   b. 使用FLTK和OpenGL实现自己的设计方案。

   c. 使用MFC和Direct3D实现自己的设计方案。
4. 对于习题3中的矩形：

   a. 要求支持与18.1节类似于HeroBall的速度定义：一旦创建，矩形就会沿着一个方向滑行，该方向是从矩形中心指向LMB释放位置的向量。

   b. 设计并实现两个矩形之间的碰撞（这是简单的二维边界相交检验）。
5. 使用习题4的结果，近似模仿一个简单的Pong游戏：
   - 拍子用矩形表示。
   - Pong球画作圆，但是使用绑定的方形（中心位于圆心的正方形，边长等于圆的半径）来近似与拍子的碰撞。

   设计并实现单个玩家的Pong游戏，其中一个球在重力的作用下降落，用户必须操作拍子弹起该球，防止它掉到应用程序窗口的下面。为此，应该：

   a. 为这个Pong游戏设计一个规范（类似于18.1节的规范）；

   b. 按照18.2.3节列举的步骤，设计一个事件驱动的编程解决方案；

   c. 使用OpenGL或者是Direct3D实现自己的设计。
6. 扩充图18-12的ApplicationView，使其包括设置世界坐标系窗口边界的功能。世界坐标系窗口确定了世界上一个在Viewport中显示的矩形区域。定义一个函数用来设置世界坐标系窗口边界，并修改*ApplicationView::DeviceToWorldXform*()函数，使其支持从鼠标点击到世界坐标系空间的变换。 448
7. 将习题6的结果集成到图18-14所示的双视图击球程序中，使小视图能够在当前HeroBall的周围聚焦。在当前没有HeroBall的时候，小视图应该不显示任何内容。当用户单击并拖动LMB，或者是用RMB选择一个HeroBall，小视图的世界坐标系窗口边界就应该集中在HeroBall的中心，并包括一个1.5倍于HeroBall直径的区域。 449

第19章

光　照

本章将讨论光照测量的实际问题，也就是通常所说的辐射测量。辐射测量中的有关术语初看起来可能比较陌生，而且有些符号和术语可能不太直观。但是，辐射测量是计算机图形学的重要基础，因此学习并了解辐射测量是必要的。本章还涵盖光度测定的内容。光度测定根据辐射量的多少估计“有用”的光照量。比如，在同样的能量强度下，绿光的亮度看起来可能是蓝光的两倍，因为眼睛对绿光更敏感。光度测定将尝试量化这种区别。

19.1 辐射测量

虽然可以在多个系统中定义辐射测量单位，但是本书采用SI（国际量纲）定义的计量单位。常见的SI计量单位包括米（m）和克（g）。光本质上是一种能量传播方式，因此可以按照能量的SI量纲定义，即焦耳（J）。

19.1.1 光子

451

为直观起见，可将辐射描述为大量光子的集合。本节将在这一语境中建立光子的含义。在本章的用途中，光子指的是有一定位置和传播方向的光量子，其波长为λ。有些奇怪的是，在SI量纲中波长的单位为纳米（nm）。这主要是历史的原因造成的，1 nm=10^{-9}m。有时也使用另一个单位埃（angstrom），1纳米等于10埃。光子还具有速度c。c仅取决于传输介质的折射系数n。有时为了方便，还使用频率f作为光的单位，$f=c/\lambda$。与c和λ不同的是，当光子折射到另一种折射系数不同的介质中时f不会改变。另外一个不变的测量单位是光子所携带的能量q，q的定义如下：

$$q = hf = \frac{hc}{\lambda} \tag{19-1}$$

其中h=6.63×10^{-34}J，为普朗克常量。尽管这些物理量可以在任一量纲系统中测量，我们还是将尽可能地使用SI量纲。

19.1.2 光谱能量

如果有大量的光子集合，那么将每个光子的能量q_i相加就能得到它们的总能量Q。一个合理的问题就是“该能量在各个波长上的分布如何？”对该问题的一个简单回答就是将所有光子分配到不同的波长槽（bin）中，计算它们的统计直方图。然后就可以获得每个间隔上的能量。例如，将所有波长在500 nm到600 nm之间的光子能量相加，可以得到10.2 J，那么可以将其表示为q[500, 600]=10.2 J。如果将该波长间隔分割为两个50 nm的区间，则可能得到q[500, 550]=5.2、q[550, 600]=5.0。这表明在[500, 600]的波长范围内，波长较短那一部分的能量高一些。如果将其分为25nm间隔，那么可能得到q[500,525]=2.5，等等。统计直方图的好处是直观，不利的是间隔大小的选

择决定了间隔的数目。

另一个更常用的方法是将能量的大小除以区间的大小。对于$q[500, 600]=10.2$ J，将得到

$$Q_\lambda[500,600]=\frac{10.2}{100}=0.12\ \mathrm{J(nm)^{-1}}$$

该方法的好处在于区间间隔的大小对区间的数目影响不大。对此一个直接的推导是令间隔大小$\Delta\lambda$趋近于0。然而，对于足够小的$\Delta\lambda$，可能会使Q_λ等于0，或者是无穷大，具体情况取决于区 452
间是只有一个光子，还是没有光子。解决这一两难问题的方法有两个。第一个是假设Q_λ很小，但是没有小到光的量子属性起作用的程度。第二个是假设光是连续的，而不是单个的光子，这样就可以得到一个真实的偏导数$\mathrm{d}Q/\mathrm{d}\lambda$。这两种方式都是可行的，最终都得到同样的计算机制。在实践中，大多数人更偏向于按照精细的小间隔来测量光，因为那是能够在实验室进行测量的。而那些做理论或计算研究的人偏向于采用无穷小的间隔，因为那样可以使用微积分的方法。

数量Q_λ被称为光谱能量，是一个强度量，不同于能量、长度、质量等广延量。强度量可以被看作密度函数，指出一个无穷小点上的广延量的密度。例如，在某个特定波长上的能量Q可能为0，而光谱能量（能量密度）Q_λ却是一个有意义的量。另一个较为熟悉的例子是：某个国家的人口为2 500万，而在该国某个点上的人口数量却没有意义。然后，如果在一个足够大的面积上测量，那么按人数计算的每平方米的人口密度却是有意义的。这与光子的情况非常类似，如果将人口看作连续统一体，即使在很小的面积上人口密度也不会变为颗粒状，那么人口密度就可以很好地使用。

本书将遵循图形学的约定，一直使用光谱能量，而很少使用能量。其结果是在使用严格意义上的符号时，需要增加一个下标λ。相反，我们将抛弃该下标，使用Q表示光谱能量。这在图形学之外阅读图形学的论文时，可能会引起一些混淆，因此一定要注意有关标准的问题。为了直观地认识光谱能量，可以设想一个带有能量传感器的测量光能q的装置。如果将一个仅允许$[\lambda-\Delta\lambda/2,\lambda+\Delta\lambda/2]$范围内的光通过的彩色滤波器放在该传感器前面，那么$\lambda$的光谱能量将为$Q=\Delta q/\Delta\lambda$。

19.1.3 功率

估计光源的产生能量的比率是有意义的。该比率被称为功率，其测量单位为瓦（W）。瓦是焦耳/秒的另一种叫法。在平常状态下这不难理解，但是由于功率是一个强度量（随时间变化的密度函数），所以即使能量的生产率随时间变化，它也是良定义的。功率的单位可能更常见一些，例如100 W的灯泡。这样的灯泡大概每秒发射100 J能量。由于热损耗的原因，该灯泡产生的功率实际上要小于100 W。但是我们仍然可以借助这个例子来帮助我们理解光子。例如，我们可以计 453
算100 W的灯泡每秒产生了多少个光子。假设每个光子产生的平均能量等于一个$\lambda=500$ nm的光子产生的能量。这样一个光子的频率为：

$$f=\frac{c}{\lambda}=\frac{3\times10^{8}\ \mathrm{ms^{-1}}}{500\times10^{-9}\mathrm{m}}=6\times10^{14}\ \mathrm{s^{-1}}$$

该光子的能量$hf\approx4\times10^{-19}$ J。这意味着每秒钟产生的光子数量达将10^{20}，即使该灯泡不是非常有效。这解释了使用快门和光子来模拟照相机产生图像效率低下的原因。

对于能量，通常关注的是功率谱。功率谱的单位为$\mathrm{W(nm)^{-1}}$。虽然功率谱的正规标准符号为Φ_λ，但是为了使用方便并与大多数图形学文献一致，仍然采用不带下标的Φ来表示。需要注意的是光源的功能谱通常是一个比功率要小的数值。例如，灯的发射功率为100 W，分布在400 nm

至800 nm的波长范围内，那么功率谱将等于100 W/400 nm=0.24 W(nm)$^{-1}$。在手工设置光源的功率谱进行调试的时候需要记住这一点。

上一小节的光谱能量测量装置需要稍做改动，增加一个开启时间间隔为Δt的读取快门。令时间间隔Δt的中心为t，那么功率谱将等于$\Delta Q/(\Delta t \Delta \lambda)$。

19.1.4 辐射照度

如果问“多少光线照到某个点上？”将引出另外一个量——辐射照度。为了回答该问题，必须再次使用密度函数。如果该点在某个表面上，自然可以使用面积来定义密度函数。此时，修改上一节的测量装置，假设传感器的有限面积为ΔA，而且要比所测量的光场小，那么辐射照度谱将等于单位面积上的功率$\Delta\Phi/\Delta A$，其完整的定义式如下：

$$H=\frac{\Delta q}{\Delta A \Delta t \Delta \lambda} \tag{19-2}$$

因此辐射照度的完整单位就是Jm^{-2}s^{-1}(nm)$^{-1}$。注意在辐射率的SI计量单位中，面积使用平方米的倒数，波长采用了纳米的倒数。这种既使用米又使用纳米的不一致性是由面积和可见光波长的自然单位引起的。

454

如果光线正离开某个表面，例如反射，那么此时辐射照度称为辐射脱离E。对入射光和反射光采用不同的术语是有帮助的，因为同一个点可能有不同的辐射照度和辐射脱离。

19.1.5 辐射率

辐射照度虽然反映了多少光线达到某个点，但是没有指明光线从何方而来。为了度量类似于人眼所看到的某些东西，需要将“多少光线”与某个具体的方向关联起来。请设想一个如图19-1所示的简单测量装置。使用一个小的辐射照度计和一个锥形“调节墙”，将击到计数器上的光线限制在一个立体角度$\Delta\sigma$的范围内，那么检测器的响应将等于

$$\text{响应}=\frac{\Delta H}{\Delta\sigma}=\frac{\Delta q}{\Delta A \Delta\sigma \Delta t \Delta\lambda}$$

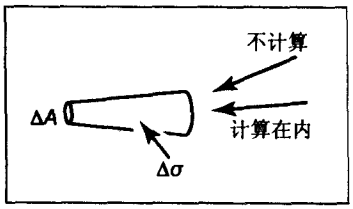


图19-1　给辐射照度检测器增加一个眼罩，只露出一个很小立体角$\Delta\sigma$，就可以测量辐射率

这就是从空中穿过的光线的谱辐射率。同样，在本书的讨论中，将省略“谱”这一词，而假设其为隐含的。

455

辐射率是一个在图形学程序中需要经常计算的量。辐射率的特殊属性之一是：它不随空间距离变化。为了说明这一点，假设有两个辐射率检测器都像图19-2所示的那样，指向一个表面。而且假定两个检测器之间的垂直距离足够小，小到可以认为被测的区域都以同样的方式发射或反射光线。因为被采样的表面面积与距离的平方成正比，达到检测器的光线与距离的平方成反比，所以两个检测器的读数将一样。

测量某个表面的辐射率是有意义的。如图19-3所示，可以设想把辐射率检测器的锥形调节墙放在表面的某个点上，然后测量锥体方向所指的表面的辐射照度H。注意表面检测器没有与锥体对齐。为此，需要在辐射率的定义中增加一个余弦校正项，即

$$\text{响应}=\frac{\Delta H}{\Delta\sigma\cos\theta}=\frac{\Delta q}{\Delta A\cos\theta\Delta\sigma\Delta t\Delta\lambda}$$

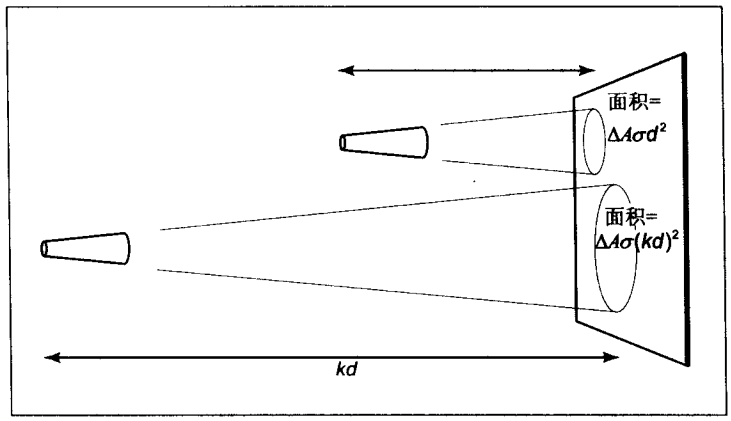


图19-2 辐射率检测器接收的信号不依赖于测量装置与被测表面的距离。本图假定检测器指向以同样方式发射光线的被测表面

对于辐射照度和辐射脱离而言，有必要区分输入到表面某点的输入辐射率，以及离开该点的输出辐射率。对于这些概念，在图形学文献中有时使用术语“表面辐射率”L_s表示离开某个表面的辐射率，“场辐射率”L_f表示表面的输入辐射率。两者都需要余弦项，因为他们都符合图19-3所示的配置情况。

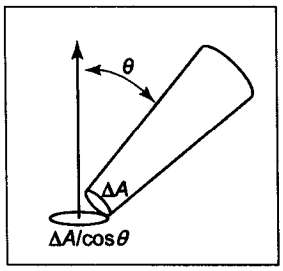


图19-3 被锥体遮盖的表面辐射照度要小于检测器测量的结果，两者相差一个余弦因子

$$L_s = \frac{\Delta E}{\Delta\sigma\cos\theta}$$

$$L_f = \frac{\Delta H}{\Delta\sigma\cos\theta}$$

辐射率和其他辐射度量量

如果某个表面的场辐射率为L_f，则据此可以推导出所有其他辐射度量量。这就是辐射率被看作基本辐射度量量的原因之一。例如，辐射照度可以表示为

$$H = \int_{所有\boldsymbol{k}} L_f(\boldsymbol{k})\cos\theta\mathrm{d}\sigma$$

该公式有几个符号上的约定，这些约定在图形学中是很普遍的，但是就是这些约定使得这样的公式对不熟悉它们的读者晦涩难懂（图19-4）。首先，$\boldsymbol{k}$指的是入射方向，可以将其看作单位向量、方向，或者是相对于表面法向的球坐标系中的坐标对(θ,ϕ)。该方向有一个关联的微分立体角度$\mathrm{d}\sigma$。场辐射率在各个方向一般都不同，因此将其写为函数$L(\boldsymbol{k})$。

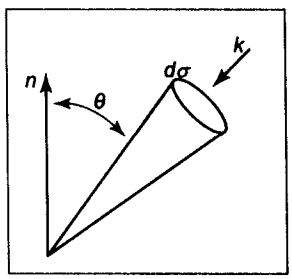


图19-4 方向$\boldsymbol{k}$有一个关联的微分立体角$\mathrm{d}\sigma$

456

作为一个例子，我们可以计算在所有方向上都具有同样场辐射率L_f的表面的辐射照度H。为了积分的方便，采用传统的球面坐标系，另外微分立体角为

$$\mathrm{d}\sigma \equiv \sin\theta\mathrm{d}\theta\mathrm{d}\phi$$

因此辐射照度为

$$H = \int_{\phi=0}^{2\pi}\int_{\theta=0}^{\frac{\pi}{2}} L_f\cos\theta\sin\theta\mathrm{d}\theta\mathrm{d}\phi = \pi L_f$$

这一关系首次出现了一个潜在的令人惊讶的常量π。π这样的因子经常出现在辐射线测定中，是由

所选择的立体角的测量方式而带来的。也就是说，单位球面的面积是π的倍数，而不是1的倍数。

类似地，沿着表面面积对辐射照度积分，可以得到冲击表面的功率：

$$\Phi = \int_{所有x} H(\boldsymbol{x})\mathrm{d}A$$

其中$\boldsymbol{x}$为表面的点，$\mathrm{d}A$是与该点关联的微分面积。注意这里没有针对输入功率和输出功率采用特别的术语或符号。因为他们的区别还未上升到需要加以区别对待的程度。

19.1.6 BRDF

因为我们对表面的外观感兴趣，所以需要刻画表面如何反射光线。从直觉来看，任何来自方向$\boldsymbol{k}_i$的入射光线，都有一小部分分散在反射方向$\boldsymbol{k}_o$附近比较小的立体角内。有很多方式可以将这一概念形式化，而标准的方式是建立一个简单的测量装置。图19-5给出了一个这样的测量装置，其中小光源位于从表面向某点查看的$\boldsymbol{k}_i$上，检测器位于$\boldsymbol{k}_o$方向上。对于每一方向对$(\boldsymbol{k}_i,\boldsymbol{k}_o)$都用检测器检测一个值。

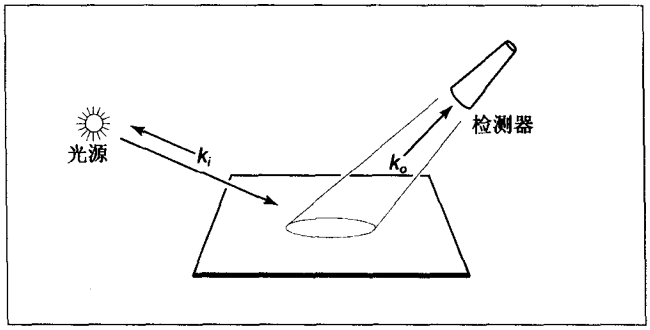


图19-5　一个简单方向反射系数测量装置。光源和检测器的位置可以移到每一个可能的方向对上。注意$\boldsymbol{k}_i$和$\boldsymbol{k}_o$都指向远离表面的方向，以便交互作用

457 下面必须决定如何测量光源的强度，并使反射函数独立于该强度。例如，当光源被一个更亮的光源取代时，将不需要考虑表面发射光线的不同。我们可以在被照亮的测光点上放置一个辐射计。然而，为了得到一个准确的不依赖于检测器$\Delta\sigma$的读数，需要光源对向一个比$\Delta\sigma$稍大的立体角。不幸的是，方向$\boldsymbol{k}_o$上的这个粗放的辐射率检测器还会把探测器锥体之外的光线计算在内。因此这看起来不是一个实用的方案。

作为一种替换方案，可以在表面的被测点上放置一个辐射照度仪表。这样就可以得到一个不太依赖于光源微妙几何形状的读数。这要求将反射系数刻画为一个比值：

$$\rho = \frac{L_s}{H}$$

其中分数ρ将随入射方向$\boldsymbol{k}_i$和出射方向$\boldsymbol{k}_o$而变化，H是位置$\boldsymbol{k}_i$上的光辐射照度，L_s为在方向$\boldsymbol{k}_o$上测得的表面辐射率。在所有的方向上都进行这种测量，就可以得到一个4维函数$\rho(\boldsymbol{k}_i,\boldsymbol{k}_o)$。该函数被称为*双向反射率分布函数*（BRDF）。BRDF刻画物体表面反射光线的方向性质。

1. 半球面方向反射系数

给定BRDF，自然会问“多少入射光线被反射？”该问题不太好回答。反射部分依赖于入射
458 光线的方向分布。因此，通常只设置固定入射方向$\boldsymbol{k}_i$上的反射部分。该部分被称为*半球面方向反射系数*$R(\boldsymbol{k}_i)$，其定义如下：

$$R(\boldsymbol{k}_i) = \frac{\text{所有出射方向}\boldsymbol{k}_o\text{上的功率}}{\text{方向}\boldsymbol{k}_i\text{上的光束功率}}$$

注意，按照能量守恒的法则，该系数的取值在0到1之间。如果投射到面积 ΔA 上的功率为 Φ_i，那么辐射照度就是 $\Phi_i / \Delta A$。此外，入射功率的比例正好等于脱离辐射率与辐射照度的比值，即：

$$R(\boldsymbol{k}_i) = \frac{E}{H}$$

按照BRDF的定义，某个特定方向上的辐射率为

$$L(\boldsymbol{k}_o) = H\rho(\boldsymbol{k}_i, \boldsymbol{k}_o) = \frac{\Phi_i}{\Delta A}$$

另外根据辐射率的定义，可得

$$L(\boldsymbol{k}_o) = \frac{\Delta E}{\Delta\sigma_o \cos\theta_o}$$

其中E为方向 $\boldsymbol{k}_o$ 上小块碎片的脱离辐射率。根据这两个辐射率的定义，可以得到

$$H\rho(\boldsymbol{k}_i, \boldsymbol{k}_o) = \frac{\Delta E}{\Delta\sigma_o \cos\theta_o}$$

重新安排有关项后，上式变为

$$\frac{\Delta E}{H} = \rho(\boldsymbol{k}_i, \boldsymbol{k}_o)\Delta\sigma_o \cos\theta_o$$

这就是在特定方向 $\boldsymbol{k}_o$ 附近反射的那一部分 E/H。为了得到总的 $R(\boldsymbol{k}_i)$，将所有的出射方向 $\boldsymbol{k}_o$ 的结果相加即可。用积分形式表示就是

$$R(\boldsymbol{k}_i) = \int_{\text{所有}\boldsymbol{k}_o} \rho(\boldsymbol{k}_i, \boldsymbol{k}_o)\cos\theta_o \mathrm{d}\sigma_o$$

2. 理想的漫反射BRDF

理想的漫反射表面被称为*朗伯表面*（Lambertian）。现实中由于热力学的原因，这样的表面是不可能存在的，但是在数学上却值得研究它们。朗伯BRDF的 ρ 在所有方向上都等于常量。也就 459
是在所有查看角度上，该表面都具有同样的辐射率，该辐射率与辐射照度成正比。

如果用 $\rho = C$ 来计算朗伯表面的 $R(\boldsymbol{k}_i)$，将得到

$$\begin{aligned} R(\boldsymbol{k}_i) &= \int_{\text{所有}\boldsymbol{k}_o} C\cos\theta_o \mathrm{d}\sigma_o \\ &= \int_{\phi_o=0}^{2\pi}\int_{\theta_o=0}^{\pi/2} k\cos\theta_o \sin\theta_o \mathrm{d}\theta_o \mathrm{d}\phi_o \\ &= \pi C \end{aligned}$$

这样，对于一个全反射朗伯表面（R=1），有 $\rho = 1/\pi$；对于 $R(\boldsymbol{k}_i) = r$ 的朗伯表面，有

$$\rho(\boldsymbol{k}_i, \boldsymbol{k}_o) = \frac{r}{\pi}$$

这里再次出现了固定立体角决定归一化常量的情况，从而引入了一个因子 π。

19.2 输送方程

类似BRDF的定义，可以按照各个不同方向输入的辐射率来描述表面的辐射率。因为在计算机图形学中，使用理想化的数学描述在实验室中不一定可行，所以可以只按照辐射率来描述 460
BRDF。如果采用立体角为 $\Delta\sigma_i$ 的一小部分光线，假设其辐射率为 L_i，然后测量方向 $\boldsymbol{k}_o$ 上反射的辐射率，就可以得到BRDF（如图19-6所示）。这一小束光线的辐射照度 $H = L_i \cos\theta_i \Delta\sigma_i$，从而BRDF为

$$\rho = \frac{L_o}{L_i \cos\theta_i \Delta\sigma_i}$$

在某些情况下这种表达方式是有用的。改写上式之后，可以得到来自 k_i 方向的光线产生的辐射率：

$$\Delta L_o = \rho(k_i, k_o) L_i \cos\theta_i \Delta\sigma_i$$

如果有来自很多方向 $L_i(k_i)$ 的光线，则可以将它们都累加起来。利用表面辐射率和场辐射率表达的积分形式如下：

$$L_s(k_o) = \int_{\text{所有}k_i} \rho(k_i, k_o) L_f(k_i) \cos\theta_i \mathrm{d}\sigma_i$$

这就是计算机图形学中常见的绘制方程(Immel, Cohen, & Greenberg, 1986)。

有时需要将输送方程写成仅含表面辐射率的形式(Kajiya，1986)。注意，在一个封闭的环境中，场辐射率 $L_f(k_i)$ 是来自某个表面的辐射率，即有 $L_f(k_i) = L_s(-k_i)$（如图19-7所示）。图中点 x' 所指向的立体角为

$$\Delta\sigma_i = \frac{\Delta A' \cos\theta'}{\|x - x'\|^2}$$

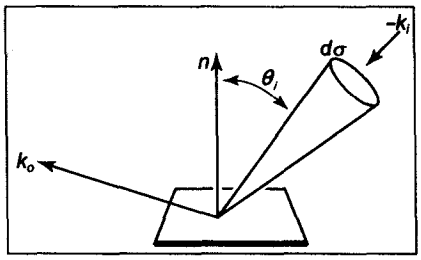


图19-6　输送方程的几何方向

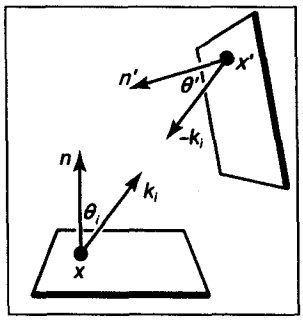


图19-7　从一个点进入另一个点的光线

其中 $\Delta A'$ 是与 x' 关联的面积。用 $\Delta A'$ 替换 $\Delta\sigma_i$ 即可得到下述输送方程：

$$L_s(x, k_o) = \int_{\text{所有对}x\text{可见的}x'} \frac{\rho(k_i, k_o) L_s(x', x - x') \cos\theta_i \cos\theta'}{\|x - x'\|^2} \mathrm{d}A'$$

注意，这里采用了没有归一化的向量 $x - x'$ 来表示从 x' 到 x 的方向。此外，L_s 被写成位置和方向的函数。

新输送方程的唯一问题就是积分域不好确定。如果引入一个可见性函数，那么就可以在积分域的复杂性和被积函数的复杂性之间进行均衡，即

$$L_s(x, k_o) = \int_{\text{所有}x'} \frac{\rho(k_i, k_o) L_s(x', x - x') v(x, x') \cos\theta_i \cos\theta'}{\|x - x'\|^2} \mathrm{d}A'$$

其中

$$v(x, x') = \begin{cases} 1 & \text{如果}x\text{和}x'\text{相互可见} \\ 0 & \text{其他} \end{cases}$$

461

19.3　光度测定

对于每一个谱辐射量都有一个相关的光度测定量，来衡量多少辐射率是对人眼有效的。假设谱辐射量为 $f_r(\lambda)$，那么相关的光度测定量 f_p 为

$$f_p = 683 \frac{\mathrm{lm}}{\mathrm{W}} \int_{\lambda=380\mathrm{nm}}^{800\mathrm{nm}} \bar{y}(\lambda) f_r(\lambda) \mathrm{d}\lambda$$

其中 $\bar{y}$ 为人类视觉系统的发光效率函数。该函数在上述积分范围之外等于零，因此将积分范围放大到0与∞之间，f_p 不会改变。发光效率函数将在第20章更详细地讨论，本章只讨论它的一般属性。开头的常量是为了该定义与历史上的绝对光度测定量保持一致。

发光效率函数在各个波长上的敏感程度并不相等（见图19-8）。波长低于380 nm的（紫外范围）光线对人眼是不可见的，因此$\bar{y}$等于零。从380 nm开始$\bar{y}$值逐渐增加，直到$\lambda = 555$ nm时达到最大值，最大值处为纯绿色的光线。然后，其值逐渐递减直到抵达800 nm的红外区域边界。

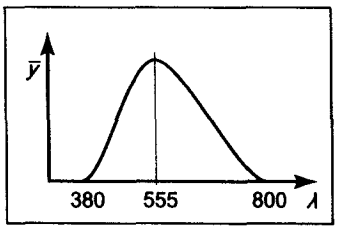


图19-8 亮度效率函数与波长（nm）

计算机图形学中最常用的光度测定量是*亮度*，其辐射率模拟结果为

$$Y = 683\frac{\text{lm}}{\text{W}}\int_{\lambda=380\text{ nm}}^{800\text{ nm}} \bar{y}(\lambda)L(\lambda)\text{d}\lambda$$

用符号Y表示亮度起源于色度学。别的领域大多采用符号L。我们将不采用这一约定，因为用L既表示亮度又表示谱辐射率太容易混淆了。亮度给出了一个物体有多亮的一般概念，而且是与观察者无关的。注意，在中午太阳底下的黑纸主观感觉上要比底亮度月光下的白纸黑。对亮度考虑太多是危险的，但是这有利于从量上把握相对可觉察的光线输出。单位*lm*表示流明（lumens）。注意大多数灯泡是按照所消耗的功率来鉴定等级的，而所产生的有效光线则是按流明计算。灯泡效率越高，产生的光线就越多，其$\bar{y}$也越大，从而每瓦所产生的流明越多。一个理想的灯泡将把所有功率转化为555 nm的光线，每瓦产生683流明。从而亮度的单位就是$(\text{lm/W})(\text{W}/(\text{m}^2\text{sr}))=\text{lm}/(\text{m}^2\text{sr})$。每个立体角单位内的流明量被定义为一烛光（candela，cd），因此亮度通常采用单位cd/m^2。 462

常见问题

- **什么是“光强度”？**

术语“光强度”在很多环境下都被采用，其用法既与时代有关，也与学科有关。实际上，作为一个特定的辐射度量量，它没有任何意义，但是有助于直观的讨论。大多数论文采用这一术语都是用来表示辐射率。

- **什么是“有效辐射”？**

有些领域使用术语“有效辐射”来表示脱离辐射率。有时也用它来描述物理空间的光线传输算法。

文献注记

有一个常用的辐射度量量没有在本章描述，那就是辐射强度（I）。辐射强度指的是从一个无穷小的点源上向每个单位立体角发射功率谱。在图形学的程序中通常应该避免它，因为点源会带来实现上的麻烦。更严格的辐射测量方法可以在*Analytic Methods for Simulated Light Transport*（Arvo，1995）一书中找到。本章的术语辐射测定和光度测定都取自照明工程协会的标准。这一标准正被越来越多的科学和工程技术领域所采用（美国国家标准研究所，1986）。有关辐射测定和外观标准的深入讨论可参阅*Principles of Digital Image Synthesis*（Glassner，1995）。

习题

1. 对于没有输出辐射率的漫反射表面，其脱离辐射是什么？
2. 一个面积为4 m^2、辐射率为L的漫反射表面，总的脱离功率为多少？
3. 一个荧光灯和一个白炽灯都消耗20 W的功率，为什么荧光灯通常更受欢迎？ 463

第20章 颜 色

第21章将提到，人有三种传感器（视锥）参与了高层照明的感知。这三种传感器类型的信号决定了观察者的颜色响应。因此，颜色在本质上是三维空间的现象。为了从量上描述颜色，需要在该三维空间中建立一个良定义的坐标系。在图形学中通常采用“红-绿-蓝”（RGB）三种颜色来表示这样的坐标系。然而，适用于该空间的坐标系几乎有无穷多种，没有哪一种本质上比别的好。在具体的环境下，则有一些颜色系统要比别的好。这就好比在一个城市里面，坐标系与街道对齐，要比精确地与东西南北对齐好。因此，在处理颜色的时候，会遇到很多缩略语，如CMY、XYZ、HSV和LUV，它们都代表有三个命名轴的坐标系。这些都很难保持平直。

此外，颜色还涉及到光进入眼睛的物理过程，以及由此产生的心理。物理、心理、认知之间的区别往往容易混淆。而使事情更为复杂的是，有些颜色空间是面向显示或打印技术的。例如CMKY就是针对使用墨水的打印机的，在三种彩色墨水之外再增加黑色（K）墨水。为了尽可能澄清这些事实，本章将从基本原理开始介绍颜色感知。对于普通的RGB颜色空间而言，本章的讨
465 论看起来可能有点太复杂。但是颜色本身就是一个复杂的问题。对于图形学中这样一个中心主题进行简化是有害的。

本章将从光检测器开始，然后逐步展开基本的三元色（三种颜色）理论。由该理论将自然导出对RGB显示系统的处理。此外，还将讨论LMS和XYZ系统。

20.1 光和光检测器

人眼之所以能够“看见”物体，是因为光线进入了眼睛，并射到眼睛背部视网膜（retina）上的光检测器上。与此类似，当射入数码相机内部光检测器阵列上的光线较多时，数码相机记录的内容也就更高。

到达光检测器的信号随波长而变，可用谱辐射率（spectral radiance）$L(\lambda)$来描述。$L(\lambda)$表示从某个特定的方向上输入某个特定波长的光的光强度。眼睛中的视网膜/晶体的作用非常类似于19.1.5节介绍的辐射率检测器，其中晶体作用是使得眼睛能够收集更多的光线，而简单的打开状态收集的光线要比这少得多。

并非所有光线都同等处理。人眼对某些波长的光更敏感，并且对波长在[380 nm, 800 nm]范围之外的光都不敏感。照相机也具有类似的灵敏特性。任何一个这样的检测器的响应都可以表示为一个加权函数w与谱辐射率的乘积的积分，即

$$\text{响应} = k\int w(\lambda)L(\lambda)\mathrm{d}\lambda$$

该响应方程是所有颜色理论的基础。其中，常量k稍具随意性，在不同的颜色理论中取值往往不同；响应函数w反映传感器的特性，在不同的颜色理论中也不一样。

20.2 三刺激颜色理论

如果假设人的颜色响应是人眼中多种不同类型传感器综合作用的结果，那么由此引出的最直接的问题就是："我们需要处理多少种传感器？"现在我们已经知道有三种被称为视锥的传感器在描述我们日常的颜色视觉。为了了解在19世纪是如何验证这一点的，请考虑一个建立在这三种传感器类型假设基础上的实验。假定传感器之间相互独立，那么各传感器对谱辐射率 $A(\lambda)$ 的响 466
应就是（Wyszecki & Stiles，1992）

$$S=\int s(\lambda)A(\lambda)\mathrm{d}\lambda$$
$$M=\int m(\lambda)A(\lambda)\mathrm{d}\lambda$$
$$L=\int l(\lambda)A(\lambda)\mathrm{d}\lambda$$

如果两个不同的辐射率 $A_1(\lambda)$ 和 $A_2(\lambda)$ 产生同样的 (S,M,L)，那么就所考虑的传感器系统而言它们是不可区分的。这样匹配的谱被看作是同一种颜色，称之为条件等色（metamer）。这一观察使我们能够验证确实存在三种传感器。

假设有三个点光源，照耀到屏幕上时，各自的谱曲线分别为 $R(\lambda)$、$G(\lambda)$、$B(\lambda)$，而且都带有一个控制旋钮，通过小数 (r,g,b) 分别对它们进行上下调节。这样，最终的谱曲线就是

$$A(\lambda)=rR(\lambda)+gG(\lambda)+bB(\lambda)$$

对这一混合光线的响应 S 为

$$\begin{aligned}S_A&=\int s(\lambda)A(\lambda)\mathrm{d}\lambda\\&=\int s(\lambda)\big(rR(\lambda)+gG(\lambda)+bB(\lambda)\big)\mathrm{d}\lambda\\&=r\int s(\lambda)R(\lambda)\mathrm{d}\lambda+g\int s(\lambda)G(\lambda)\mathrm{d}\lambda+b\int s(\lambda)B(\lambda)\mathrm{d}\lambda\\&\equiv rS_R+gS_G+bS_B\end{aligned}$$

上面最后一个等式就是将整个光的响应 S 定义为 (S_R,S_G,S_B) 的结果。

现在假设有第四个光源，谱辐射率为 $C(\lambda)$，照耀在屏幕上紧邻上述三种重叠的彩色光源。传感器对第四个光源的响应为 (S_C,M_C,L_C)。这样整个实验的关键在于：可以调整三种重叠光源的 (r,g,b) 加权值，使得光源对传感器而言看起来是一样的，即

$$\begin{aligned}S_C&=S_A=rS_R+gS_G+bS_B\\M_C&=M_A=rM_R+gM_G+bM_B\\L_C&=L_A=rL_R+gL_G+bL_B\end{aligned}$$

注意，这是一由三个方程和三个未知数 (r,g,b) 组成的线性方程组。只要该系统不是退化的，就存在唯一的 (r,g,b) 满足该系统。 467

进行该实验，可以匹配颜色，进而验证确实存在三种传感器。对于该实验的实际性能，有一种重要的细节需要注意，那就是不能保证 r、g 和 b 的值都是非负的，或者是小于等于1。在这样的情况下，匹配实验就无法进行。但是，如果允许在第四个光源中混入前三种光源的组合，那么匹配实验就可以一直进行。例如：当匹配情况出现在 $r=-0.2$ 时，可以在第四种光源中混入 $0.2R(\lambda)$，这与从三个重叠光源中减去 $0.2R(\lambda)$ 的结果是一样的。

一旦建立这三种传感器，下一个重要问题就是："响应函数 $s(\lambda)$、$m(\lambda)$ 和 $l(\lambda)$ 分别是什么？"不幸的是，无法采用非入侵过程来推导这些函数。这些响应函数的估计到20世纪80年代才确定。然而，实际上提出颜色匹配方案，并不需要知道视锥响应函数。任何三种线性独立的光都可以用

来指定一种颜色。例如，针对前面讨论的三种光，可以采用(r,g,b)值指定颜色。如果两个光谱具有同样的(r,g,b)值，那么它们就是同样的“颜色”。注意，很容易出现两种不同的光谱具有同样(r,g,b)值的情况，此时这两种光谱就是条件等色。

最佳匹配光源是什么，以便将颜色值标准化？该问题在20世纪30年代提出，并逐步发展成为*XYZ*颜色系统。该系统至今仍然是指定三刺激颜色的压倒性选择。下一节将讨论该系统。

20.3　CIE 三刺激值

CIE（一个颜色标准化组织）观测到：一旦列举出一组给定光源的数据，就可以从数学上计算指定光谱的三刺激值。他们还进一步观测到：任何一组实际的光源都将导致一些测试光谱出现负的三刺激值。因此，他们认为没有必要将其限制为物理上可实现的光源。例如，已知实际光源$R(\lambda)$、$G(\lambda)$和$B(\lambda)$的数据，就可以推演出这些光源的线性组合的数据，如$-R(\lambda)$、$G(\lambda)-2B(\lambda)$和$B(\lambda)+R(\lambda)$，即使这些光源在物理上是不存在的。其三刺激值将为$(-r,g-2b,b+r)$。

CIE决定采用具有如下两种良好特性的虚构的光源：

- 光源之一是“灰色”的，不提供任何色调信息。

468

- 其他两种光源的亮度为零，且只提供色调信息。

这三种光的响应由三元组(X,Y,Z)确定，其中Y表示亮度。由于人眼只对380 nm到800 nm范围内的光有响应，所以积分的范围也就限定在这一范围内。因为在该范围之外加权函数衰减为零，所以采用这一明确范围有一定的冗余。常量683是为了遵循亮度的标准。CIE三刺激值(X,Y,Z)的公式为

$$X = 683\int_{380}^{800}\overline{x}(\lambda)L(\lambda)\mathrm{d}\lambda$$

$$Y = 683\int_{380}^{800}\overline{y}(\lambda)L(\lambda)\mathrm{d}\lambda$$

$$Z = 683\int_{380}^{800}\overline{z}(\lambda)L(\lambda)\mathrm{d}\lambda$$

任意给定的谱辐射率都将有相应的(X,Y,Z)值。

20.4　色度

通常，我们需要分离出亮度而专注于颜色。这么做的标准方法是采用色度（chromaticity）值（图20-1）：

$$(x,y)=\left(\frac{X}{X+Y+Z},\frac{Y}{X+Y+Z}\right)$$

对于z也有类似的公式，但是很少用，因为$x+y+z=1$。最为*XYZ*的替代，人们经常使用xyY。这样就可以单独讨论“颜色”和“强度”。由xyY也可以计算XYZ：

$$(X,Y,Z)=\left(\frac{xY}{y},Y,\frac{(1-x-y)Y}{y}\right)\qquad(20\text{-}1)$$

在某种意义上，可以将(x,y)看作是色调（hue）所需要的所有信息。色调是颜色

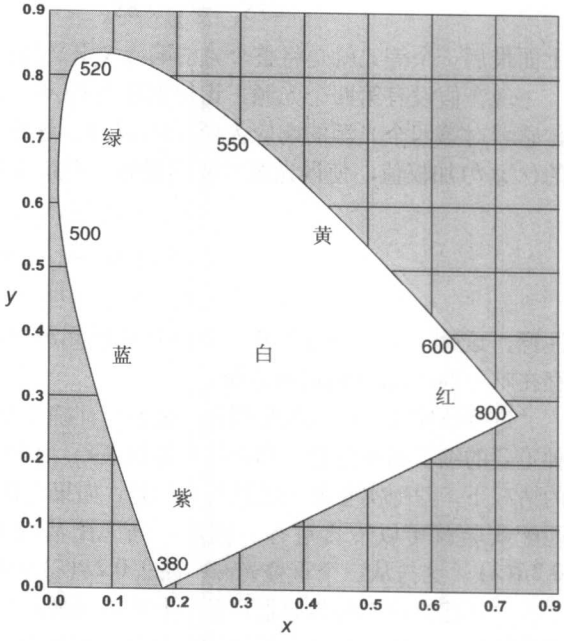


图20-1　CIE xy空间。图中的曲线边界由纯光谱的颜色构成，并显示了纯光谱的波长（纳米）

的彩色部分。由于那是一个二维空间，因此可以将与(x,y)空间关联的所有颜色都绘制在一个平面上。在外观上比较奇特的是，与颜色关联的点(x,y)形成了一个奇怪的形状。实平面上大多数点都没有关联的颜色。由于所有三刺激值都是非负的，所以(x,y)值也是非负的。由于(x,y)中的每一个值都是由一个非负的数除以一个不比它小的非负数而得，所以(x,y)被限定在区间$[0,1]^2$内。然而，实际取值的限制比这更严格。

图20-2中的表格给出了三刺激值和暗视灵敏度曲线的取值。注意，$\bar{y}$值从不超过$\bar{x}$的5倍。也就是说，不管谱输入为多少，Y值从不超过X的5倍。这将y限制为最多5/6。最极端的情况出现在纯光谱颜色下。波长为λ_0的纯光谱颜色的谱辐射率为一伸缩δ函数，即$k\delta(\lambda_0)$。其三刺激值为

$$(X,Y,Z)=683k(\bar{x}(\lambda_0),\bar{y}(\lambda_0),\bar{z}(\lambda_0))$$

这样，相应的色度值就是：

$$(x,y)=\left(\frac{\bar{x}(\lambda_0)}{\bar{x}(\lambda_0)+\bar{y}(\lambda_0)+\bar{z}(\lambda_0)},\frac{\bar{y}(\lambda_0)}{\bar{x}(\lambda_0)+\bar{y}(\lambda_0)+\bar{z}(\lambda_0)}\right) \tag{20-2}$$

20.5 暗视亮度

对于低级亮度，例如月光，人眼进入一种不同感知模式，而且谱灵敏度也发生变化。此时XYZ值是不相关的，明暗是由暗视亮度（scotopic luminance）决定的：

$$V'=k'\int_{380}^{800}\bar{v}'(\lambda)L(\lambda)\mathrm{d}\lambda$$

其中常量$k'=1700\frac{\mathrm{S}}{\mathrm{W}}$，取该值是为了使555 nm的单色光束（白天视觉的峰值灵敏度）具有同样的亮度和暗视亮度。

对于那些实际上处于暗视范围内而使用V值更合适的数据，往往给出的是三原色值(X,Y,Z)。虽然不可能从三元色值推导出V值，但是Ward指出在很多情况下，可以合理地使用下面的经验公式（Larson, Rushmeier, & Piatko, 1997）：

$$V=Y\left[1.33\left(1+\frac{Y+Z}{X}\right)-1.68\right] \tag{20-3}$$

λ (nm)	$\bar{x}$	$\bar{y}$	$\bar{z}$	$\bar{v}'$
380	0.0014	0.0000	0.0065	0.0006
390	0.0042	0.0001	0.0201	0.0022
400	0.0143	0.0004	0.0679	0.0093
410	0.0435	0.0012	0.2074	0.0348
420	0.1344	0.0040	0.6456	0.0966
430	0.2839	0.0116	1.3856	0.1998
440	0.3483	0.0230	1.7471	0.3281
450	0.3362	0.0380	1.7721	0.4550
460	0.2908	0.0600	1.6692	0.5670
470	0.1954	0.0910	1.2876	0.6760
480	0.0956	0.1390	0.8310	0.7930
490	0.0320	0.2080	0.4652	0.9040
500	0.0049	0.3230	0.2720	0.9820
510	0.0093	0.5030	0.1582	0.9970
520	0.0633	0.7100	0.0782	0.8350
530	0.1655	0.8620	0.0422	0.8110
540	0.2904	0.9540	0.0203	0.6500
550	0.4334	0.9950	0.0087	0.4810
560	0.5945	0.9950	0.0039	0.3288
570	0.7621	0.9520	0.0021	0.2076
580	0.9163	0.8700	0.0017	0.1212
590	1.0263	0.7570	0.0011	0.0655
600	1.0622	0.6310	0.0008	0.0332
610	1.0026	0.5030	0.0003	0.0159
620	0.8544	0.3810	0.0002	0.0074
630	0.6424	0.2650	0.0000	0.0033
640	0.4479	0.1750	0.0000	0.0015
650	0.2835	0.1070	0.0000	0.0007
660	0.1649	0.0610	0.0000	0.0003
670	0.0874	0.0320	0.0000	0.0001
680	0.0468	0.0170	0.0000	0.0001
690	0.0227	0.0082	0.0000	0.0000
700	0.0114	0.0041	0.0000	0.0000
710	0.0058	0.0021	0.0000	0.0000
720	0.0029	0.0010	0.0000	0.0000
730	0.0014	0.0005	0.0000	0.0000
740	0.0007	0.0002	0.0000	0.0000
750	0.0003	0.0001	0.0000	0.0000
760	0.0002	0.0001	0.0000	0.0000
770	0.0001	0.0000	0.0000	0.0000

图20-2 暗视灵敏度和1931 CIE三刺激灵敏度曲线取值（Rea，1993）

469

470

20.6 RGB显示器

我们有时候知道CRT的RGB通道的三刺激值。假设红色通道的为(X_r,Y_r,Z_r)，绿色通道的为(X_g,Y_g,Z_g)，蓝色通道的为(X_b,Y_b,Z_b)。假设显示器上有一个全黑的点（实际上不太可能），那么给定一个伽马校正的RGB信号(r,g,b)，就可以计算屏幕上的三刺激值(X_s,Y_s,Z_s)：

$$\begin{bmatrix} X_s \\ Y_s \\ Z_s \end{bmatrix} = \begin{bmatrix} rX_r + gX_g + bX_b \\ rY_r + gY_g + bY_b \\ rZ_r + gZ_g + bZ_b \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

对上述方程求逆，就可以找到计算给定(X,Y,Z)值的(r,g,b)的方法。注意，得到的值可能比1.0大，或者是比0.0小，因此需要采取一些措施来处理这一问题。然而，更大的问题是，显示器的制造厂商几乎从来不会公开(X_r,Y_r,Z_r)等值。实际上，他们提供的是荧光粉的色度(x_r,y_r)、(x_g,y_g)、(x_b,y_b)，以及白点（white point）的色度(x_w,y_w)。此外，可能通常需要使用光度计测量最亮的白屏的亮度Y_w。如果不能测量，则可以假定该值大约为$Y_w = 100\ \text{cd/m}^2$。制造商不公布该值的理由是：亮度的调节会改变该值。白点变化的原因则是：通常的“白”指的是房间的平均色。这样，在荧光照明的房间中看到的白色主要是短波长的光，而在白炽灯照明的房间中看到的白色则主要是长波长的光。因此，如果将一个看起来图像是白的显示器，从荧光照明的房间移到白炽照明的房间，同样的显示将看起来发蓝。正是这一问题使得摄影师购买“日光”胶卷或“钨”胶卷。

471
|
472

为了转换制造商提供的信息（三刺激值），需要做一些代数变换。首先，写出直观的等式

$$\begin{bmatrix} \frac{X_r}{Y_r} & \frac{X_g}{Y_g} & \frac{X_b}{Y_b} \\ 1 & 1 & 1 \\ \frac{Z_r}{Y_r} & \frac{Z_g}{Y_g} & \frac{Z_b}{Y_b} \end{bmatrix} \begin{bmatrix} Y_r \\ Y_g \\ Y_b \end{bmatrix} = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

然后进行一些置换，得到只有未知量(Y_r,Y_g,Y_b)的形式：

$$\begin{bmatrix} \frac{x_r}{y_r} & \frac{x_g}{y_g} & \frac{x_b}{y_b} \\ 1 & 1 & 1 \\ \frac{1-x_r-y_r}{y_r} & \frac{1-x_g-y_g}{y_g} & \frac{1-x_b-y_b}{y_b} \end{bmatrix} \begin{bmatrix} Y_r \\ Y_g \\ Y_b \end{bmatrix} = \begin{bmatrix} \frac{x_w y_w}{y_w} \\ Y_w \\ \frac{(1-x_w-y_w)Y_w}{y_w} \end{bmatrix}$$

可以使用数值方法或者是代数上的克莱姆法则求解(Y_r,Y_g,Y_b)。一旦得到该值，就可以利用式（20-1）得到(X_r,X_g,X_b)、(Y_r,Y_g,Y_b)和(Z_r,Z_g,Z_b)。既然有了该结果，就可以将其作为具体的显示器参数使用。这些参数可以通过测量获得，也可以采用适当的标准值，例如ITU在1990年规定的值，或网页http://www.w3.org/Graphics/Color/sRGB.html给定的值。

20.7 近似颜色操作

虽然上一节的技术对于高度控制的设置而言是有用的，但是在实践中很少知道使用他们所需要的足够数据。而往往所做的图像需要在很多未知的显示器上显示，例如网页图像。因此，比较

方便的方法是，采用“归一化”的空间，将$(R,G,B)=(1,1,1)$变换为$(X,Y,Z)=(1,1,1)$。该空间对大多数真实的显示器而言都是“合理的”。在*Color Transfer Between Images*（Reinhard, Ashikhmin, Gooch, & Shirley, 2001）一书中给出了一个这样的空间，本书稍加修改之后，将其列举如下： 473

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.5149 & 0.3244 & 0.1607 \\ 0.2654 & 0.6704 & 0.0642 \\ 0.0248 & 0.1248 & 0.8504 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{20-4}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 2.5623 & -1.1661 & -0.3962 \\ -1.0215 & 1.9778 & 0.0437 \\ 0.0752 & -0.2562 & 1.1810 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{20-5}$$

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3897 & 0.6890 & -0.0787 \\ -0.2298 & 1.1834 & 0.0464 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{20-6}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1.9102 & -1.1122 & 0.2019 \\ 0.3709 & 0.6291 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix} \tag{20-7}$$

这就给出了RGB和LMS视锥空间之间的如下变换：

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.3816 & 0.5785 & 0.0399 \\ 0.1969 & 0.7246 & 0.0785 \\ 0.0248 & 0.1248 & 0.8504 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{20-8}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 4.4620 & -3.5832 & 0.1213 \\ -1.2178 & 2.3803 & -0.1626 \\ 0.0486 & -0.2448 & 1.1962 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix} \tag{20-9}$$

20.8 对立颜色空间

虽然存在很多“直观的”颜色空间，创建了颜色的三个维度，但是最著名的也许是对立颜色模型（opponent color model）(Wyszecki & Stiles, 1992)。这些模型有三个通道，通常类似于：

$$\begin{aligned} \text{全色} &\propto r+g+b \\ \text{黄蓝} &\propto r+g-b \\ \text{红绿} &\propto r-g \end{aligned} \tag{20-10}$$

这类模型虽然有一些生物学上的理由，但是它的提出却是建立在19世纪主观实验的基础上）(Wyszecki & Stiles, 1992)。例如，在人的体验中有带蓝色的绿，却没有带黄色的蓝。这暗示蓝和绿有不同的通道，黄和蓝则是一个共享的通道，要么是黄，要么是蓝，但不可能既是黄又是蓝。

虽然有些对立模型采取对数空间（Ruderman, Cronin, & Chiao, 1998; Reinhard et al., 2001），但是这里给出的是一个线性模型，其中α和β为彩色通道： 474

$$\begin{bmatrix}\mathcal{L}\\ \alpha\\ \beta\end{bmatrix}=\begin{bmatrix}\frac{1}{\sqrt{3}} & 0 & 0\\ 0 & \frac{1}{\sqrt{6}} & 0\\ 0 & 0 & \frac{1}{\sqrt{2}}\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & -2\\ 1 & -1 & 0\end{bmatrix}\begin{bmatrix}L\\ M\\ S\end{bmatrix}$$

$$\begin{bmatrix}L\\ M\\ S\end{bmatrix}=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & -1\\ 1 & -2 & 0\end{bmatrix}\begin{bmatrix}\frac{\sqrt{3}}{3} & 0 & 0\\ 0 & \frac{\sqrt{6}}{6} & 0\\ 0 & 0 & \frac{\sqrt{2}}{2}\end{bmatrix}\begin{bmatrix}\mathcal{L}\\ \alpha\\ \beta\end{bmatrix}$$

常见问题

- **什么是“色调”？**

色调是颜色的非白成分中占优势的颜色的名称。例如，粉红的色调为红。很多颜色系统将色调编码为0°到360°之间的角度，红色在0°位置上，绿色在120°，蓝色为240°。

- **什么是“光亮度（lightness）”？**

光亮度是指表面反射率的总强度。该值对于黑色最小，对于白色最大。两种不同色调的颜色可以具有同样的光亮度。光亮度通常用0到1之间的标量表示。

- **什么是“饱和度”？**

饱和度（saturation）是指颜色的纯度。例如，红比粉红更饱和，而灰色根本不饱和。

- **什么是“估计值”？**

估计值（value）是光亮度的另一叫法，通常以近似的数值刻度来表示它。例如，在有些系统中，估计值等于RGB的平均值。这样具有相同估计值的颜色，可能具有不同的主观光亮度，因为
475 RGB通道的感知是不同的。

文献注记

Computer Generated Color (Jackson，MacDonald, & Freeman，1994)是一篇优秀的针对计算机图形用户的颜色综述。从颜色科学角度撰写的书有*The Reproduction of Colour*（Hunt，2004）和*Color Appearance Models*（Fairchild，2005）。在Meyer和Greenberg（Meyer & Greenbery，1988）的书中讨论了针对颜色缺陷观察者的颜色空间。

习题

1. 使用式（20-2）绘制CIE色度图的轮廓。
2. 物理上什么颜色的CIE值(x, y)可能为零？
476 3. 编程单独调节RGB空间和对立空间中的每个颜色通道。哪一种产生更坏的结果？

第21章

视觉感知

William B. Thompson

计算机图形学的最终目的是产生供人查看的图像。这样，计算机图形系统的成功就取决于它如何将相关信息传递给人类观察者。由于物理世界内在的复杂性和显示设备的限制，不可能向观察者呈现观察自然环境时同样的光模式。当计算机图形系统的目标为物理真实时，可以期望的最佳结果是该系统在感知上有效：显示的图像看起来跟“预期的”一样。对于类似技术演示之类的应用，通常期望在视觉上突出显示相关信息。这样*感知效率*（perceptual effectiveness）成为一种明确的需求。

艺术家和插图画家已经根据经验开发了大量的工具和技术，以便高效地传递视觉信息。提高计算机图形感知效率的途径之一就是在自动系统中利用这些技术和方法。第二种途径直接建立在人类视觉系统知识的基础上，将感知效率当作优化计算机图形系统设计的标准。这两种途径并非完全不同。实际上，最初的系统化的视觉感知验证之一就是在达芬奇（Leonardo da Vinci）的笔记本中找到的。

本章的剩余部分将局部回顾人类所了解的有关视觉感知的内容，重点在于与计算机图形学相关的人类视觉方面。人类视觉系统在操作和体系结构上都是极其复杂的。像本章这样的篇幅最多只能提供关键要点的概述，请注意避免将本章介绍的内容过分一般化。更深入的视觉感知处理可以在Wandell（1995）和Palmer（1999）的文献中找到。Gregory（1997）和Yantis（2000）的文献中提供另外一些有用的信息。比较好的类似于Forsyth和Ponce（2002）的计算机视觉参考文献也是有帮助的。重要的是要注意到：尽管经过了150多年深入细致的研究，我们在很多方面对视觉的认识和了解还非常有限、非常不完善。

21.1 视觉科学

一般认为视觉是人最强大的感觉。视觉产生的关于这个世界的有用信息要远远超过听觉、触觉、嗅觉或味觉。这是由光的物理特性（见图21-1）直接导致的结果。照明是普遍而深入的，特别是在白天，晚上也是这样，因为有月光、星光和人造光源的存在。物体表面反射大部分入射照明（incident illumination），反射的方式取决于具体的材质和表面形状。（大部分）光线在空气中沿直线传播的事实使得视觉能够获取远方位置信息。

光：
- 传播远
- 传播快
- 沿直线传播
- 与材料相互作用
- 遇到物体会反射
- 自然产生
- 能量大

——Steven Shafer

图21-1 光使视觉成为强大感觉的自然特性

视觉研究的历史很长也很丰富。关于人眼的很多知识可以追溯到17世纪的哲学家和物理学家所做的工作。在19世纪中叶，感知心理学家就开始了视觉现象学的探索工作，并提出了有关视觉如何工作的模型。20世纪中叶开始了现代神经科学，研究细微尺度上的单个神经元的工作，以及大尺度上的大脑和神经系统的组织结构。大部分神经科学的研究工作都集中在视觉上。最近，计算机科学为视觉感知的理解作出了重要贡献，

提供了精确描述视觉计算假设模型的工具，允许完全根据经验检验计算机视觉程序。术语视觉科学（vision science）主要指多学科的视觉感知研究，其中涉及感知心理学、神经科学和计算分析。

视觉科学认为视觉的目的是：根据观察者接收光线而获得的图像模式，产生有关现实世界中物体、位置和事件的信息。心理学家用术语远侧刺激（distal stimulus）表示观察的物理世界，用
478 近侧刺激（proximal stimulus）表示视网膜图像[1]。按照该术语，视觉的功能就是根据近侧刺激生成远侧刺激的情况描述。当所产生的描述精确反映了真实世界时，视觉感知就被认为是诚实的（veridical）。实际上，单独考虑物体、位置和事件的描述没有多大意义。相反，视觉在其所服务的运动和认知功能中能得到更好的理解。

21.2 视觉灵敏度

视觉系统根据入射照明的性质产生可视环境的描述。所以，重要的是掌握人类视觉系统实际上检测什么样的入射照明性质。一个关于人类视觉系统的关键观察是：视觉主要对光的模式（pattern）敏感，而不是对光能的绝对大小敏感。眼睛并不是像光度计那样工作。实际上，眼睛检测的是光在视网膜上成像的空间模式、时间模式和谱模式。有关这些光模式的信息构成了所有视觉感知的基础。

已经有清晰的生态学工具来衡量视觉系统对照明时空变化的敏感度。精确地感知环境的变化对于生存来讲是至关重要的[2]。一个系统如果能够测量光能的变化，而不是光能本身的大小，将更具有工程价值。因为那将更容易在大的光强范围内检测光的模式。对于计算机图形学而言，视觉以这种方式工作是一件好事。显示设备在物理上限制了投影光能和自然场景动态变化范围的能力。如果要求图形显示产生与物理世界一样的光模式，那么显示的效率就不高。幸运的是，只要求显示器产生类似于现实世界时空变化的模式。

21.2.1 亮度和对比度

在明亮的光照下，人的视觉系统能够区分由明暗条纹组成的光栅。这样的光栅可以精细到每度50～60个周期（在这里，一个“周期”由一对相邻的明暗条纹组成）。最为比较，当前最好的
479 LCD计算机监视器，在正常的视距上能够显示的模式最高约为20个周期/度。在明亮关照下，人的视觉系统能够检测出来的最小边缘对比差异为该边缘平均亮度的1%。在大多数8位显示器上，单一灰度级的区别在一定的光亮度范围内往往很明显，这是由灰度级与实际显示亮度之间的映射造成的。

描绘视觉系统检测精细模式的能力（视觉灵敏度（visual acuity））以及检测亮度变化的能力，要比描绘照相机或类似图像获取装置的情况更为复杂。如图21-2所示，人的视觉在对比度与敏锐度之间存在交互作用。在该图中，模式的尺度从左到右逐渐减小，而对比度从上到下逐步增加。如果在正常的视距上查看该图，将清楚地看到在模式可见处的最小对比度是该模式空间频率的函数。

从现实世界中表面某点进入人眼的光线的强度L，与照射到该点的光线强度I和观察到的该点表面反射率R之间一种线性关系：

480
$$L = \alpha I \cdot R \tag{21-1}$$

1. 在计算机视觉中，通常用场景（scene）表示外部世界，而用图像（image）表示场景在传感平面上的投影。
2. 曾经有人说视觉的主要目的是为了支持吃和繁殖、避免被吃掉以及移动碰撞。将视觉看作一种目标导向的活动是有帮助的，但是需要从更详细更具体的层面来分析。

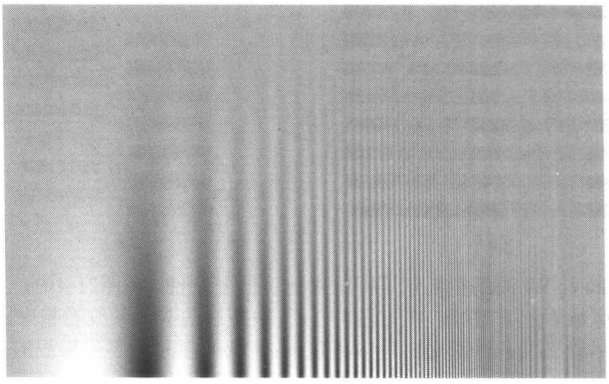


图21-2 条纹之间的对比度以固定模式从上到下逐渐增加，而可见度则随频率变化

其中α取决于表面几何形状、入射照明模式和查看方向之间的关系。虽然人眼只能直接测量L，但是人的视觉对R的估计要比L好得多。为了阐明这一点，请在直射量光下查看图21-3。用手将模式之一遮起来，而另一个处于直接光照下。虽然两个模式反射的光线将明显不同，但是两个中心方块的外观亮度看起来几乎一样。常用术语光亮度（lightness）来描述表面外观上的亮度，就是为了将其与实际亮度区别开来。在很多情况下，光亮度对大的照明变化都是不变的。该现象被称为光亮度不变性（lightness constancy）。

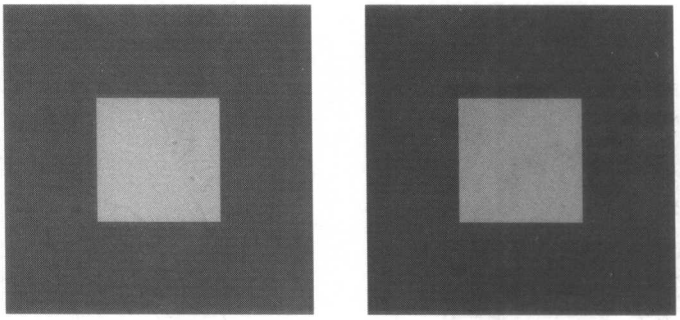
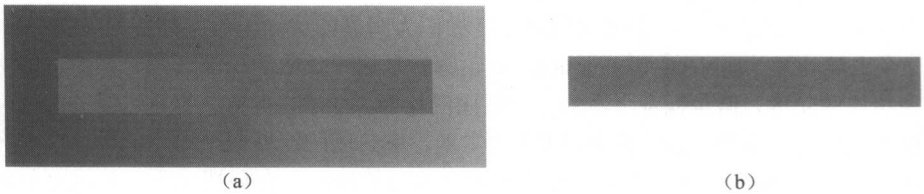


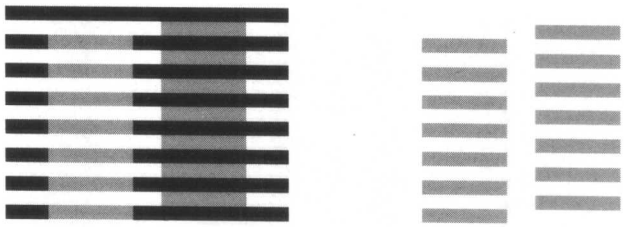
图21-3 光亮度不变性。用手投射一个阴影遮住上面一个模式，将发现两个中心方块的外观亮度几乎仍然一样

人的视觉系统实现光亮度不变性的机制还不清楚。如图21-2所示，视觉系统对缓慢变化光模式的感觉要相对迟钝一些。这可能是为了抵消光照缓慢变化的效果。外观亮度受周围环绕区域亮度的影响（见图21-4）。当区域光照不同时，这有助于理解光亮度不变性。虽然通常将即时对比度（simultaneous contrast）看作在环绕区域对比亮度的基础上对感知亮度的一种修改，但是实际上它比图21-5和图21-6所示的情况更为复杂。关于光亮度感知的详细情况，请参阅文献（Gilchrist et al., 1999）和（Adelson, 1999）。 481



（a） （b）

图21-4 （a）即时对比度：中间横条的外观亮度受周围区域亮度的影响；（b）不带可变环绕的同一横条



（a）　（b）

图21-5　Munker-White幻觉揭示了即时对比度的复杂性。在图21-4中，当周围区域较暗时，中间区域看起来更亮。在本图的（a）中左边的灰色条纹看起来比右边的灰色条纹更亮，尽管它们几乎都是白色区域包围；（b）显示了不带黑线的灰色条纹

虽然视觉系统在很大程度上忽略慢变的亮度模式，但是对于由亮度不连续的线条构成的边缘（edge）是极其敏感的。在光强度图像中，边缘通常对应着表面边界或者是环境中其他重要特征（见图21-7）。视觉
482 系统还能检测运动中的局部差异、立体视差现象、纹理以及其他图像性质。然而，如果没有上述其他性质差异伴随的话，视觉系统很难检测空间颜色的不连续。

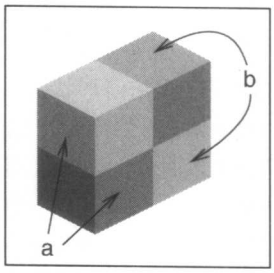
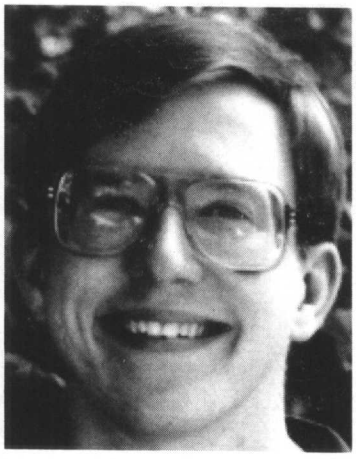


图21-6　亮度感知受三维结构感知的影响。两个标记为a的表面具有同样的亮度，标记为b的表面也是如此（摘自Adelson（1999））



（a）



（b）

图21-7　（a）原始的灰度图像；（b）图像边缘，在某些方向上边缘是空间高度变化的线条

边缘感知好像与形状感知结合在一起。虽然边缘给了视觉系统识别形状所需要的信息，但是如果边缘产生了更为完整的形状，缓慢变化的亮度将作为明显的边缘出现（如图21-8所示）。图21-9显示了一个主观轮廓（subjective contour）。该轮廓就是这一效应的极端形式。虽然在实际的图像中不存在这样的轮廓，但是视觉上看到一个封闭的轮廓。最后，视觉系统对边缘的敏感性好像还是亮度感知机制的一部分。请注意图21-9中由主观轮廓封闭的区域似乎要比该页面周围区域
483 稍微明亮一些。图21-10显示了另一种边缘与亮度相互作用的情况。在该情况下，边缘处的特定亮度轮廓对边缘两边表面的外观亮度产生了巨大影响。

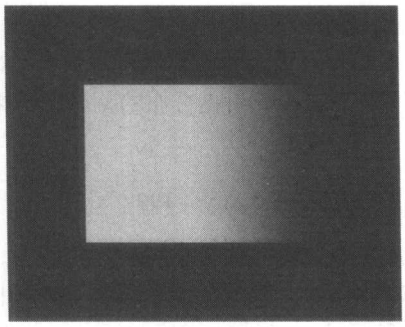


图21-8 有时即使不存在明显的亮度中断，视觉系统也能看到“边缘”。在本图中心模式的右边就出现了这一情况

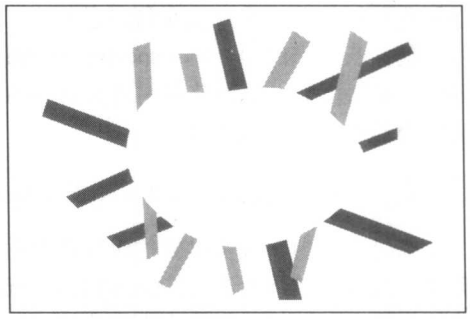


图21-9 有时视觉系统将看到主观轮廓，而无需任何相关亮度的改变

如上所述，如果两个相邻区域之间的亮度差别至少为平均亮度的1%，人能够检测出这种差别。这就是Weber定律（Weber's law）的例证。该定律指出：在一个刺激中刚好可察觉的差异（just noticeable difference, jnd）与该刺激的大小之间的比值是一个常量，即

$$\frac{\Delta I}{I} = k_1 \qquad (21\text{-}2)$$

其中 I 为刺激的大小，ΔI 是刚好可察觉的差值大小，k_1 为特定于该刺激的常量。Weber定律是在1846年提出来的，至今仍然是很多感知效果的有效表征。Fechner定律（Fencher's law），于1860年提出，将Weber定律一般化了，允许描述任何感官体验的强度，而不仅仅是jnd：

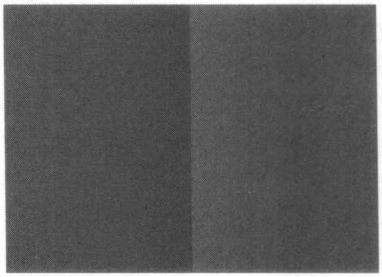


图21-10 感觉的亮度更多地依赖于边缘处的局部对比度，而不是表面之间的亮度。请试着用铅笔盖住图中中间位置的垂直边缘看看。该图是Craik-O'Brien-Cornweet幻觉之一

$$S = k_2 \log(I) \qquad (21\text{-}3)$$

其中 S 为感官体验的感知强度，I 为相应刺激的物理大小，k_2 为特定于该刺激的缩放常量。目前的实际做法是采用指数函数（Steven定律），对刺激感知强度与实际强度之间的联系建模：

$$S = k_3 I^b \qquad (21\text{-}4)$$

其中 S 和 I 与前面相同，k_3 为另一个缩放常量，b 为特定于该刺激的指数。对于大量涉及视觉的感知量，b 都小于1。在别处描述的CIE L*a*b*颜色空间，就采用了修改的Steven定律表示形式来刻画亮度值之间的感觉差异。注意在前两个定律对刺激感知强度的刻画中以及Steven定律中 $b<1$ 的情况下，如果刺激的平均值很小的话，刺激变化引起的感知效果要大于刺激值很大时同样的物理变化所产生的效果。 484

上述定律并不是对感知操作的物理约束。相反，它们是有关感知系统响应特定物理刺激的一般规律。在感知心理学领域，物理刺激与感知效果之间量化关系的研究被称为*心理物理学*（psychophysics）。虽然心理物理学的定律是以经验为主的观察推导，而不是机械的计算，但是如此多的感知效果都通过简单的指数函数进行了很好的建模，却是令人震惊的事情。这也许能够帮助我们深入洞察其内在的机理。

21.2.2 颜色

在1666年，牛顿使用棱镜明显地展示了白色的太阳光可以分解成彩色光谱（spectrum），而且

这些彩色可以重新结合产生呈现白色的光。我们现在已经知道光能是由光子组成的，每个光子都有特定的波长。光的**谱分布**（spectral distribution）就是各个波长上光的平均能量的测量。对于自然照明，表面反射光的谱分布因表面材质的不同而有明显区别。因此对谱分布的刻画能够提供环境表面自然性质的可视信息。

颜色视觉的研究历史因其尖刻而显得不平凡。——Richard Gregory（1997）

大部分人在观察现实世界的时候都有普遍的颜色感觉。颜色感知依赖于光的频率分布。人的可见光谱的变化范围大约在370 nm到730 nm之间（见彩图 XVI）。视觉系统从谱分布推导出颜色感觉的方式，早在1801年就有人首次系统地分析过，在随后的150年中一直争论不休。问题在于视觉系统对谱分布模式的响应完全不同于亮度分布模式。

即使考虑光亮度不变性之类的现象，明显不同的空间分布看起来几乎总是明显不同的。更为重要的是，假设视觉系统的目的是根据近侧刺激产生远侧刺激的描述，那么感觉到的光亮度模式至少大致符合环境中表面的光亮度模式。但是颜色感知却不是这样。很多差别很大的光谱分布都能产生任何一种具体颜色的感觉。因此，某个表面为某种颜色的感觉，只提供了很少量的与该表面的光谱分布直接有关的信息。例如，假设谱分布由波长为700 nm和540 nm的光组合而成，通过选择适当的相对强度，看起来将与单一波长为580 nm的几乎没什么区别。（感觉上不可区分而谱
485 成分却不同的颜色，被称为**条件等色**（metamer））。我们看到“黄”色时，我们并不知道该颜色是由哪种谱分布产生的，它可能是由这些谱分布中某种或者是另一种产生的，也可能是由别的无穷的谱分布家族产生的。因此，在视觉上下文关系中，术语**颜色**（color）仅仅是指一种纯粹的感觉质量，而不是一种物理性质。

在人的视网膜上有两类光感受器。**视锥**（cone）涉及到颜色感知，而视网膜内的**视杆**（rod）则对可见光范围内的光能敏感，并不提供关于颜色的信息。视锥分为三种，每种都有不同的谱灵敏度（见图21-11）。*S* 视锥对应于可见光谱中蓝色区域的短波长。*M*视锥对应的波长处于可见光谱中间呈绿色的区域。*L*视锥对应的波长更长一些，包括可见光谱的绿色和红色部分。

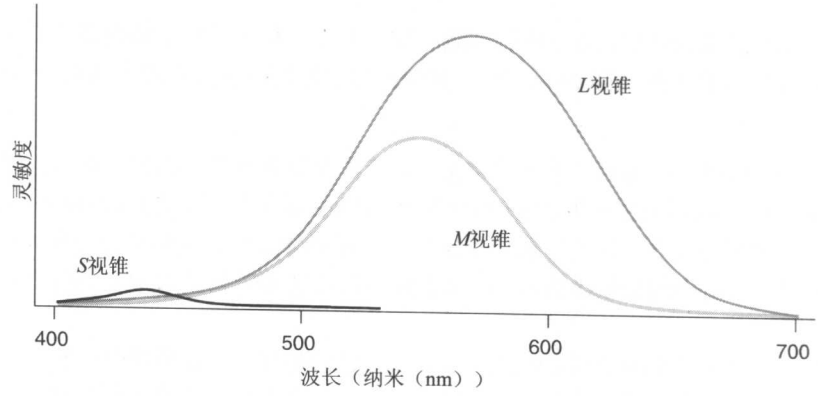


图21-11　视网膜中短、中、长视锥的谱灵敏度

虽然通常将这三种视锥描述为红、绿、蓝，但这并不是准确的术语，也没有精确地反映图21-11所示的视锥灵敏度。*L*视锥和*M*视锥是宽带调谐的，意味着它们能够响应较宽的频率范围。三种视锥的灵敏度曲线之间还有部分重叠。将这两种属性综合起来，那就意味着不太可能重构一个逼近
486 函数来模拟这三种视锥响应给出的原始谱分布。这与视网膜（和数码相机）中的空间采样是矛盾的。在空间采样中，为了能够检测到局部对比的细微情况，感受器在空间灵敏度上是窄带调谐的。

事实上，人的视网膜中只有三种颜色敏感的光感受器，极大地简化了在计算机监视器和图形显示器上显示颜色的任务。计算机图形显示器显示的颜色为三种固定颜色分布的加权组合。通常，这三种颜色为截然不同的红、绿、蓝。因此，在计算机图形学中，颜色通常被表示为红绿蓝（red-green-blue,RGB）三元组，分别代表显示某种特定颜色所需要的红绿蓝三原色的强度。三种基色（basic color）足以显示大部分可知觉的颜色，因为通过三种适当选择的颜色的适当加权组合，能够产生这些可知觉的颜色的条件等色。

使用RGB表示颜色至少有两个问题。第一个问题是，不同监视器上红绿蓝三原色的谱分布不同。这样，感觉上正确的颜色再现就涉及到每个监视器上RGB值重新映射。这当然只有RGB值满足某些良定义的标准时才有可能，而实际上往往并非如此。有关该问题的详细情况见第20章。第二个问题是，RGB值没有按照与主观感知一致的方式定义颜色。在看到“黄”色的时候，并没有感觉到它是由等量的红光和绿光组成的。相反，它看起来更像一种单一色，带有一些附加的涉及到亮度和颜色“量”的性质。将颜色表示为*S*视锥、*M*视锥和*L*视锥的输出也没有什么帮助，因为用这些属性来刻画颜色与用RGB显示性质来刻画相比，并不能获得更多现象学上的意义。

有两种不同的颜色刻画方法更接近于反映人的感知。各种不同CIE颜色空间的目标是“感觉均匀”，以便两种颜色表示值之间的差值大小与感觉到的颜色差别成正比（Wyszzecki & Stiles, 1992）。这已经证明是一个很难实现的目标。因此这些年来提出了几个修改的CIE模型。此外，虽然CIE颜色空间的一个维度对应于感觉的亮度，但是其他两个指定色度的维度并没有什么直觉意义。

第二种方法以更自然的方式刻画颜色。该方法源于以下观察：有三种截然不同的独立性质支配着颜色的主观感觉。光亮度（lightness）（表面的外观亮度）已经讨论过了。饱和度（saturation）指颜色的纯度或者是鲜亮程度。颜色可以从完全不饱和的灰色变化到部分饱和的清淡色彩，乃至 487
完全饱和的“纯”色。第三个性质为色调（hue），与“颜色”这一词的非正式感觉最接近，是以类似于可见光谱中颜色的方式来刻画的，从黑紫变化到黑红。彩图XVII显示了色调-饱和度-亮度（Hue-Saturation-Lightness，HSV）颜色空间图。因为亮度与光亮度之间的关系既复杂又不太明白，因此HSV颜色空间几乎一直采用亮度，而不是试图估计光亮度。然而，与光谱中的波长不同的是，通常色调的描述反映了可见光谱的两端实际上在外观上类似的事实（见彩图 XVIII）。在特定颜色值的RGB和HSV表示之间存在简单的变换。因此，虽然HSV颜色空间是基于感觉考虑而提出来的，但是包含的信息并不比RGB多。

色度-饱和度-光亮度描述颜色的方法建立在单点谱分布的基础上，因此对于分布在空间的光谱分布，只能近似其感觉上的响应。颜色感知也有类似于亮度/明度的不变性以及即时对比效应。不过两者都不能以RGB表示方式来捕捉。同样HSV表示方式也不行。对于颜色不变性的例子，可以分别在室内的白炽灯下和室外的日光下查看一张白纸。在两种情况下，该纸张看起来都将是“白”的，虽然白炽灯有明显的黄色调，使得纸张反射的光也呈黄色调，而日光具有更均匀的彩色光谱。

另一个不能由CIE颜色空间或者是HSV编码捕捉的颜色感知状况是，在查看连续可见光谱（彩图 XVI）或者是自然出现的彩虹的时候，我们只能看到少量的明显不同的颜色。对大多数人而言，可见光谱好像被分成4到6种不同的颜色：红、黄、绿、蓝，外加可能的淡蓝和紫。再加上非光谱颜色，在英语中通常使用的基本颜色术语也只有11个，即red、green、blue、yellow、black、white、gray、orange、purple、brown、pink。将内在连续的谱分布空间分割成相对较小的感觉类别集合，将其与定义好的语言术语关联起来，好像是感知的基本性质，而不仅仅是一个文化上的人工制品（Berlin & Kay，1969）。然而，该处理的确切性质并不明白。

21.2.3 动态范围

自然照明强度的变化范围超过了6个数量级大小（见图21-12）。人的视觉系统能够在整个亮 488

度级别范围内起作用。然而，在任何一个时间点上，视觉系统只能检测在非常小的范围内的变化的光强度。因为视觉系统所处环境的平均亮度随时变化，因此可区分的亮度范围也以相应的方式改变。如果从光亮的室外迅速进入一个很暗的房间，这种效应将是最明显的。最初，我们看不到任何东西，但是过一会儿之后，房间里面的细节开始清晰起来。所出现的暗适应（dark adaptation）涉及眼睛内部很多生理学上的变化。有效暗适应的发生需要花费几分钟的时间，完全暗适应则需要大约40分钟。如果再返回到明亮光照下，不仅视觉困难而且实际上还会有疼痛感。在眼睛能够再次看清楚之前，需要进行光适应（light adaptation）。光适应的发生要比暗适应快很多，通常需要的时间不到1分钟。

直射日光	10^5
室内光照	10^2
月光	10^{-1}
星光	10^{-3}

图21-12　白色表面在不同光照条件下的近似亮度级别，单位为cd/m^2（Wandell，1995）

在人的视网膜中有两类光感受器对不同的亮度范围敏感。视锥提供大部分正常光照条件下的可视信息，其范围可从明亮的日光变化到暗淡的室内照明。视杆只在非常低光照级别下才有效。适应光的（photopic）视觉涉及亮光，其中只有视锥有效；而暗适应（Scotopic）视觉则涉及暗光，其中只有视杆起作用。对于视锥和视杆都有一个光强度范围，只有在该范围内才对光的变化敏感。这就是所谓的过渡视觉（mesopic）条件（见第22章）。

21.2.4　视域和灵敏度

在人的视觉系统中，每只眼睛的视域大约为水平160°、垂直135°。双眼并用时，两只眼睛的视域只有部分重叠。这样总的视域就加宽了，大约为水平200°、垂直135°。两眼重叠的区域大约在水平120°、垂直135°的范围内。

对于正常视觉或者是校正到正常的视觉，通常会有这样一种主观体验，即无论在什么情况下都能相对地看到细节。然而，这实际上是一种幻觉。在每只眼睛的视域中，只有很小的一部分是真正对细节敏感的。为了说明这一点，请按照图21-13所示，拿着一张写有一般大小文字的纸，保持在手臂能伸到的地方。然后用另一只手遮住一只眼睛，让另一只眼睛盯住大拇指不动，你将注意到大拇指上方的文字立即清晰可见，而两边的则看不清。高灵敏度视觉被限制在一个比伸直
489 手臂上的拇指稍微宽一点的视角范围内。我们通常注意不到这一点是因为眼睛经常不断移动，使得视域中的不同区域都以高分辨率被查看。然后视觉系统将不同时刻的信息融合起来，产生一种主观体验，好像整个视域都是以高分辨率查看到的。

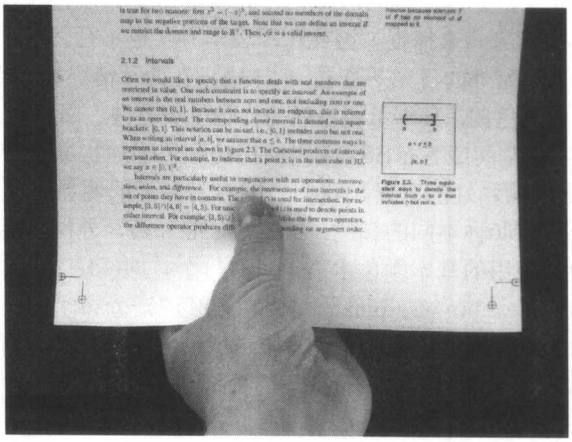


图21-13　如果举着一页文本，伸直手臂并盯着大拇指看，你将发现只有大拇指附近的那些文本是清楚的（摄影Peter Shirley）

在人的视觉皮层中没有足够的带宽，难以处理整个视网膜图像稠密采样所产生的信息。因此，在视网膜中包装了各种不同密度的光感受器，并与指向感兴趣区域的快速的眼动机制结合起来，提供了一种即时优化灵敏度和视域的方式。其他动物已经进化了各种不依赖于快速眼动的灵敏度与视域平衡方式。有的只拥有高敏度视觉，但是视域却很窄。有的则视域宽广，但是看见细节的能力有限。

通过眼睛运动将环境中感兴趣的区域聚焦于视网膜中央凹陷处的过程称为*扫视*（saccade）。扫视的呈现非常迅速。从开始触发到完成眼睛运动的时间大约为150～200 ms。这一时间的大部分都花在对扫视的规划上。实际运动过程平均只需20 ms左右。扫视期间眼睛运动的速度非常快，通常最大旋转速度超过500°/s。在两次扫视之间，视点指向一个感兴趣的区域（*凝视*（fixate）），花费300 ms左右的时间获取可视细节信息。将多次凝视融合起来形成宽广视域上整体细节的主观感觉的机制仍不清楚。

图21-14展示了人眼视网膜中视锥和视杆的分布密度变化图。视锥，负责正常光照条件下的视觉，主要群集在视网膜的中央凹陷处（见图21-14）。当眼睛凝视环境中某一特定点时，该点的图像就落在该凹处。中央凹处视锥群集密度越高，光照图像（见第14章）的采样率也就越高，因此在采样模式中就包含更多的细节。视网膜中央凹视力环绕大约1.7°的范围。这正好是手臂能够到的距离上一个拇指宽度所占据的视角范围。

490

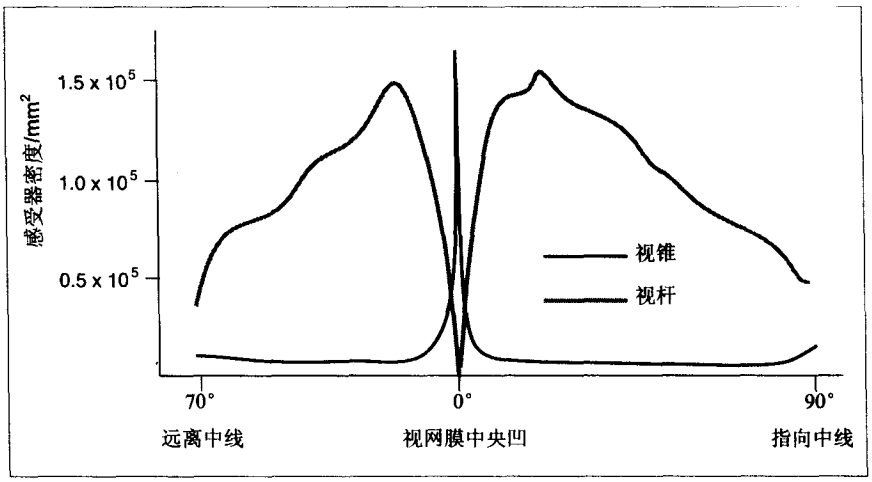


图21-14 人眼视网膜中视杆和视锥的分布密度（选自（Osterbrg，1935））

虽然在大多数介绍人眼视觉感知的教材中都出现了图21-14的版本，但是该图只提供了神经生理学视觉灵敏度的部分解释。单个视锥和视杆的输出通过眼内神经连接以各种不同方式汇聚在一起，然后沿着视神经传输到视觉皮层[1]。这一集中过程将过滤入射照明模式提供的信号，其中的滤波方式对可检测的光模式有重要影响。特别地，离视网膜中央凹越远对亮度进行平均的区域就越大。其结果是，空间灵敏度急剧下降。大多数显示视锥和视杆分布密度的图都指出了视网膜盲点（blind spot）的位置。盲点是将视觉信息从眼睛输送到大脑的神经束穿过视网膜的地方，该点对光不敏感。总的来说，盲点对现实世界感知的唯一实际影响是在介绍感知的教材中将其当作幻影使用，因为正常的眼睛运动补偿了时域信息损失。

491

如图21-14所示，视杆的聚集密度在视网膜中央凹的中间下降为零。而在远离该凹处的方向

1. 视神经中所有细胞以及视觉皮层中几乎所有的细胞都有一个关联的视网膜感受域。照射在视网膜上细胞感受域外的光模式对该细胞的冲动率没有任何影响。

上，视杆密度先增后减。其结果之一是，当照明很低时没有视网膜中央凹视力。视网膜中央凹缺乏视杆，在一个没有月亮的晚上远离城市灯光观察天空时，可以证明这一点。在观察时，你将发现有些星星非常暗淡，如果将视点稍微移到该星星旁边的天空时，则可以看见该星。但是如果直接看着它们，则又消失了。出现这一现象是因为直接看这些特征的时候，特征图像只落在视网膜的视锥上，而视锥在检测特征时对光并不十分敏感。稍微往旁边看，将使图像落在对光更敏感的视锥上。暗视觉在灵敏度方面也是有限的，一部分原因是视网膜大部分区域的视杆密度较低，另一部分原因是视网膜中汇聚的视杆信号量增加了，以便增加传给大脑的视觉信息的光敏感度。

21.2.5　运动

在阅读印在纸上的视觉感知文字材料以及静态图片时，很容易忘记运动在视觉体验中是无所不在的。由于眼睛和肢体的运动，以及现实世界目标的移动，落在视网膜上的光模式是不断变化的。本节将介绍检测视觉运动的能力。21.3.4节将介绍如何使用视觉运动确定环境的几何信息。21.4.3节将介绍如何使用该运动引导我们在环境中的移动。

视网膜上特定光模式的运动检测能力是一个依赖于速度、方向、模式大小和对比度的复杂函数。另外，在运动感知中出现类似于亮度感知的即时对比效应，使得该问题进一步复杂化。极端的情况是，一个单一的细小模式沿着同质的反衬背景移动，为了感知该运动，要求有相当于每秒0.2°～0.3°视角的运动速率。而同样的模式在有纹理的背景上运动，则需要大约十分之一的速度就可以检测到。

利用视网膜的运动敏感性，结合扫视时眼睛运动的频率和速度，就会惊奇地发现我们所查看的世界通常是稳定的。视觉系统通过三种方法成功地完成这一任务。在扫视期间降低对比敏感度，减少眼睛位置快速变化所产生的视觉影响。在扫视之间，会出现各种不同复杂程度的眼睛位置调
492 节机制，补偿头和肢体的运动以及现实世界中感兴趣目标的运动。最后，视觉系统使用有关眼睛位置的信息，将多个凝视所产生的小片高分辨率画像组装成一个单一的整体。

直线和边缘的运动，如果没有端点和拐角的出现，将是含糊不清的。这一现象被称为孔径问题（aperture problem）（见图21-15）。孔径问题的原因是平行于直线或边缘的运动没有产生任何视觉变化。真实世界的几何形状是十分复杂的，因此在实践中，除了那些故意的类似于理发店招牌的错觉外，很少因这一点而产生困难。然而，在一些计算机图形透视图中发现的简化几何和纹理，可能会使感知到的运动不精确。

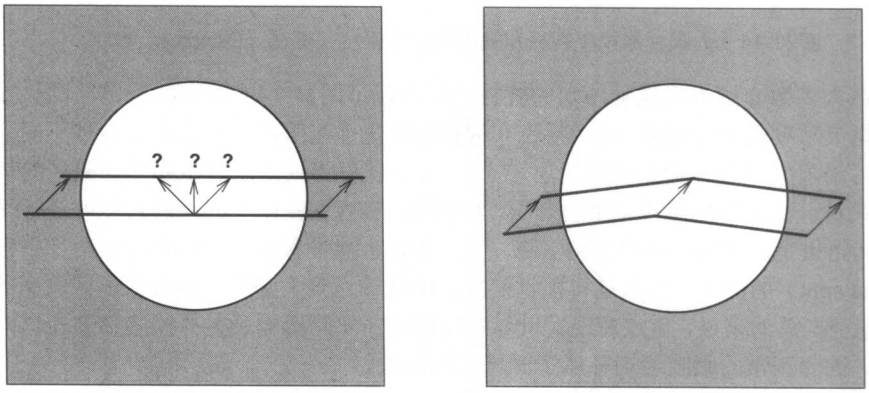


图21-15　孔径问题：（a）如果直线或边缘将端点隐藏起来移动，那么视觉信息将不足以确定该直线事实上的运动；（b）如果线条上存在拐角或者其他明显标记，那么线条的二维运动就是明确的

实时的计算机图形、电影和视频，如果没有“不连续运动”这一重要的感知现象，将是不可能的。不连续运动是指一系列静态图像在离散的时间间隔内可见，然后按照离散的空间间隔移动这些图像，结果形成一种与连续运动几乎不可区分的效果。这一效果被称为视动（apparent motion），以强调该连续运动的外表实际上是一个幻觉。

图21-16阐述了连续运动与视动之间的区别。连续运动是真实世界中的典型情况，而视动则基于存在于所有的动态图像显示设备中。图21-16b所绘制的运动包含一个与图21-16a所示运动相当的平均运动。该平均运动经过了一个高频时空调制。该调制源自两种模式之间的切换，即平稳模式和不连续地移到新位置的模式。出现连续运动的外观感觉是因为视觉系统对运动的高频分量不敏感。 493

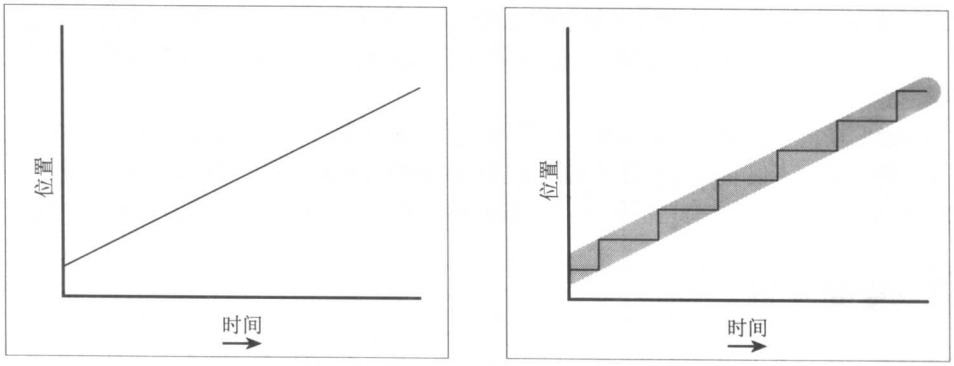


图21-16 （a）连续运动；（b）平均速度相同的不连续运动。在某些情况下，这两种运动模式的感知结果可能是类似的

当单幅图像出现频率超过10 Hz时，只要相继图像之间的位置变化不太大，就会出现强制性的视动感觉。然而对于大多数显示设备而言，该速率还不足以产生令人满意的连续运动感觉。几乎所有这样的设备都会在图像切换到下一幅时引入明度变化。在良好光照条件下，人的视觉系统对这类亮度变化非常敏感，即使变化速率高达80 Hz也能探测到。在较低光照条件下，这种探测能力大约为40 Hz。当改变亮度的速率足够高时，就会出现闪烁融合（flicker fusion），这种变化就再也看不见了。

为了产生强制性的视动感觉，图像显示必须满足两个独立的约束：

- 图像更新速率大于等于10 Hz。
- 图像更新过程中引入的闪烁速率必须大于等于60～80 Hz。

解决办法之一就是要求图像更新速率大于或者等于60～80 Hz。然而，在很多情况下，这都是不可能的。对于计算机图形显示设备，帧的计算时间实质上通常大于12～15毫秒。由于传输带宽以及较老的监视器技术的限制，正常的广播电视被限制为每秒25～30帧图像。（一些HDTV格式运行在60帧/秒）。电影图像的更新速率为24帧/秒，这主要是受曝光时间以及胶片物理运动的机械困难的限制。 494

不同的显示技术可以不同的方式解决这一问题。不管图像内容改变的频率是多少，计算机显示器都以大约70～80 Hz的速率刷新显示的图像。对于这样的显示器，术语帧率（frame rate）有些模糊。因为需要两个术语来刻画该显示器：一个是刷新速率（refresh rate），指出图像被重新显示的速率；另一个是帧更新速率（frame update rate），指出产生新的显示图像的速率。标准的非HDTV广播电视采用60 Hz（NTSC，在北美和其他一些地区使用的标准）或者是50 Hz（PAL，在北美以外的大多数地区使用的标准）的刷新速率。帧更新速率为刷新速率的一半。该显示是隔行

扫描的（interlaced），而不是每幅新图像都显示两次。隔行扫描将交替的图像水平线分割成偶数场和奇数场（field），然后交替地显示偶数场和奇数场。在电影中为了避免闪烁，采用了机械快门，使胶片中的每一帧在移到下一帧之前都闪烁三次，从而产生72 Hz的刷新速率，将帧更新速率维持在24 Hz。

使用视动来模拟连续运动偶尔会产生令人不快的失真。最著名的失真是*货车车轮幻觉*（wagon wheel illusion）。在幻觉中，旋转车轮的轮辐看起来旋转的方向好像与传统车轮运动的预期方向相反。货车车轮幻觉是时域混淆的例子之一。轮辐，或者是其他旋转盘片上空间周期变化的模式，对于相对车轮或盘片中心固定的查看位置而言，将产生时域周期信号。固定帧更新速率就相当于在时间上对这一时域周期信号进行采样。如果被采样模式的时间频率太高，那么采样频率的不足将导致图像显示时的混淆以及时间频率降低的现象。在某些情况下，时间频率的失真会导致空间失真，即车轮看起来好像向后运动。电影要比电视视频更容易出现货车车轮幻觉，因为电影的时间采样率更低。

视动影像在从一种媒体转换到另一种媒体时也可能出现问题，特别是在将24 Hz的电影转换为电视视频的时候。不仅非隔行扫描的格式需要转换成隔行扫描格式，而且还没有直接将24帧/秒转换为50或60场/秒的方法。有些高端显示设备能够部分补偿将电影转换为视频而引入的失真。

21.3　空间视觉

495 视觉系统执行的关键操作之一就是估计可视环境的几何性质，因为这些性质是确定有关目标、位置和事件的重要信息。有时将视觉描述为*逆向光学*（inverse optics），是为了强调视觉系统使图像形成过程颠倒的功能。这一逆向颠倒是为了确定在视网膜上产生特定光模式的现实世界的几何形状、材质和光照情况。视觉系统的主要问题是，可视环境的性质在视网膜的光模式图像中混淆不清。亮度是照明和反射率的函数。由于光传输的复杂性，亮度可能依赖于空间大片区域的环境性质。环境场所在图像中的投影区域最多可以将该区域的位置限定为半条直线。因此，几乎不可能唯一确定产生特定光模式图像的自然性质。

确定*表面布局*（surface layout）（环境中可视表面的位置和方向）被认为是人眼视觉处理的关键步骤。在有关视觉系统从所接受的光模式中提取表面布局信息的讨论中，大部分都将该问题划分成一组*视觉线索*（visual cue），每个线索都描述一种特定的视觉模式。视觉模式结合所需要的推理规则，可以推导出表面布局的性质。因为仅从单一的视觉很难精确确定表面布局，因此表面布局的推理通常需要额外的非视觉信息。这些信息可以来自其他感觉，或者是真实世界中可能出现的有关物体的假设。

视觉线索通常被分成四类。*眼球运动线索*（Ocularmotor cue）包含有关眼睛位置和焦点的信息。*视差线索*（diaparity cue）包含双眼查看同一表面时提取的信息，那些从眼睛定位中可以获取的信息除外。*运动线索*（motion cue）提供那些由观察者或者是目标的运动产生的信息。*绘画线索*（Pictorial cue）源自三维表面形状到视网膜上二维光模式的投影。本节将介绍与表面孤立点几何信息提取有关的视觉线索。更一般的位置和形状信息的提取将在21.4节介绍。

21.3.1　参考坐标系和测量尺度

描述可视表面上点的位置和方向，必须在特定参考坐标系上下文中进行。该参考坐标系将指定用来表示几何信息的坐标系的原点、方向和缩放比例。人类视觉系统使用多个参考坐标系，其中部分原因在于不同视觉线索中可用信息的种类不一样，另一部分原因是各类信息的使用目的不
496 同（Klatzky，1998）。*以自我为中心的*（egocentric）表示是相对于观察者的肢体而言的。*以环境*

为中心（allocentric）表示，也称外中心（exocentric）表示，是相对于观察者外部某个参考物体来定义的。以环境为中心参考坐标系可以是局部环境目标的某种配置，也可以按照独特的位置、重力或地理性质进行全局指定。

从观察者到环境中某个特定可见位置的距离，如果采用以自我为中心的表示，那么在感知文献中，通常称为深度（depth）。表面方向既可以采用自我中心表示，也可以采用以环境为中心坐标。在方向的以自我为中心表示中，涉及视线和表面法线之间的角度时，采用术语倾斜度（slant）来表示，而术语坡度（tilt）则表示表面法线在垂直于视线的平面上的投影方向。

距离和方向都可以采用不同的测量尺度（measurement scale）来表示。绝对（absolute）描述按照标准来指定，该标准本身并不是感觉信息的组成部分。这些尺度可以是明文确定的标准（例如米），也可以是相对于观察者肢体而言的标准（例如，眼睛的高度，肩膀的宽度）。相对（relative）描述将感觉到的几何性质与另一个性质关联起来（例如a点比b点远两倍）。顺序（ordinal）描述 497
是相对测量的一种特殊情况。在该描述中表达的只是关系的符号，而不是大小。图21-17中列举了最常见的视觉线索，以及这些线索可能提供的信息种类的表示特征。

线　索	a	r	o	绝对深度的需求
适应性调节	x	x	x	非常有限的范围
双目会聚	x	x	x	有限范围
双目视差	-	x	x	有限范围
直线透视图、图片高度、水平比率	x	x	x	要求视点高度
惯用大小	x	x	x	
相对大小	-	x	x	
空间透视	?	x	x	适应本地条件
绝对运动视差	?	x	x	需要视点速度
相对运动视差	-	-	x	
纹理梯度	-	x	-	
明暗	-	x	-	
遮挡	-	-	x	

图21-17 常见视觉线索：a表示绝对深度，r表示相对深度，o表示顺序深度

21.3.2 眼球运动线索

有关深度的眼球运动信息直接来源于眼睛的肌肉控制。有两种截然不同的眼球运动信息。适应性调节（accommodation）是指眼睛光学聚焦于某一特定距离的过程。会聚（convergence）（通常也称为聚散（vergence））则指两眼指向三维空间中同一个点的过程。适应性调节和会聚都有潜力提供绝对深度信息。

在生理学上，人眼聚焦是通过眼睛前端透镜的变形来实现的。视觉系统能够根据变形量推导出深度。适应性调节是相对比较弱的距离线索，在2 m之外就无效了。大部分人在45岁以后聚焦一定距离范围的难度就逐渐增大。对于他们而言，适应性调节的作用越来越小。

那些不熟悉视觉感知特性的人，有时会将适应性调节估计的深度信息与由模糊引起的深度信 498
息混淆起来，其中模糊的产生与有限的景深（depth-of-field）有关。适应性调节深度线索提供视域中焦点对准的那一部分的距离信息。该信息不依赖于视域中其他部分的模糊程度，除了模糊被视觉系统用来调节焦点之外。尽管对这一效果的研究还有限，但是景深好像确实没有提供某种程度上的顺序深度信息（见图21-18）。

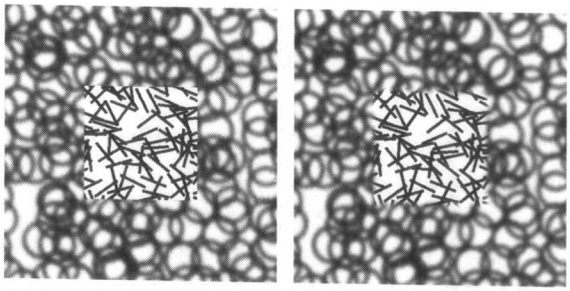


图21-18　中间的方块是出现在圆圈图案的前面，还是通过圆圈图案上的一个方孔看到中间的方块？两幅图像唯一的区别在于线条图案和圆圈之间边缘的清晰度不一样（Marshall, Burbeck, Arely, Rolland, and Martin，1999），授权使用）

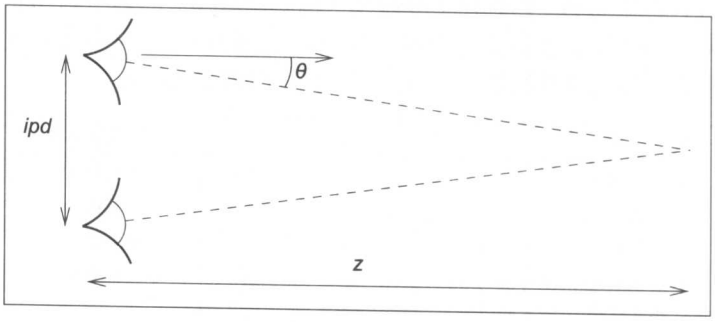


图21-19　两眼的聚散（vergence）提供凝视点的距离信息

如果两眼凝视空间同一点，则可以使用三角法确定从观察者到查看位置的距离（如图21-19所示）。最简单的情况是，感兴趣的点直接位于查看者的前方，则有：

$$z = \frac{ipd/2}{\tan\theta} \tag{21-5}$$

其中z是到现实世界中某点的距离，*ipd*为瞳孔距离（interpupillary distance），即两眼之间的距离，θ为聚散角（vergence angle），指的是眼睛相对于直前方的方向。对于小的θ（眼睛的配置正是这种情况），当用弧度表示时，有$\tan\theta = \theta$。这样，聚散角的微分就按照下式指定了距离差：

$$\Delta\theta \approx \frac{ipd}{2} \cdot \frac{1}{\Delta z} \tag{21-6}$$

当θ以均匀步长趋近于零时，Δz将变得无穷大。这意味着立体视觉在整个深度增加时对深度变化不太敏感。会聚实际上只提供几米距离内的绝对深度信息。在此范围之外，距离改变所产生的聚散角的变化将小到不起作用。

499

在人的视觉系统中，适应性调节和会聚是相互作用的：适应性调节用来帮助确定合适的聚散角，而聚散角可用来帮助焦距的设置。通常，在存在不确定性的时候，这能够帮助视觉系统确定适用性调节或者是聚散。然而，立体照相的计算机显示打破了真实世界中焦点和会聚之间的关系，结果导致很多感知困难（Wann, Rushton, & Mon-Williams, 1995）。

21.3.3　双目视差

眼睛凝视空间某个普通点时的聚散角，只是视觉系统根据双目立体视觉确定深度的方式之一。第二种机制涉及双目视网膜图像的比较，不要求眼睛指向哪个方向的信息。一个简单的例子就可以证明这一效果。将手臂向前伸直，大拇指向上。凝视大拇指，然后闭上一只眼睛。现在，同

时睁开闭上的眼睛，并闭上睁开的眼睛。大拇指差不多是静止的，但是大拇指后面较远处的表面则看起来忽左忽右（见图21-20）。场景中的点在左右眼视网膜上位置的变化称为视差（disparity）。



左眼图像　　右眼图像

图21-20 双目视差。从左眼到右眼的视图显示了深度不同于凝视点的表面点的偏移。Peter Shirley提供

双目视差线索要求视觉系统能够将一只眼中的点图像与其在另一只眼中的成像位置匹配起来。这一匹配过程称为对应问题（correspondence problem）。这是一个比较复杂的问题，只有部分情况被人所了解。一旦建立了对应关系，现实空间中的特定点在左右视网膜上的相对位置，就可以指出这些点是离凝视点更近还是更远。当对应点处于视网膜中央凹外时，就会出现交叉视差（crossed disparity），显示出表面点比凝视点更近；当对应点处于视网膜中央凹内，显示表面点与凝视点更远时，就会出现不交差的视差（uncrossed disparity）（见图21-21）[1]。双目视差是一种相对深度线索，但是通过会聚伸缩可以提供绝对深度信息。式（21-5）适用于双目视差以及双目会聚。跟会聚一样，双目视差对深度变化的敏感度也随深度递减。

500

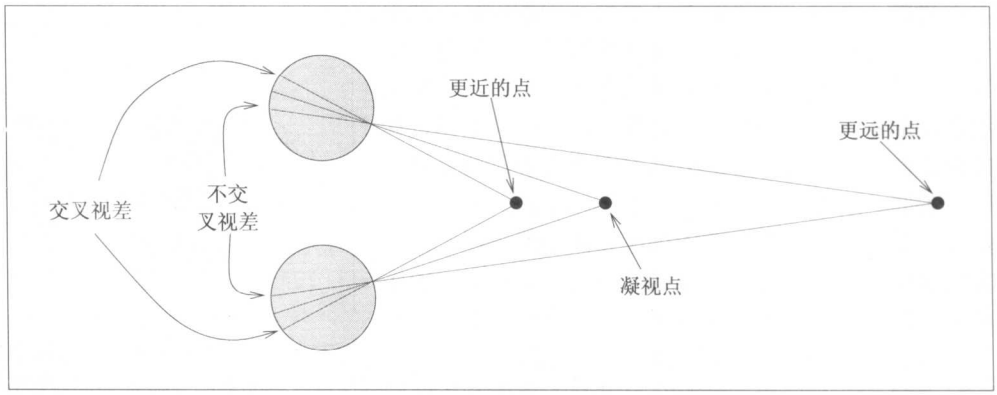


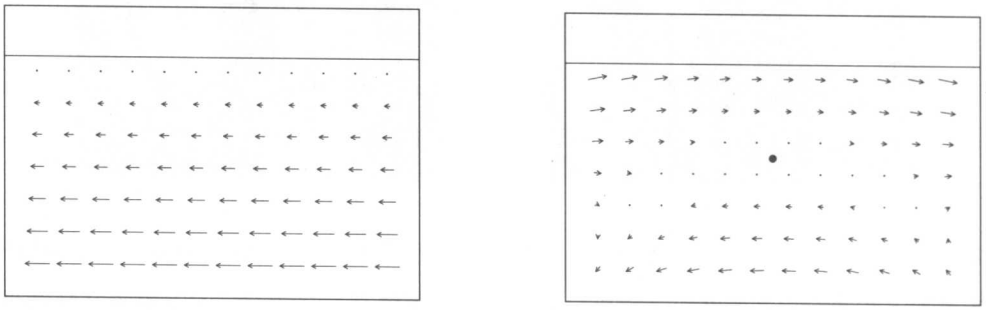
图21-21 在视线附近，表面点比凝视点更近，在相反的方向上比凝视点更远，都会产生视差

21.3.4 运动线索

眼睛和可视表面之间的相对运动将导致视网膜上那些表面图像的变化。眼睛和表面点之间的三维相对运动将在视网膜上产生表面点投影的二维运动。视网膜的这一运动被命名为光流（optic flow）。光流是多种深度线索的基础。此外，光流还可以用来确定人在现实世界中如何移动，以及是否即将发生碰撞（见21.4.3节）。

1. 从技术上讲，交叉视差和不交叉的视差是指产生视差的表面点比双眼视界（horopter）更近或者更远。双眼视界不是到眼睛的某一固定距离，而是一个通过凝视点的曲面。

如果在一直凝视某一表面点的同时移向另一侧，那么光流就可以提供类似于立体视差的深度
501 信息。这一现象被称为*运动视差*（motion parallex）。对于投影到视网膜上接近凝视点的表面点，光流为零，表示其深度等同于凝视点。与头部移动方向相反的流表示更近的点，等价于交叉视差；与头移动方向一致的点表示更远的点，等同于不交叉视差（见图21-22）。运动视差是强有力的相对深度线索。在原理上，运动视差可以提供绝对深度信息，如果视觉系统能够获取头部运动速度的话。实际上，运动视差看起来最多为一种弱绝对深度线索。



（a）　（b）

图21-22　（a）看着延伸的地平面往右移动产生的运动视差；（b）眼睛跟踪凝视点，做同样的运动

除了运动视差产生的以自我为中心的深度信息外，视觉运动还能提供相对于观察者移动的物体的三维形状信息。在感知文献中，这被称作*运动深度效应*（kinetic depth effect），在计算机视觉中则被称为*运动结构*（structure-from- motion）。运动深度效应假定物体运动的一个分量为*深度旋转*（rotation in depth），即在垂直于视线的方向上有一个旋转分量。

如图21-23所示，光流还提供表面边界的位置和形状信息。光流在空间上的不连续几乎总是对应着深度的不连续，或者是源于单独移动的物体。简单比较光流的大小不足以确定深度变化的符号，除非是在观察者穿过另外一个静态世界的特殊情况下。然而，即使出现单独移动的物体，在跨越表面边界时，往往可以通过其
502 他方式确定深度变化的符号。运动通常改变表面边界处较远的可见表面。表面纹理的出现（*增长*（accretion））或消失（*缺失*（deletion）），是因为近处正封闭的表面逐渐揭开或覆盖较远处已封闭的表面。边界两边表面纹理运动的比较还可用来推导顺序深度，即使没有纹理增长或缺失也是如此。光流的中断和纹理的增长/缺失，统称为*动态遮挡*（dynamic occlusion）线索。这是另一种强有力的有关环境空间结构的视觉信息源。

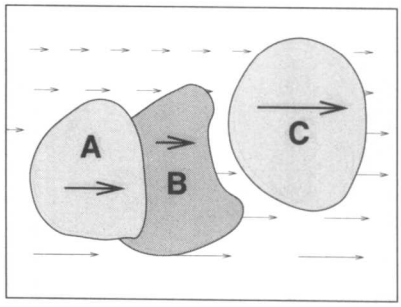


图21-23　表面边界光流信号的中断。在很多情况下，可以确定深度变化的符号（即顺序深度）

仅依据视觉运动不能确定观察者相对于点的移动速度（见21.4.3节）。尽管有这一限制，不能确定速度，但是仍有可能利用视觉信息来确定达到空间某可见点所需要花费的时间。当速度不变时，*接触时间*（time-to-contact）（一般称为*碰撞时间*（time-to-collision））是由观察者所接近的实体在视网膜上的图像大小、以及图像大小的增长速率决定的，其值等于图像大小除以增长速率。[1]在生物视觉文献中，这通常被称为τ函数（Lee & Reddish，1981）。如果可以获得估计碰撞时间所依据的结构距离信息，则可以确定速度。

1. 术语碰撞时间和接触时间有些误导，因为只有观察者的轨迹真正经过或者接近查看实体时才会发生接触。

21.3.5 绘画线索

图像可以包含很多产生该图像的现实空间的结构信息，即使是在没有双目立体视觉或运动的情况下也是如此。作为这一事实的证明，请注意即使闭上一只眼睛，保持头部不动，也不在环境中移动，现实世界看起来仍然是三维的。（对于照片和别的显示图像，情况更为复杂。在21.5节将讨论。）有三类这样的绘画深度线索（pictorial depth cue）。最著名的是直线透视（linear perspective）。即使没有透视，也还有大量的遮挡线索（occlusion cue）提供顺序深度信息。最后，与明暗处理有关的照明线索（illumination cue）、阴影和相互反射以及空间透视也提供有关空间布局的信息。



503

图21-24 经典的线性透视效果包括：物体大小按距离缩放、平行线会聚、地平面延伸为一条可见的地平线、地平面上物体的位置关系都是相对于地平线的（图像由Sam Pullara提供）

术语线性透视（linear perspective）常用于表示的图像性质包括：图中物体大小按照距离伸缩、平行线的会聚、地平面延伸到可见地平线，以及地平面上物体的距离与那些物体相对于地平线物体的图像位置之间的关系（见图21-24）。更正式地讲，线性透视线索是指那些利用了透视投影的视觉线索。在透视投影下，现实点投影到图像上的位置被伸缩了$\frac{1}{z}$，其中z为投影点到环境点的距离。这一关系的直接结果是远处的点被投影到图像中心附近（平行线会聚），现实世界中点的位置越远，在图像中对应点之间的间隔将越小（物体大小按照距离伸缩）[1]分析透视投影方程，令$z \to \infty$，可以解释现实世界中无限的平面终止与有限的地平线的事实。

除了21.4.2节描述的与大小有关的效果外，大多数绘画深度线索都涉及到直线透视投影图，该图依赖于所关注的物体与地平面的接触。实际上，这些线索估计的并不是到物体的距离，而是到地平面接触点的距离。假设观察者和物体都在水平地平面的上方，那么观察到的较低地平面的位置将更接近。图21-25从数量上阐明了这一效应。对于地面上方h处的视点，以及水平线与地面上关注点之间的偏差角θ，那么从观察者所在位置到关注点之间的距离就是$d = h\cot\theta$。偏差角提供了任意固定视点的相对深度信息，如果可以按照眼睛高度（h）伸缩，则还能够提供绝对深度信息。

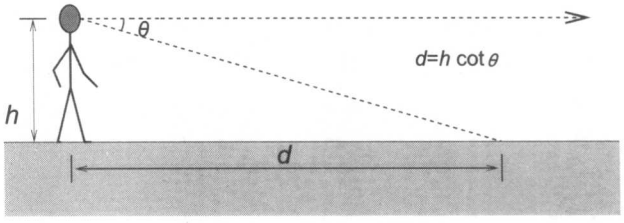


图21-25 到地面某位置的绝对距离，可以根据水平偏差角和眼睛高度来确定

虽然人的视觉系统几乎确实使用偏差角作为深度线索，但是准确地获取相关信息的机制仍然不清楚。角度θ的获取涉及到重力或者是可见地平线，有些迹象显示两者都在人的视觉中使用。眼睛高度h可能建立在姿态的基础上，在视觉上通过查看地上的脚来确定；或者是通过经验学习，假定为常量。虽然很多研究人员已经研究了这一问题，但是这些值是否确定以及如何确定的问题仍没有确切答案。

504

1. 真正分析生物视觉特性的数学方法并非如此，因为在计算机图形学和其他大多数成像应用中所采用的地平面投影公式不能很好地逼近眼睛的投影方式。

阴影提供了多种有关三维空间布局的信息。附着阴影（attached shadow）表明物体与另一个表面接触，该表面通常为地面。分离阴影（detached shadow）表明物体接近某表面，但并没有接触该表面。阴影可以作为间接深度线索使用，因为阴影能使物体出现在地面阴影位置所在深度上（Yonas，Goldsmith, & Hallstrom，1978）。使用这一线索时，视觉系统好像假设了光线直接来自上方（见图21-26）。

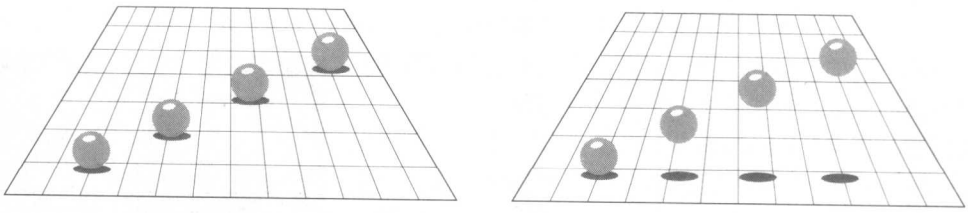


图21-26　阴影可以当作间接深度线索使用，因为阴影能够将物体的深度与地面的某个位置关联起来（源自Kersten, Mamassian and Knill（1997））

视觉提供有关表面方向以及距离的信息。按照坡度（tilt）和倾斜度（slant）来表示视觉确定的表面方向是比较方便的。坡度定义为表面法线投影图像的方向，倾斜度定义为表面法线与视线之间的夹角。

可见表面地平线可以用来查找某表面（实际上无限的表面）相对于观察者的方向。坡度的确定很直观，因为表面的坡度就是可见地平线的方向。倾斜度也可以揭示出来，因为从眼点到地平面的视线定义了一个平行于该表面的平面。在很多情况下，要么表面地平线不可见，要不就是表面小到其远边并不对应真正的地平线。在这样的情况下，可视纹理仍可用来估计方向。

在感知上下文中，术语纹理（texture）是指表面上由一些重复的子图案组成的可见图案。子图案及其分布可以像棋盘那样是固定的、规则的，也可以是在统计意义一致的，就像绿色田野那样[1]。当从倾斜的角度查看有纹理的表面时，纹理的投影视图相对于表面实际标记而言将扭曲失真。其中会出现两种非常独特的失真（Knill，1998），而且这两种失真都受倾斜度的影响。纹理
505 元素的位置和大小遵循前面介绍的直线透视图效果。这会带来纹理梯度（texture gradient）（Gibson，1950），因为元素大小和间距随距离下降（见图21-27a）。倾斜查看时各个纹理元素的图像以及元素的失真都是按照透视远近比例缩小的（见图21-27b）。这带来坡度方向的压缩。例如，斜视一个圆看起来像一个椭圆，次轴与主轴的比率等于倾斜度的余弦值。注意：透视缩短并不是直线透视的结果，虽然在实际中直线透视和透视缩短都提供关于倾斜度的信息[2]。

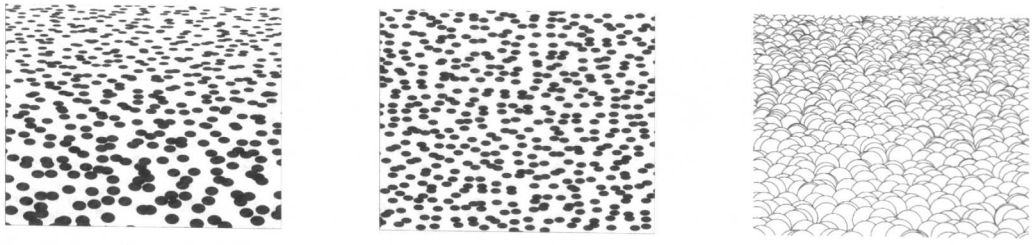


图21-27　倾斜度的纹理线索。（a）近处表现压缩和纹理梯度的表面；（b）远处只表现压缩的表面；（c）具有规则几何变化特性的近表面在外观上的可变性

1. 在计算机图形学中，术语纹理有不同的含义，指的是绘制过程中运用于表面的任意图像。
2. 如图21-27c所示，在斜着查看明显具有三维起伏特征的表面时，会出现第三种形式的视觉失真。当前还不知道人类视觉系统是否利用或者是如何利用这种效果来确定倾斜度。

对于当作表面倾斜度线索使用的纹理梯度，要求纹理元素的大小和间隔在整个有纹理的表面上都是不变的。如果空间大小和图像间隔没有完全遵循投影过程，那么通过颠倒投影效果来推导表面方向时可能会产生不正确的结果。同样，如果纹理元素的形状不是各向同性的，那么透视缩短也将失败。因为那样产生的形状不对称的纹理元素图像将不一定是与斜视关联的。这些是为了空间可视线索有效所需的假设的例子。这类假设对于它们通常反映世界的出现性质的程度而言是合理的。

明暗处理也提供有关表面形状的信息（见图21-28）。观察到的表面点的明度取决于表面反射系数，以及该表面相对于方向性光源和观察点的方向。当物体的相对位置、查看方向和照明方向都保持不变时，反射系数固定的表面的亮度变化能够显示出物体表面的方向变化。明暗成形（shape-from-shading）指的是根据观察到的亮度变化恢复表面形状的过程。尽管明暗能够结合其 506
他线索提供有效的表面形状迹象，但是仅从明暗变化几乎不可能恢复实际的表面方向。对于按照精确的几何比例变化的，明暗能够提供令人注目的三维外观，甚至绘制在二维表面上的图像也是如此（见图21-29）。



（a） （b）

图21-28 明暗成形。由于表面亮度变化率的差异，（a）和（b）中的图像看起来似乎具有不同的三维形状

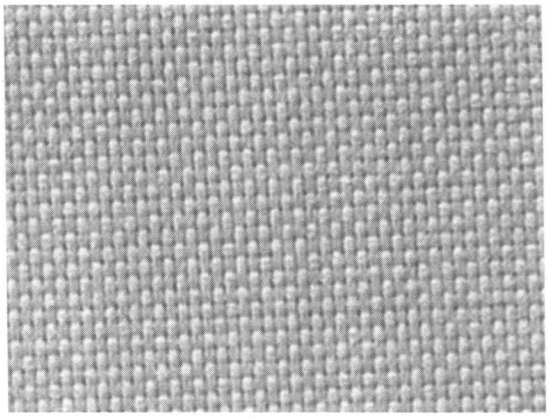


图21-29 明暗处理能够产生很强的三维形状感。在该图中，如果在几米远外单眼查看该图，那么这一效果将更强烈。如果在该图前面放一张纸板，在纸板上剪一个比该图片稍小的洞，那么三维形状感还能变得更强（见21.5节）。图片经Albert Yonas授权使用（另见彩图XIX）

有很多绘图线索能产生有关深度的次序信息，但并没有直接显示实际距离。在直线绘图中，不同类型的接合处约束了可能产生该图的三维几何形状（见图21-30）。在自然图像中也会出现很多这样的效果。感觉最有效的接合线索为T形接合（T-junction）。T形接合很明显地指出与T形主干相对的表面至少遮挡了一个较远的表面。T形接合通常产生一种非模态完成（amodal completion）的感觉，即看到某个表面延伸到另一个更近的遮挡表面下（见图21-31）。

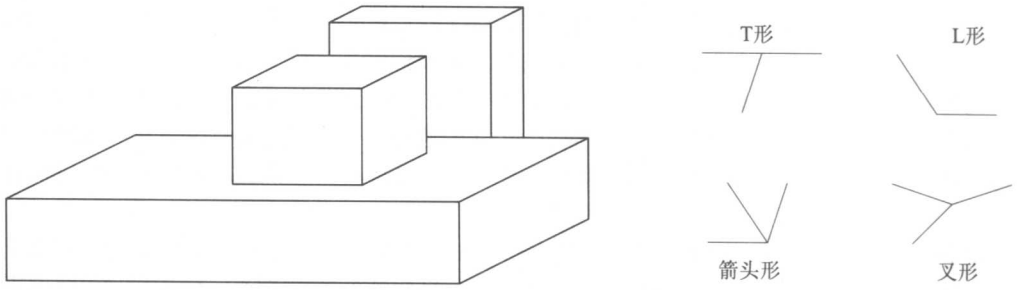


图21-30　(a) 接合处提供了有关遮挡和拐角凹凸的信息；(b) 普通二维表面物体的接合类型

507
~
508

大气效应也能引起视觉变化，提供有关深度的信息，特别是在很远的野外。达芬奇是第一个描述空间透视（aerial perspective）的人。空间透视也称为大气透视（atmospheric perspective），指的是散射使得远处景物的对比度降低，从而远处景物看起来比在近处的时候带有更多的蓝色（da Vinci，1970）（另见彩图XX）。空间透视是一种起支配作用的深度线索，尽管也有些推测可能会影响绝对距离的感知。虽然很多人认为物体距离越远看起来越模糊是因为大气效应，但是实际上大气散射引起的模糊很小。

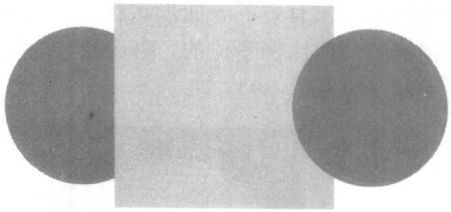


图21-31　T形接合使得左边的圆盘似乎延伸到矩形后面，而右边的圆盘则出现在矩形的前面，矩形在该圆盘后面继续延伸着

21.4　物体、位置和事件

虽然目前的视觉科学家普遍认为视觉的目的是提取有关物体（object）、位置（location）和事件（event）的信息，但是在提取什么样的关键信息、如何提取、如何利用所提取的信息执行任务等方面仍没有一致意见。对于物体识别的本质以及物体识别和其他感知特性之间的交互，存在较大的争论。我们所了解的有关位置的大部分内容都涉及低级空间视觉。低级空间视觉并不是与复杂物体的空间关系关联的问题，也不是在复杂环境中导航所需要的视觉处理。我们对于移动速度和方向的感知有一定的了解，但是对于实际事件感知的掌握则非常有限。视觉注意涉及到物体感知、位置感知和事件感知等方面。对于相对简单的受控刺激，有不少关于视觉注意力现象的数据，但是我们对于视觉注意力如何服务于高层感知目标的了解仍然很少。

21.4.1　物体识别

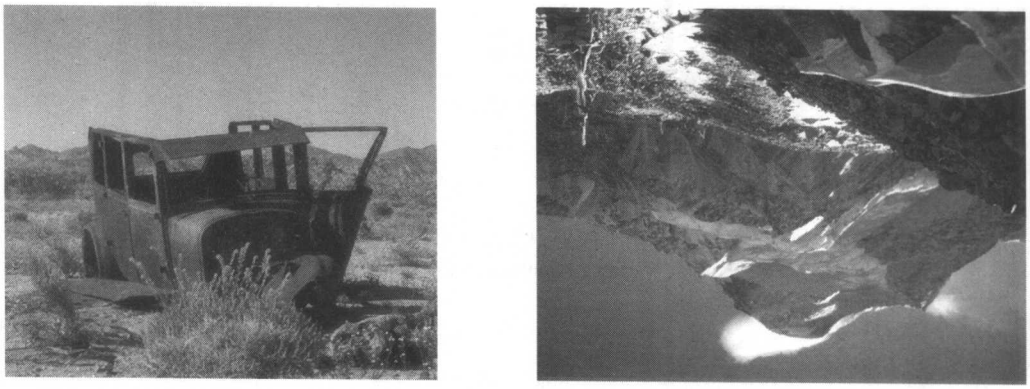
物体识别涉及图像的隔离，即将图像隔离为不同物理实体对应的组成部分，以及这些实体身份的确定。图21-32举例说明了这一过程的复杂性。即使从来没有看到这种车辆的图片，也从没有将车辆与这一前后关系关联起来过，我们仍毫不费力就可识别出左边的图像是某种车辆。右边的图像稍微难辨一些，除非将页面上下颠倒一下，这显示出人在物体识别中的方向偏好。

509

一般认为物体识别包含两个相当不同的步骤。第一步是将可见域组织成可能的组群（grouping）。组群对应着物体和表面。这样的分组处理是非常高效的（见图21-33），虽然对产生分组效应的低级图像特征没有清醒的意识[1]。分组是以如下要素为基础的：复杂的邻近交互作用，

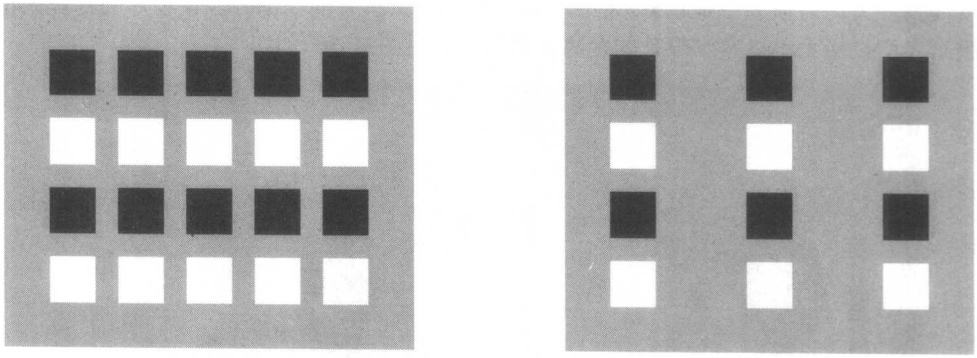
1. 最常见的视觉伪装形式将是添加纹理，欺骗感知分组过程，使得查看的结果在分组时不能将伪装的物体分离出来。

图像的亮度、颜色、形状以及主要结构方向的相似性，常见的运动，以及各种更复杂的关系。



(a) (b)

图21-32 物体识别的复杂性。（a）即使从前没有见过这一特殊的车辆视图，我们也能识别出那是一个类似车辆的物体；（b）快速看一下该图很难识别。如果将书本上下颠倒一下识别起来就容易多了



(a) (b)

图21-33 在感觉上，图像是按照复杂的相似性和组织准则识别为组群的。（a）亮度相似性导致4个水平的组群；（b）邻近关系产生三个垂直的组群

物体识别的第二个步骤是将组群解释为确定的物体。计算分析表明有很多明显不同的方法可 510
用来识别物体。感觉数据以及人的视觉实际采用了哪些数据都是不清楚的。物体识别要求视觉系统对每类物体的描述足以将其与别的类别区别开来。各种物体识别理论的区别主要在类别信息的描述，以及这些描述与实际物理视图的匹配机制。

可能存在三种一般的描述。*模板*（template）按照每类物体的原型视图来表示物体类别。图21-34展示了一个简单的例子。*结构化描述*（structural description）根据每个类别的区别特征以及这些特征之间的几何关系来表示物体类别。区别特征都是物体视图中比较容易检测的特征。结构化描述既可以表示成二维的，也可以表示成三维的。对于二维物体模型，要求每个明显不同的物体视图都有一个单独的描述。对于三维模型，可能有两种不同形式的匹配策略。第一种策略是，在使用有关空间线索进行分类之前，先确定被查看物体的三维结构，然后将视图的三维描述与已知物体的三维原型匹配。另一种可能的策略是允许使用某种机制来确定所查看物体的方向。该方向信息将用来旋转并投影潜在的三维描述，以便允许该描述与观察的物体进行二维匹配。最后一个描述物体类别性质的选项是*恒定特征*（invariant feature）。恒定特征按照更一般的几何性质描述物体类别，特别是那些对不同物体视图不敏感的特征。

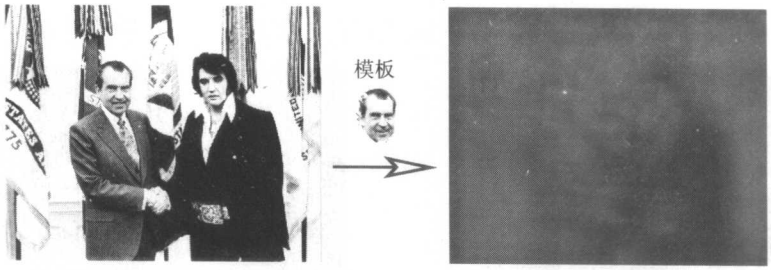


图21-34　模板匹配。右边图像中的亮点指出了与左边图像中的模板的最佳匹配位置。图像经National Archives and Records Administration授权使用

21.4.2　大小与距离

511

在缺乏明确深度信息的情况下，在视网膜上投影面积较大的物体，与那些视网膜投影面积较小的物体比较，看起来更近。这一效应被称为相对大小（relative size）。一个更强大的线索是习惯大小（familiar size）。习惯大小能够提供大小已知的可识别物体的绝对距离信息。在类似于图21-35的错觉中可以看到习惯大小作为深度线索的强大。在该图中，习惯大小与地平面基于透视的深度线索冲突。习惯大小是距离-大小（size-distance）关系的一部分，使物体的物理大小、在视网膜上光学投影大小以及到眼睛的距离联系起来（见图21-36）。



图21-35　左：透视和习惯大小线索都是一致的。右：透视和习惯大小线索不一致。图片由by Peter Shirley, Scott Kuhl和J.Dylan Lacewell.提供

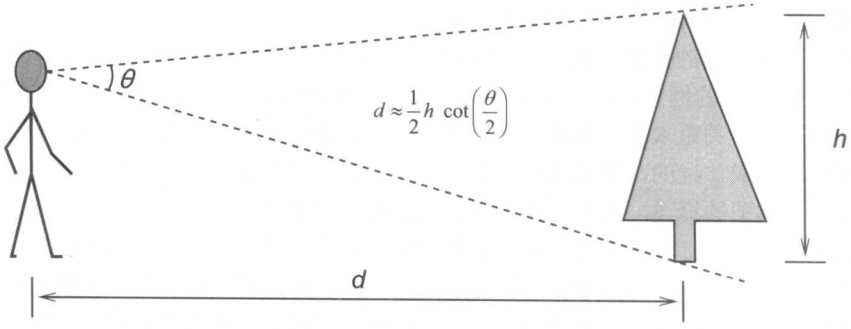
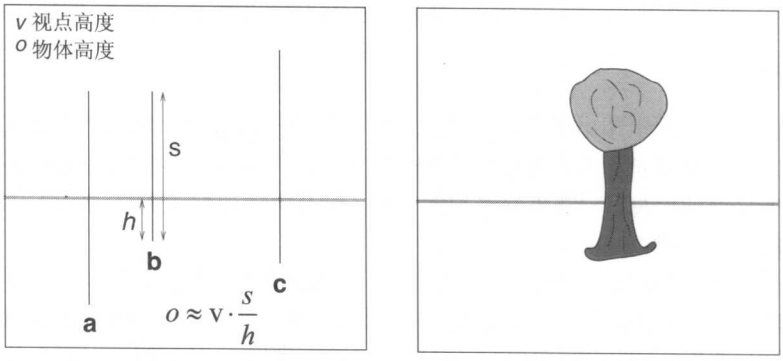


图21-36　距离-大小关系允许根据物体所对的视角来确定大小已知的物体的距离。同样，在已知距离上物体的大小也可以根据物体所对的视角来确定

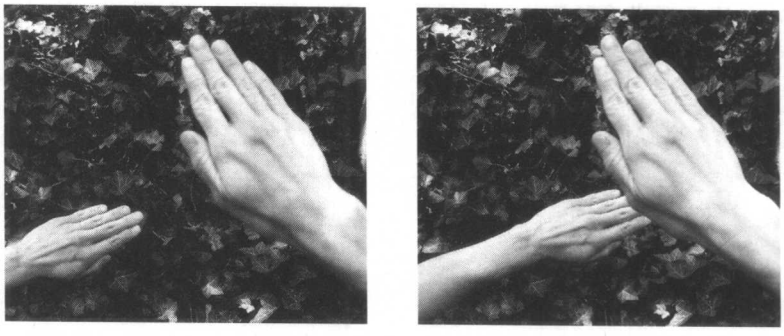
当物体立于平坦的地面上时，有一些额外的深度信息可以利用，特别是在地平线可见的情况下，或者是可以根据其他透视信息推导出地平线的情况下。相对于触地点的倾斜角度是一种相对深度线索，当按照眼睛高度伸缩时，能提供以自我为中心的绝对距离，就像前面的图21-25所示那样。地平线比率（horizon ratio）可以用来确定物体的实际大小，甚至是在未知物体距离的情况下（如图21-37所示）。地平线比例是指物体总的可见高度与该物体在地平线下可见部分的高度比。地平线比例隐含了这样一个事实：对于平坦地面，观测地平线的视线在某一位置上与物体相交，交点位置就是眼睛离地面的高度。 512 513



（a） （b）

图21-37 （a）地平线比率可以用来确定深度。地平线比率是指物体在地平线下的可见部分与整个垂直可见部分的比值；（b）实际例子

人的视觉系统有足够能力确定大多数所查看到的物体的绝对大小。大小的感知是受实际物理大小支配的。但是对于物体在视网膜上的大小则几乎没有任何清醒的意识。这类似于前面讨论过的亮度不变性，即亮度的感知受实际推理性质的支配，而不是由视网膜上光感受器实际感觉的低级特征决定的。Gregory（1997）描述了一个大小不变性（size constancy）的简单例子。像图21-38a所示的那样，向前伸出两只手，一只手臂伸直，另一只放在手臂中间的位置上。两只手看起来大小几乎一样，尽管它们的视网膜大小相差2倍。如果较近的那只手部分遮挡远处的那只手，那么这一效果将没有那么强，特别是在闭上一只眼睛的时候（见图21-38b）。视觉系统还能展示形状不变性（shape constancy），即感知的几何结构接近于物体的实际几何特性，而不是特定视网膜图像失真所预期的结果（见图21-39）。



（a） （b）

图21-38 （a）大小不变性使离眼睛距离不同的手的大小几乎与实际查看时的大小相同，尽管视网膜大小完全不同；（b）当一只手被另一只部分遮挡时，效果就没有那么强，特别是在闭上一只眼睛的时候（图片由Peter Shirley和Pat Moulis提供）



图21-39　形状不变性——桌子看起来是矩形，尽管它在图中的形状是一个不规则的四边形

21.4.3　事件

事件感知的很多方面都超出了本章的范围，因为他们涉及到复杂的非视觉认知过程。然而，有三种事件感知主要是视觉上的，而且与计算机图形学有直接联系。视觉系统能够提供一个人如何在现实世界中移动的信息，判断单独移动的物体，以及由于观察者的移动或者是移向观察者的物体可能产生的潜在碰撞。

视觉可以用来确定旋转以及相对环境的平移方向。最简单的情况是，朝着一个垂直于视线方向的平坦表面移动。假定有足够的表面纹理可以恢复光流，那么该流场将形成图21-40a所示的对称模式。流场膨胀中心（focus of expansion）在视域中的位置，将有一条对应于平移方向的关联视线。虽然光流可以在视觉上确定运动方向，但是并没有包含确定速度所需要的足够信息。为了了解这一点，请考虑如下情形：世界增大到两倍，观察者的移动速度也增加到两倍。结果距离加倍所引起的光流值大小的降低，将完全被速度加倍所引起的光流值大小的增加所抵消，导致流场保持不变。

514　图21-40b展示了观察者（更准确地说是观察者的眼睛）围绕垂直轴旋转的光流场。与平移运动不同的是，光流提供了足够的信息来确定旋转轴和旋转（角）速度。实际利用这一特性时的问题是，纯粹地绕着垂直于视线的轴旋转的运动所产生的流场，非常类似于与视线和旋转轴都垂直的纯粹平移运动所产生的流场。这就使得在视觉上很难区分两种完全不同类型的运动（见图21-40c）。图21-41展示了在一个更为现实的环境中移动所产生的光流模式。

如果观察者完全固定，那么移动物体的视觉探测就很容易，因为这样的物体将与视域中唯一的非零光流关联起来。当观察者也移动的时候，事情就变得相当复杂了。因为视域将受非零流的支配，其中的大部分或者是所有流都是由观察者与静态环境之间的相对运动所引起的（Thompson & Pong，1990）。在这种情况下，视觉系统必须对光流场中那些与整体光流不一致的模式特别敏感，而整体光流是由于观察者相对于静态环境的运动所产生的（见图21-42）。

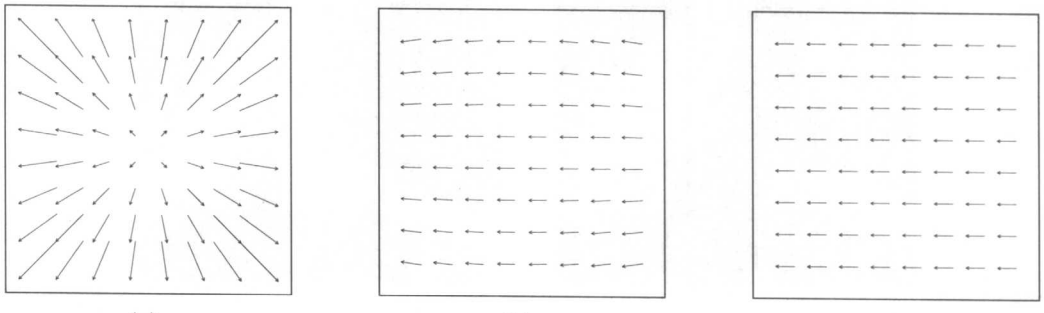


图21-40　（a）朝着一个平坦的带纹理的表面移动，产生一个膨胀的流场。膨胀中心指出了对应于运动方向的视线；（b）在查看垂直于视线的平坦表面时，围绕垂直轴旋转所产生的流场；（c）沿着带纹理的平坦表面平移所产生的流场

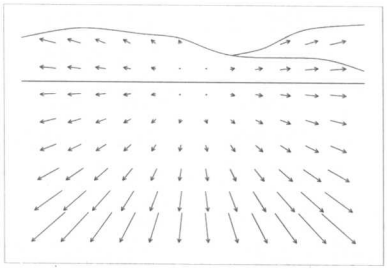


图21-41 穿过一个别样的静态环境所产生的光流，既提供了相对于环境的运动信息，也提供了环境中点的距离信息。在这一情况下，观察方向被压到地平线下，但是膨胀中心表明，该运动与地面平行

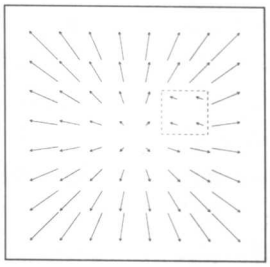


图21-42 从一个移动的观察点视觉探测移动的物体，要求识别出那些与穿过静态环境的运动无关的光流模式

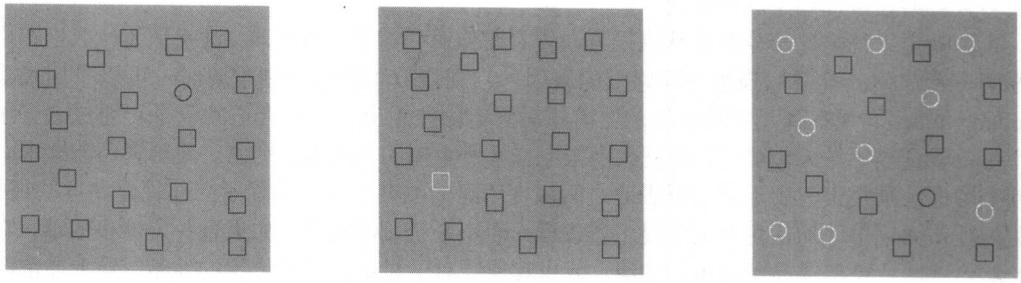
515

在21.3.4节介绍了在运动速度未知时，如何使用视觉确定与环境中某点的接触时间。假定观察者沿着一条直的、速度恒定的轨迹移动，而且没有独立移动的物体，那么不管什么样的表面，在对应于膨胀中心的视线方向上，经过τ关系所给出的时间，都能接触上。独立移动的物体使确定实际是否发生碰撞的过程复杂化。航海员采用了一种检测潜在碰撞的方法。在人的视觉系统中可能也运用了这种方法。那就是：对于非加速直线运动，在以自我为中心的参考坐标系中，将与视觉上正在膨胀的物体发生碰撞，而视觉上保持不变的则不会。

还有一种更为复杂的事件感知值得在此讨论，因为它在交互式计算机图形学中非常重要。人们对于人的运动特别敏感。运转力能够在只有步行者关节上的光可见的情况下识别出来（Johansson，1973）。这样的*移动光显示*（moving light display）足以识别出诸如步行者的性别以及可能的负重之类的性质。在计算机图形绘制中，观察者将注意到动画角色上很小的不精确的东西，特别是在他们企图模仿人的时候。

术语*视觉注意力*（visual attention）涵盖了从眼睛的指向，到复杂场景的认知效果，以及对所观测内容的解释等范围内的现象（Pashler，1998）。图21-43提供了一个注意过程如何影响视觉的例子，即使是非常简单的图像也是如此。在左边的两个面板中，在形状或颜色上不同于其他的模式立即“弹了出来”，毫不费力就注意到了。在右边的面板中，颜色和形状都不同的模式找起来就困难一些。这一现象的原因是，视觉系统能够并行搜索由个体性质区别的项目。当查找由两种同时出现的区别特征标识的项目时，则需要更有识别力的顺序搜索。基于图形的人机接口在设计的时候应该（但是通常没有！）掌握如何利用人的视觉注意过程，以便快速高效地传递重要信息。

516



（a） （b） （c）

图21-43 在（a）和（b）中视觉注意力很快就被吸引到不同形状或颜色的项目上。在（c）中为了找到一个形状和颜色都不同的项目，顺序搜索似乎是必要的

21.5 图片感知

到目前为止，本章已经介绍了直接通过人眼查看现实世界时的视觉感知。当我们查看计算机

图形结果时，自然，我们看到的是绘制的图像，而不是真实的世界。这有重要的感知暗示。在原理上，应该可以产生看来与真实世界没有区别的计算机图形，至少对于单眼查看没有物体运动或者是观察者运动的情况是如此。设想通过玻璃窗看世界。现在，严格按照原先在各个点上看到的颜色对玻璃窗上的每一点着色。[1]通过这一操作进入眼睛的光线并没有改变。也就是说，不管是查看着色的玻璃还是通过玻璃查看真实世界，感觉都应该是一样的。可以认为计算机图形学的目标是：在实际上并没有等价的真实视图可用的情况下，生成该着色的窗户。

计算机图形学和其他视觉艺术的问题是，在实践中不能通过对平坦表面着色来匹配真实世界的视图。采用任何现有的显示技术都不可能再现真实世界中光的亮度和动态范围。绘制图像的分辨率也经常低于人的视觉可感知的最小细节。图片中的亮度不变性和颜色不变性也没有真实世界那么显而易见。这可能是因为视觉试图根据观察环境周围的照明情况来补偿照明亮度和颜色的变
517 化，而不是根据绘制图像所关联的照明来进行补偿。这就是为什么照片中逼真的彩色外观依赖于拍摄时针对光源特性所进行的胶片色彩平衡的原因。这同时也解释了为什么逼真的彩色视频需要白平衡处理步骤。虽然对分辨率、亮度和动态范围方面的局限如何影响简单模式的检测能力有了很多了解，但是在这些显示性质如何影响空间视觉或物体识别方面，几乎什么也不知道。

我们对于该问题的其他方面已经有比较好的了解。心理学家将其称为绘画空间（pictorial space）感知（S. Rogers, 1995）。查看图像和查看真实世界的区别之一是：适应性调节、双目立体视差、运动视差或别的深度线索，可能显示所查看的表面非常不同于客观世界所要表现的距离。在这种情况下看到的深度可能是介于图中绘画线索显示的深度与该图像自身距离之间的某个结果。在查看照片或计算机显示的时候，这经常导致一种比预期尺度更小的感觉。另一方面，看同样的电影，与在电视上看相比，在大屏幕剧院看能产生一种更引人注目的宽阔感，即使调整电视机的距离使视角一样也是如此，因为电影屏幕在更远处。

使用透视投影绘制的计算机图形有一个视点，指定模型空间的一个位置和方向；还有一个平截头体，指定查看的水平域和垂直域，以及该观察变换其他个别方面的状况。如果不是从正确的位置查看绘制的图像，那么到图像边框的视角将不能匹配创建该图像时所采用的平截头体。在图像内的所有视角也将失真，结果是导致所有基于直线透视的绘画深度和方向线索失真。在实践中，当观察者离照片或显示表面太近或者是太远时，经常出现这种影响。如果观察者太近，透视深度线索就会被压缩，表面倾斜度线索将显示该表面比实际情况更接近于视线的垂直方向。如果观察者离照片或屏幕太远，这一情况则会反过来。如果视线不经过查看区域的中心，情况则更为复杂，而大多数观察通常都属于这种情况。

人的视觉系统能够部分补偿从错误方向查看图像时引起的透视失真。这就是为什么坐在电影
518 院不同位置却体验着类似描述空间感觉的原因。当视点位置控制特别重要时，可以采用观察管（viewing tube）。观察管是安装在显示器前相对固定的位置上、大小合适的管子。通过该管子观察者可以看到显示结果。观测管将观察点限制在（所希望的）正确范围内。观测管还能够很有效地减少图像绘画线索与实际显示器表面之间深度信息的冲突。它们消除了立体视差和运动视差。这两种视差如果出现的话将对应于显示器表面，而不是绘制的视图。如果直径足够小的话，它们还能够隐藏图片框架或显示设备的边缘，减少其他针对于显示器表面位置的线索。异乎寻常的视
519 觉沉浸显示设备（例如头戴显示器（HMD））功能更强，能够隐藏显示器表面位置的视觉线索，增加与绘制内容几何属性一致的双目立体视差和运动视差。

1. 这一思想最初是由画家Leon Battista Alberti在1435年描述的，也就是现在所知的Alberti窗。它与摄影启蒙密切相关。

第22章

色调再现

Erik Reinhard

在第21章的讨论中提到了，人的视觉系统能够适应大范围的观察条件。在正常观察情况下，我们能够区分的照明范围大约是4到5个对数单位。也就是说，能够看清的最亮区域与最暗区域的比值可以高达100 000:1。通过自适应处理，还能适合更大的照明范围。那些与人的视觉系统的能力匹配的图像，被称为是高动态范围的（high dynamic range）。

视觉刺激通常产生高动态范围的图像（Ward Larson & Shakespear, 1998）。图像捕捉技术的最新成果使我们可以将多次曝光结果对齐，并重新接合为单一的高动态范围图像（Debevec & Malik, 1997）。多次曝光技术也适用于视频。此外，我们期望未来的硬件能够直接产生高动态范围的场景照片或电影。通常，将像素点看作由三个浮点数组成的三元组。

由于创建高动态范围影像正变得越来越容易，因此对显示这些影像的需求也在快速增长。不幸的是，目前大多数显示设备、监视器和打印机，都只能显示大约2个对数单位的动态范围。这样的设备被认为是低动态范围的。如今大多数现有图像都是采用byte-per-pixel-per-color通道来表示的。这一表示符合当前的显示设备，但是并不符合所表示的场景。

通常，低动态范围的图像不能无损地表现场景。一个常见的例子就是在室内透过窗户看室外。人很容易看清室内和室外的细节。而传统的照相一般不能捕捉整个范围内的信息。摄影师不得不选择场景的室内部分或者是室外部分进行适当曝光（如图22-1所示）。如果采用高动态范围成像，并准备选用本章介绍的技术来显示这些图像，那么这样的取舍决定就可以避免（见图22-2）。

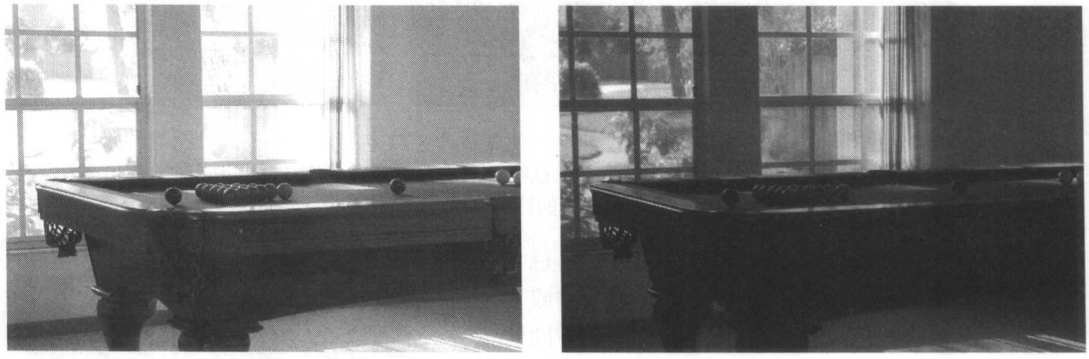


图22-1 对于传统的照片，场景的某些部分可能曝光不足或者是曝光过多。为了显现斯诺克台球桌，左边的图像中窗户外的景色被烧毁了。反过来，如果对场景的室外部分进行适当曝光，那么斯诺克台球桌看起来就太暗了。请将本图与图22-2进行对比，图22-2展示了一幅高动态范围的图像，该图像是采用色调再现算法为显示目的而准备的

有两种策略可以显示高动态范围的图像。第一种是开发能够直接容纳高动态范围的显示设备（Seetzen, Whitehead, & Ward, 2003; Seetsen et al., 2004）。第二种是准备高动态范围的图像，在低动态范围的显示设备上显示（Upstill，1985）。这是当前较为普遍的方法，也是本章所要介绍的主题。虽然可以预见在不久的将来将广泛使用高动态范围的显示设备，压缩高动态范围图像的需求可能减少，但是不可能消失。实际上，类似于本书的打印媒体本质上是低动态范围的。

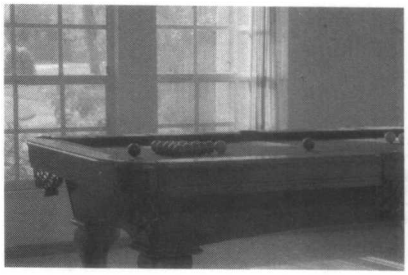


图22-2　采用最近色调再现算子，进行色调映射后的高动态范围图像（Reinhard & Devlin，2005）。在该图中，室内部分和窗外部分都适当曝光了

为了在低动态范围显示设备上的显示而压缩图像的动态范围的处理，被称为**色调映射**（tonemapping）或者是**色调再现**（tone reproduction）。一个简单的压缩函数就是对图像进行归一化（见图22-3中的左图）。归一化是由线性伸缩构成的。线性伸缩只有当图像的动态范围在边上高于显示设备的动态范围时，才是充分的。对于动态范围较高的图像，小差别的亮度值将被量化为同样的显示值，结果丢失了可见的细节。在图22-3的中间图像中，所有像素值超过用户指定的最大值的像素，都被设置为用户指定的最大值（即，他们是钳位的）。这使得归一化对外来噪声的依赖比较小，但是在图像的明亮区域有信息丢失。作为比较，图22-3右图是经过色调映射后的图像，展现了黑暗区域和明亮区域的细节。

522



图22-3　对高动态范围图像进行线性伸缩，使其适合于给定的显示设备，可能导致明显的细节丢失（左图和中图）。左图是经过线性伸缩后的图像。在中间的图像中高值被钳制了。作为比较，右边是色调映射后的图像，亮区和暗区的细节都是可见的

一般而言，线性伸缩不适合于色调再现。色调再现的关键问题在于：压缩图像的同时保留一种或多种图像性质。不同的色调再现算法一般专注于不同的属性，例如对比度、可见细节、亮度或外观。

理想情况的是，在低动态范围的显示设备上显示色调映射图像产生的视觉效果，与观察到的原始场景一样。在特定显示设备的约束下，这一般是达不到的，尽管能够朝着尽可能逼近的目标努力。

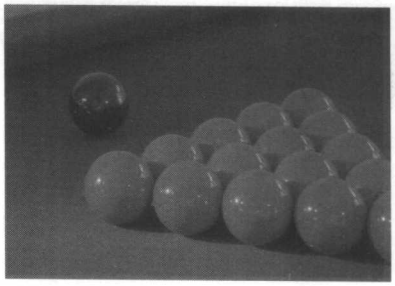


图22-4　用于图22-5中色调再现的示例图像

作为一个例子，我们创建了图22-4所示的高动态范

围图像。对该图像进行色调映射，并在显示设备上显示出来。然后将该显示设备本身放到场景中，使其显示自身的背景，如图22-5所示。在理想情况下，该显示应该是透明的。这一目标的实现取 523
决于色调再现算子的质量，以及所绘场景的特性。

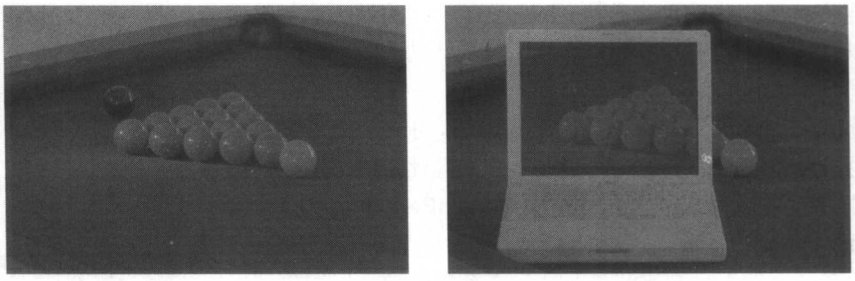


图22-5 图22-4中的图像经过色调映射后，显示在一个监视器上。然后将该监视器放在场景中拍摄该图像的位置上。结果应该是监视器看起来好像透明的，但是具体情况取决于色调再现算子的质量

22.1 分类

虽然可以按照所服务的属性或者是所面临的任务来分类色调再现算子，但是我们还是按照一般的技术来分类这些算法。这将使我们能够展现大量不同算子之间的差异和相似之处。从而有希望针对特定的色调再现任务，进行有意义的具体算子的选择。

主要的分类方案取决于色调再现算子的实现。而实现往往建立对在不同学科的洞察基础上。特别是，那几个建立在人类视觉感知基础上的算子。

人的视觉系统使用视网膜上的光感受器检测光。光被转换为电信号，经视网膜部分处理后，传输给大脑。除了视网膜中最初几层细胞外，检测光线产生的信号都是以脉冲序列传输的。携带的信息量就是这些脉冲的发生频率。

人的视觉系统能够检测的光的范围，远远超过大脑传输信息的频率范围。因此，人的视觉系统毫不费力就解决了色调再现问题——将大范围的亮度变换为小频率范围内的脉冲序列。因此，效仿人类视觉系统的某些方面就是一种有价值的色调再现方法。该方法的详细情况将在22.7节中解释。 524

第二类算子建立在物理学的基础上。光在被感受器吸收之前与表面和体进行交互作用。在计算机图形学中，通常采用绘制方程建模光的交互作用。对于纯散射表面，绘制方程可以简化为表面入射光线（照度）与表面反射能力（反射率）的乘积（Oppenheim, Schafer, & Stockham, 1968）。

因为反射系数是表面的被动性质，按照定义，散射表面的反射率是低动态范围的——取值一般在0.005到1之间（Stockham, 1972）。表面反射率不会超过1，因为反射的光线不可能比入射的光线还多。另一方面，照度可以取任意大的值，而且只受光源的强度和接近度的限制。

因此，图像的动态范围主要由照度支配。在实际的散射场景中，可行的色调再现方法是，将反射率与照度分离，压缩照度部分，然后重新组合图像。

然而，假设场景中所有表面都是散射表面，一般是不对的。很多高动态范围图像都描绘加亮区或者是直接可见的光源（见图22-3）。镜面反射的亮度可能与光源几乎一样高。

当前所采用的各种色调再现算子将图像分割为高动态范围的基础层和低动态范围的细节层。如果描绘的场景是完全散射，这些层就可能表示照度和反射率。对于包含直接可见光源或者是镜面高光区的场景，虽然不能对各层附加直接的意义，但是基础层与细节层的分离仍然允许设计高效的色调再现算子。这样的算子将在22.5节中讨论。

22.2　动态范围

传统图像在存储时为每个像素的红、绿、蓝分量都分配一个字节。这样的编码方案所提供的动态范围，取决于可表示的最小值与最大值的比值，以及相继值之间的步长。这样，对于低动态范围的图像，每个颜色通道只有256种不同取值。

高动态范围图像编码的取值范围明显要大，可表示的最大值可能
525 大很多，而相继值之间的步长则可能小很多。因此，高动态范围图像的文件大小通常也更大，尽管在标准中（OpenEXR高动态范围文件格式（Kainz，Bogart, & Hess，2003））至少包含一个强大的压缩方案。

另一种限制文件大小的方法是对高动态范围的数据运用色调再现算子。然后将该结果编码成JPEG格式。此外，可以按照像素方式将输入图像除以色调映射图像。然后对除的结果进行亚采样，并像JPEG图像那样在文件头中花少量的数据存储该结果（G．Ward, & Simmons, 2004）。这种子带编码图像的文件大小与传统的JPEG编码处于同样的量级上。显示程序可以直接显示JPEG图像，或者是将色调映射图像与文件头存储的数据相乘，重构高动态范围的图像。

一般，最小步长和可表示的最大最小比值共同决定了图像编码方案所能提供的动态范围。尽管有更高效的编码方案，但是对于计算机生成的影像，在存入文件或者实现到屏幕上之前都是以三元浮点数值保存的（Reinhard，Ward，Debevec, & Pattanaik，2005）。由于大多数显示设备装备的仍然是8位数模转换器，因此可以认为色调再现就是将浮点数映射成字节，使结果能够在低动态范围显示设备上显示。

单个图像的动态范围通常比较小，而且主要是由场景中的最大最小亮度决定的。因此测量图像动态范围的简单方法就是计算图像中最大像素值与最小像素值的比值。忽略少量最暗和最亮的像素，可以降低对局部异常的敏感性。

做为一种选择，还可以将该比值表示成对数域的差值。这一测度对局部异常的敏感性更低。在图22-6～图22-10中展现了不同动态范围的图像示例。注意在这种情况下夜景的动态范围不比白天小。虽然夜景的所有值都偏小，但是最大最小值的比值没有变。

然而，记录设备或者是绘制算法会引入噪音，从而降低可用的动态范围。这样，图像动态范围的测量就应该加入噪音音素。因此更好的动态范围测度可能是信噪比。跟信号处理一样，信噪比是用分贝表
526 示的。



图22-6　动态范围2.65个对数（$\log_2$）单位

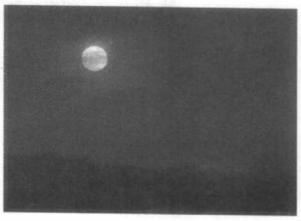


图22-7　动态范围3.96个对数（$\log_2$）单位



图22-8　动态范围4.22个对数（$\log_2$）单位



图22-9　动态范围5.01个对数（$\log_2$）单位

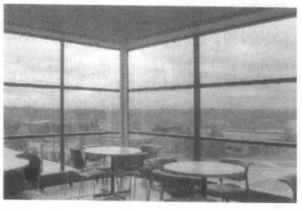


图22-10　动态范围6.56个对数（$\log_2$）单位

22.3　颜色

色调再现算子通常压缩亮度值，而不是对彩色图像的红绿蓝分量直接操作。在将亮度值压缩成显示值$L_d(x,y)$后，只要保持压缩后颜色通道之间的比例与压缩前一致，就可以重构彩色图像，即（其中s=1）（Schlick，1994b）：

$$I_{r,d}(x,y)=\left(\frac{I_r(x,y)}{L_v(x,y)}\right)^s L_d(x,y)$$

$$I_{g,d}(x,y)=\left(\frac{I_g(x,y)}{L_v(x,y)}\right)^s L_d(x,y)$$

$$I_{b,d}(x,y)=\left(\frac{I_b(x,y)}{L_v(x,y)}\right)^s L_d(x,y)$$

该结果常常出现过饱和的情况，因为就整个亮度级别而言，人的颜色感知是非线性的。这意味着在暗淡的环境下查看显示器上明亮的室外场景图像时，眼睛适应的是暗淡的环境而不是室外的光照。保持颜色比例不变，并没有考虑到这一效果。

作为替换，可令饱和常量 s 稍微比1小一些。如图22-11以及彩图X所示，每个通道的伽马校正能将饱和度降低到一个适当的水平（Fattal，Lischinski, & Werman，2002）。更为全面的方案是将颜色外观建模的思想结合到色调再现算子中（Pattanaik, Ferwerda, Fairchild, & Greenberg, 1998; Fairchild & Johnson, 2004; Reinhard & Devlin, 2005）。

527

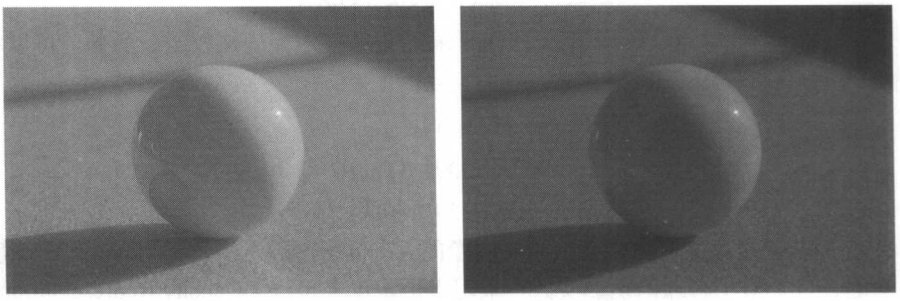


图22-11 每个通道的伽马校正能够降低图像的饱和度。左图是令s=0.5减小饱和度的结果，右图是没有减小饱和度的结果（s=1）（另见彩图X）

最后，如果示例图像已经有了颜色表示方案，那么可以将该颜色方案作用于新的图像。这种图像之间的颜色映射可以用于细微的颜色校正，例如饱和度调整或者是更富有创造性的颜色映射。映射过程是将源图像和目标图像都转换到一个去相关的颜色空间。在这样的颜色空间中，每个颜色通道的像素值都单独处理，不会引入太大的失真（Reinhard et al., 2001）。

在去相关的颜色空间中，图像之间的颜色映射是比较直观的：先计算源图像和目标图像各个颜色通道中所有像素的平均值和标准差。然后，伸缩平移目标图像，使目标图像每个颜色通道的平均值和标准差都与源图像一样。最后将去相关的颜色空间转换到RGB空间，并将取负值的像素钳制为零，就可以得到结果图像。使用该算法，图像的动态范围可能发生变化。因此，建议对高动态范围的图像应用该算法，并在其后应用传统的色调再现算法。比较合适的去相关的颜色空间是20.8节介绍的对立空间。



图22-12 用来演示颜色变换技术的图像。结果显示在图22-13和图22-31中（另见彩图XI、彩图XII和彩图XIV）

528

对图22-12中的图像应用这种颜色变换的结果显示在图22-13中。

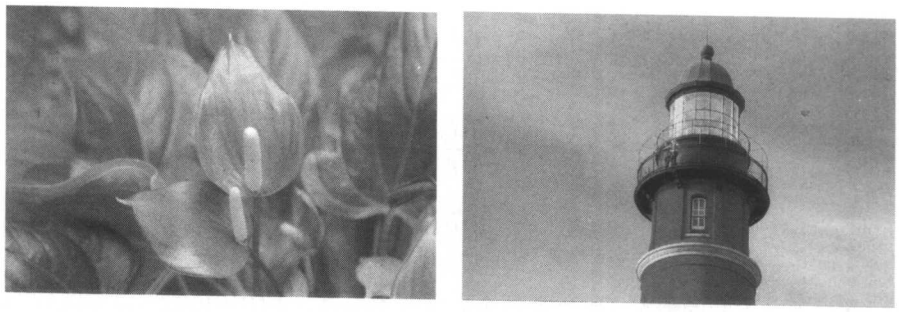


图22-13　左边的图像用来调整图22-12所示图像的颜色，结果显示在右边（另见彩图XII）

22.4　图像形成

眼下我们假定图像的形成是表面漫反射光线的结果。本章后面将把该约束放松到直接描绘光源和高光的场景中。这样每个像素的亮度 L_v 就可由下面的乘积来逼近：

$$L_v(x,y) = r(x,y)E_v(x,y)$$

其中，r 表示表面反射率，E_v 表示照度。下标 v 表明正在使用光度加权量。另外，可以将该表达式写成对数域的形式（Oppenheim et al.，1968）：

$$\begin{aligned} D(x,y) &= \log(L_v(x,y)) \\ &= \log(r(x,y)E_v(x,y)) \\ &= \log(r(x,y)) + \log(E_v(x,y)) \end{aligned}$$

摄影幻灯片是通过改变材质密度来记录图像的。在传统摄影术中，密度的变化与亮度之间是对数关系。这样，类似于普通的摄影实践，采用术语*密度表示法*（Density Representation，D）来称呼对数亮度。当采用对数表示时，反射率和照度就变成相加的关系。这有利于两个分量的分离，尽管反射率或照度的分离是一个欠约束问题。实际中，分离只能达到一定的程度，而且还依赖于图像的构成。不管怎么样，色调再现将建立在这两个图像形成分量的松解基础上，就像随后两节所展示的那样。

22.5　基于频率的算子

对于常见的散射场景，由于纹理表面以及表面边缘的出现，使得反射率显示出空间高频特性。另一方面，照度往往是一个随空间缓慢变化的函数。

因为反射率是低动态范围的，而照度是高动态范围的，所以可以试着将这两个分量分离。反射率和照度的频率依赖为此提供了一种解决方案。例如，计算图像的傅里叶变换并消弱低频部分，
529 就可以压缩照度分量，而保持反射率不受影响——据我们所知最早的数字色调再现算子采用的就是这一方法（Oppenheim et al., 1968）。

最近，一些别的算子也遵循了这一推理路线。特别是双向滤波器和三向滤波器，可以将图像分离成基础层和细节层（Durand & Dorsey, 2002; Choudhury & Tumblin, 2003）。这两种滤波器都是保留边缘的平滑算子，可以多种不同方式使用。将保留边缘的平滑算子作用于密度图像，能够产生模糊化的图像，但是锐利的边缘仍然被保留下来（见图22-14左图）。可以将这样的图像看作基础层。然后按像素方式以该基础层除以高动态范围图像，就可以获得细节层。细节层包含所有高频细节信息（见图22-14右图）。

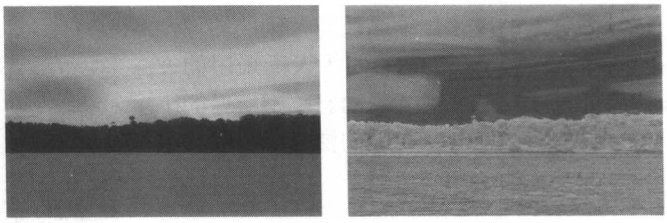


图22-14 双向滤波删除了不重要的细节，但是保留了锐利的梯度（左图）。与其关联的细节层显示在右图中

对于散射场景，基础层和细节层类似于照度与反射率的表达形式。对于描绘加亮区和光源的场景，这种平行对应关系就不成立了。但是，不管图像内容如何，都可以将图像分离成基础层和细节层。在重组压缩密度图像之前，压缩基础层可以创建低动态范围的密度图像（见图22-15）。经求幂运算后，即可得到可显示的图像。



图22-15 使用双向滤波进行色调映射后的图像。在压缩基础层后将图22-14中的基础层和细节层重新组合

保留边缘的平滑算子还可以用来计算每个像素的局部自适应级别。该自适应级别可以用于空间可变的色调再现算子或者是局部色调再现算子。在22.7节将介绍双向滤波器和三向滤波器。

530

22.6 梯度域算子

上一节基于频率算子的讨论在梯度域也成立。假设没有直接可见的光源，那么反射率分量在梯度域将是一个带有尖锐脉冲的常量函数。类似地，照度分量在各个地方都是小梯度。

人通常能够分离常见场景中的照度和反射率。除去照度之后，对表面反射率的感知称为光亮度（lightness）。为了评估只包含散射表面的图像的光亮度，B. K. P. Horn首次提出在梯度域分离反射率和照度（Horn, 1974）。他采用简单的阈值方法，删除所有小梯度，然后利用完全多栅格法（Full Multigrid Method）（Press, Teukolsky, Vetterling, & Flannery, 1992）求解一个Poisson方程，将图像合成起来。

其结果类似于保留边缘的平滑滤波器，这也是预料中的结果，因为Oppenheim基于频率的算子建立在同样的场景反射率和图像形成假设基础之上。特别的是，Horn工作的直接目标是“Mondrian的迷你世界”。那是散射场景的简化版，类似于著名荷兰画家Piet Mondrian的抽象油画。

Horn的工作不能直接作为色调再现算子使用，因为大部分高动态范围的图像都包含光源。但是，稍做改变就可以将这一工作转化为一种合适的色调再现算子。如果图像中描绘了光源或者是镜面表面，那么大的梯度将与光源和高亮区的边缘关联在一起。这些因素将导致图像拥有高动态范围。图22-16展现了一个这样的例子，其中斯诺克球的高亮区产生了锐利的梯度。

531

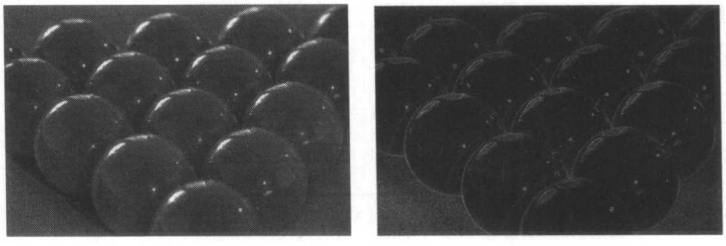


图22-16 左图（采用了梯度域压缩算法进行色调映射）显示了一个带有高亮区的场景，这些高亮区域显现了右图中的大梯度。右图中梯度的大小被映射为灰度级（黑色的梯度为0，白色的梯度最大）

这样，我们可以消减大的梯度来压缩高动态范围的图像，而不是对梯度场进行阈值限制。这就是Fattal等人所采用的方法。他们通过压缩梯度场，成功地展现了高动态范围影像的压缩（见图22-17）（Fattel et al., 2002）。Fattal的梯度域压缩并不局限于散射场景。



图22-17　采用梯度域压缩进行色调映射后的图像

22.7　空间算子

在后续几节，将讨论不经过变换而直接对像素进行压缩的色调再现算子。这些算子通常分为全局算子和局部算子。在全局色调再现算子中所有像素都按照同一个压缩函数改变像素的亮度值。全局的意思来源于，很多这样的函数都需要分析整个图像，确定一些锚点值。在实践中大部分算子都采用几何平均值$\bar{L}_v$来控制压缩，即：

$$\bar{L}_v = \exp\left(\frac{1}{N}\sum_{x,y}\log(\delta + L_v(x,y))\right) \tag{22-1}$$

在方程22.1中，小的常量δ主要用来防止出现黑色像素的零均值。该几何均值通常被映射到预先定义好的显示值上。图22-18展现了几何平均值映射到不同显示值上的效果。另外，有时也使用图像的最小最大亮度。设计全局算子面临的主要挑战在于压缩函数的选择。



图22-18　在将几何平均值映射到不同显示值后所采用的空间色调映射算子（左：0.12，右0.38）

另一方面，局部算子按照具体的压缩函数压缩每个像素，其中压缩函数是由所选择的邻近像素来调节的，而不是整个图像。其合理性在于明亮区域的亮像素在感觉上可能不同于暗淡区域的
532 亮像素。开发局部算子的设计关键在于压缩函数的选择、具备邻近区域大小的选择，以及使用局部像素值的方式。一般，局部算子能够取得比全局算子更好的压缩性能（见图22-19），当然计算代价也更高。

全局算子和局部算子的灵感往往都来自人的视觉系统。大部分算子都采用两个不同压缩函数
533 中的一个。该函数正交于局部算子和全局算子之间的差别。显示值$L_d(x,y)$通常都是按照下面两种函数形式，从图像亮度$L_v(x,y)$中导出来的，即

$$L_d(x,y) = \frac{L_v(x,y)}{f(x,y)} \tag{22-2}$$

$$L_d(x,y) = \frac{L_v(x,y)}{L_v(x,y) + f^n(x,y)} \tag{22-3}$$

在这些方程中，$f(x,y)$既可以是常量，也可以是随像素变化的函数。前一种情况得到的就是全局算子，而随空间变化的函数$f(x,y)$将导致局部算子。指数n通常为常量，其取值因特定算子而异。

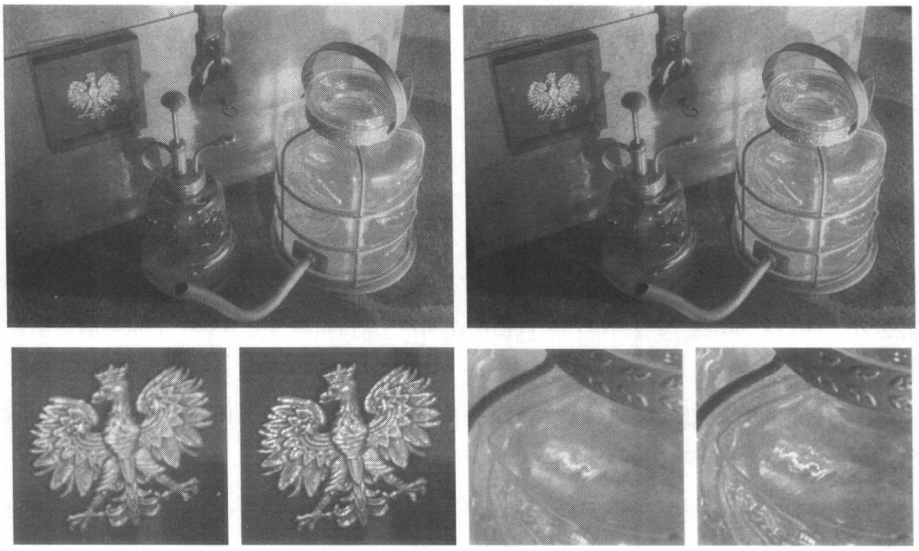


图22-19 每幅图像的全局色调再现算子（左）和局部色调再现算子（右）（Reinhard, Stark, Shirley, & Ferwerda, 2002）。局部算子显示了更多的细节，例如右边的金属徽章展现了更好的对比度，高亮区的图像轮廓也更鲜明

式（22-2）将每个像素的亮度值除以一个推导值，该推导值可以来源于整个图像，也可以源自局部邻近区域。式（22-3）在线性对数（log-linear）图上有一条S型的曲线，因此称为S型曲线。该函数形式拟合了不同物种的光感受器对光闪烁的电子响应测量数据。下面几节将讨论这两种函数形式。

22.8 除法

可以将每个像素都除以一个常量，将高动态范围图像调整到可显示的范围内。这种相除的方法本质上是线性缩放，就像图22-3所显示的那样。虽然图22-3展现的是一种特殊的线性缩放，但是该方法可以进一步精炼，即运用心理物理数据推导式（22-2）中的缩放常量 $f(x,y)=k$（G. J. Ward, 1994; Ferwerda, Pattanaik, Shirley, & Greenberg, 1996）。

作为替换，有个别方法计算随空间变化的除数。在这些情况中，$f(x,y)$ 为图像的模糊版本，也就是 $f(x,y)=L_v^{\text{blur}}(x,y)$。模糊化是通过图像与高斯滤波器的卷积来实现的（Chiu et al., 1993; Rahman, Jobson, & Woodell, 1996）。此外，模糊化图像时 $f(x,y)$ 的计算可能与白点移动接合起来，以方便颜色外观建模（Fairchild & Johnson, 2002; Johnson & Fairchild, 2003; Fairchild & Johnson, 2004）。

高斯滤波器的大小和权重对最终可显示的图像有深刻影响。高斯滤波器具有局部加权平均的选择效果。这样，色调再现实际上就是将每个像素除以与之关联的局部加权平均。如果滤波器内核的大小选得太小，将出现光晕失真（见图22-20左图）。光晕是局部算子的普遍现象，并且在使用除法进行色调映射时特别明显。

534

一般采用大的滤波器核可以将光晕失真最小化（见图22-20右图）。令滤波器的大小至少等于图像的四分之一，可以获得适当的结果。有时甚至期望更大的滤波器核来最小化失真。注意，在极限情况是滤波器与图像本身一样大。在那种情况下局部算子就变成全局的了，局部方法所提供的额外压缩也就不存在了。



图22-20　除以高斯模糊版本而进行的色调再现图像。左边图像滤波器核的大小为64个像素，右边图像为512个像素。对于基于除法的算法，选择大的滤波器核可以最小化光晕失真

这种函数形式，即每个像素都除以空间同一位置上高斯模糊化的像素，要求在压缩程度与失真大小之间进行令人讨厌的均衡。

22.9　S形曲线

式（22-3）遵循了一种不同于除法的函数形式，从而在压缩量、失真程度以及计算速度之间提供了不同的均衡。

S形曲线有几个合乎需要的性质。对于非常小的亮度值，映射几乎是线性的，因此保留了黑暗区域的对比度。该函数有一条等于1的渐近线，这意味着输出映射被绑定在0到1之间。

式（22-3）中的$f(x,y)$既可以作为全局常量计算，也可以是一个随空间变化的函数。遵照电生理学中的公共习惯，将$f(x,y)$称为半饱和（semi-saturation）常量，其值将以最优的方式决定输入图像中哪些值在色调映射之后是可见的。特别是，如果假设指数n等于1，那么等于半饱和常量的亮度值将被映射到0.5。选择不同半饱和常量的效果如图22-21所示。

535

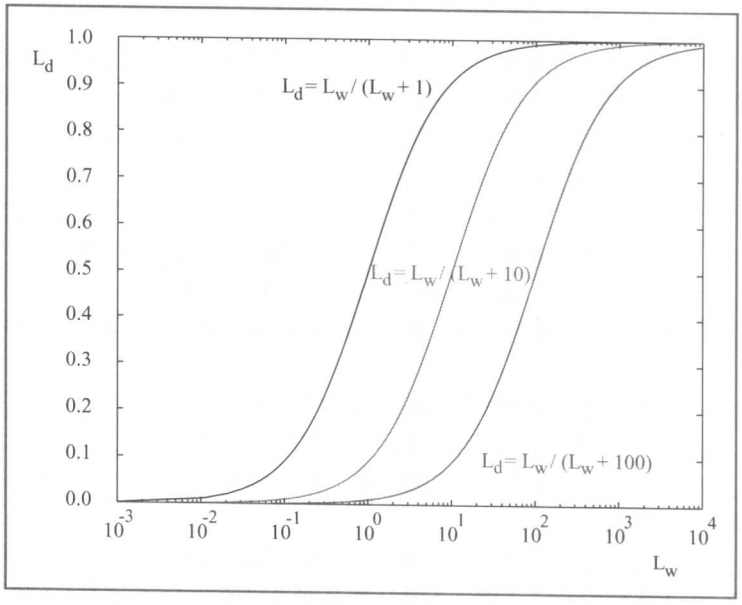


图22-21　半饱和常量的选择决定着输入值到显示值的映射

有几种不同的方式可用来计算函数$f(x,y)$（Reinhard et al., 2005）。最简单的形式是令

$f(x,y)=\overline{L}_v/k$，从而几何平均被映射到用户参数k（见图22-22）（Reinhard et al., 2002）。在这种情况下，比较好的初始值是k=0.18，尽管对于特别亮或特别暗的场景，该值可能升高或者是降低。该值可以从图像本身估计得来（Reinhard，2003）。式（22-3）中的指数n可以设为1。

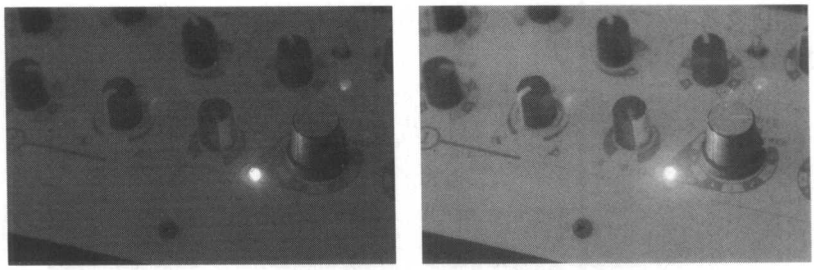


图22-22 线性伸缩图像（左）与用S形曲线压缩的色调映射图像（右）

在这种方法中，半饱和常量是几何平均值的函数，因而操作算子是全局的。这一全局算子的 536
变体之一就是通过几何平均值与每个像素亮度值之间的线性插值来计算半饱和常量：

$$f(x,y)=aL_v(x,y)+(1-a)\overline{L}_v$$

这一插值由用户参数a控制，该参数能够改变可显示图像的对比度（见图22-23）（Reinhard & Devlin，2005）。对比度大意味着明亮区域和黑暗区域的可见细节少了，反之亦然。这一插值结果可以看作是处于全局算子和局部算子之间半道上的房子，其中对两个极端情况进行插值，但是不需要昂贵的模糊操作。



图22-23 线性插值使色调映射图像的对比度多样化。左边图像的a参数为0.0，右边图像的为1.0

虽然一般算子都压缩亮度值，但是这一特定的算子可以扩充到包括一种简单的色调自适应形式，从而有机会调整通常与色调映射关联的饱和程度，就像本章开头所讨论的那样。

除了压缩亮度通道，S形曲线压缩还可以作用于每一个颜色通道：

$$I_{r,d}(x,y)=\frac{I_r(x,y)}{I_r(x,y)+f^n(x,y)}$$

$$I_{g,d}(x,y)=\frac{I_g(x,y)}{I_g(x,y)+f^n(x,y)}$$

$$I_{b,d}(x,y)=\frac{I_b(x,y)}{I_b(x,y)+f^n(x,y)}$$

$f(x,y)$的计算也被修改为双线性插值，在几何平均亮度与像素亮度以及每个独立颜色通道的平均亮度和像素的亮度值之间进行插值处理。从而，需要计算几何平均亮度值$\overline{L}_v$，以及红绿蓝通道的几何平均值（$\overline{I}_r$、$\overline{I}_g$和$\overline{I}_b$）。根据这些值，单独计算每个通道每个像素的$f(x,y)$。下面给出了红 537
色通道（$f_r(x,y)$）的计算方程：

$$G_r(x,y) = cI_r(x,y) + (1-c)L_v(x,y)$$
$$\bar{G}_r(x,y) = c\bar{I}_r + (1-c)\bar{L}_v$$
$$f_r(x,y) = aG_r(x,y) + (1-a)\bar{G}_r(x,y)$$

跟前面一样，插值参数a控制着对比程度，而新的插值参数c则提供了一种简单的颜色校正（见图22-24和彩图XIII）。



图22-24　针对颜色校正的线性插值。在左边的图像中参数c设为0.0，在右边的图像中c为1.0（另见彩图XIII）

迄今为止，还没有讨论式（22-3）中的指数n的取值。电生理学研究表明n的取值在0.2到0.9之间（Hood, Finkelstein, & Buckingham, 1979）。虽然该指数可以由用户指定，但是对于大多数图像都可以按照下面的经验公式，根据图像的几何平均亮度$\bar{L}_v$和最小最大亮度（$L_{\min}$和$L_{\max}$），估计出一个合理的值：

$$n = 0.3 + 0.7\left(\frac{L_{\max} - \bar{L}_v}{L_{\max} - L_{\min}}\right)^{1.4}$$

到目前为止，展现的几种S形曲线压缩本质上都是全局的。它们的优点是计算速度快，非常适合于中等动态范围到高动态范围的图像。对于动态范围非常高的图像，可能需要求助于局部算子，因为局部算子能够提供一些额外的压缩。一种最直接的扩展S形曲线压缩的方法是，将全局半饱和常量替换为随空间变化的函数。该函数可以有多种不同的计算方式。

538

换言之，函数$f(x,y)$迄今为止都被假设为常量，但是除此之外还可以作为空间局部化的平均值来计算。对此最简单的方法也许是再次使用高斯模糊后的图像。在模糊化的图像中，每个像素都表示一个局部平均值，可以将该值看作一种合适的半饱和常量选择[1]。

与前一节讨论的基于除法的算子一样，在此也必须考虑光晕失真。然而，在图像除以其自身的高斯模糊化版本时，为了最小化光晕失真要求高斯滤波器的核比较大。而S形曲线与随空间变化的半饱和常量一起使用时，则要求高斯滤波器核比较小，以便最小化失真。这是一种明显的改善，因为少量的高斯模糊可以直接在空域高效计算。也就是不需要诉求于昂贵的傅里叶变换。实际上，几个像素宽度的滤波器核就足以抑制明显的失真，同时在色调映射后的图像中产生更局部化的对比。

高斯模糊的一个潜在问题是，该滤波器以模糊细微细节的同样方式，模糊化对比强烈的边缘。实际中，如果所考虑的像素附近存在大对比梯度，那么高斯模糊后的像素将明显不同于该像素本身。这就是光晕效应的直接原因。在基于除法的方法中，通过非常大的滤波器核将这样的大对比平均化了。

在S形曲线压缩方案中，小的高斯滤波器能最小化强烈对比梯度之间的重叠。在那种情况下，仍然会出现光晕，但是光晕的大小通常不易觉察，相反还被认为是一种增强性的对比。

1. 尽管$f(x,y)$不再是常量，但是仍然将其称为半饱和常量。

另一种模糊图像同时能够最小化大对比间隔附近负面效应的方法是，避免模糊整个边缘。一种简单但是计算量比较大的方法是，以不同大小的内核计算一堆高斯模糊图像。对于每个像素，都选择一个最大的、但不覆盖显著梯度的高斯模糊版本。

在一个相对统一的邻域中，不管滤波器核的大小如何，高斯模糊后的像素值应该是一样的。这样，两个不同高斯滤波器滤波后的像素值之间的差值应该近似为零。只有在较大的滤波器核覆盖了一个包含强烈对比间隔的邻域，而较小滤波器却没有的情况下，该差值才会发生明显的改变。

因此，分析内核大小不同的高斯滤波器，有可能找到一个最大的不包含明显边缘的像素邻域。对于图22-25所示的图像，每个像素所选择的尺度显示在图22-26（左图）中。这种尺度选择机制在照相色调再现算子（Rainhard et.al，2002）以及Ashikhmin算子中得到运用（Ashikhmin, 2002）。



539

图22-25 用来演示图22-26所示的尺度选择机制的示例图像

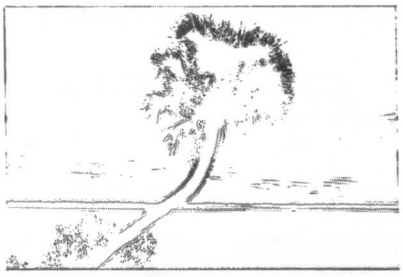


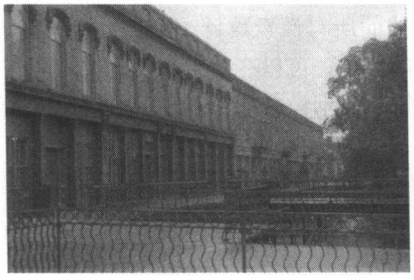


图22-26 尺度选择机制：左图显示了图22-25所示图像的每个像素选择的尺度；像素越暗，尺度越小。计算该图像总共使用了8种不同的尺度。右图是展现了邻域选择机制下每个像素局部平均计算的结果

一旦已知了每个像素的合适邻域，就可以用该邻域的高斯模糊平均值 L_{blur} 来指导半饱和常量，例如在照相再现算子中使用：

$$L_{\text{d}} = \frac{L_{\text{w}}}{1 + L_{\text{blur}}}$$

另一种可论证的更好的办法是采用边缘保留平滑算子。边缘保留平滑算子是专门用来清除细微细节而保持明显对比不变的算子。这样的滤波器有很多种，例如双向滤波器（见图22-27）、三向滤波器、Susan滤波器、LCIS算法、均值平移算法，等等。虽然有些滤波器计算量比较大，但是它们都是合适的（Durand & Dorsey, 2002; Choudhury & Tumblin, 2003; Pattanaik & Yee, 2002; Tumblin & Turk, 1999; Comaniciu & Meer, 2002）。



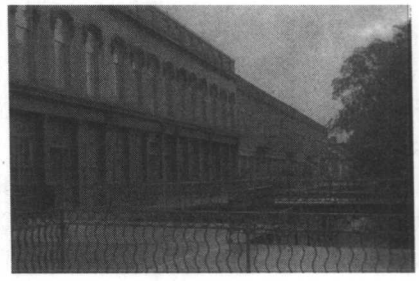


图22-27 S形曲线压缩（左图）以及使用双向滤波计算半饱和常量的S形曲线压缩（右图）。请注意右图天空的对比度提高了

22.10　其他方法

540 到此为止，前面的章节讨论了大部分色调再现算子。但是，还有一两个不能直接归入上述类别的算子。其中最简单的就是对数压缩的变体，另一个就是基于直方图的方法。

只要数值大于1，取对数就能有效降低动态范围。然后采用下面的方程能够将任何正数非线性地缩放到0和1之间：

$$L_d(x,y)=\frac{\log_b(1+L_v(x,y))}{\log_b(1+L_{\max})}$$

上式没有指定对数底数b，任意底数都是可行的。这种对数底数的自由选择可以用来使底数随输入亮度变化，从而获得更匹配于压缩图像的算子（Drago，Myszkowski, Annen, & Chiba, 2003）。这一方法采用了接受用户参数p的Perlin和Hoffert的偏移函数：

541

$$\text{bias}_p(x)=x^{\log_{10}(p)/\log_{10}(1/2)}$$

让底数b依赖于亮度，并在2和10之间平滑地插值底数，可以将上述对数映射提炼为：

$$L_d(x,y)=\frac{\log_{10}(1+L_v(x,y))}{\log_{10}(1+L_{\max})}\cdot\frac{1}{\log_{10}\left(2+8\left(\left(\frac{L_v(x,y)}{L_{\max}}\right)^{\log_{10}(p)/\log_{10}(1/2)}\right)\right)}$$

对于用户参数p，取0.85左右的初始值能够产生似乎真实的结果（见图22-28右图）。

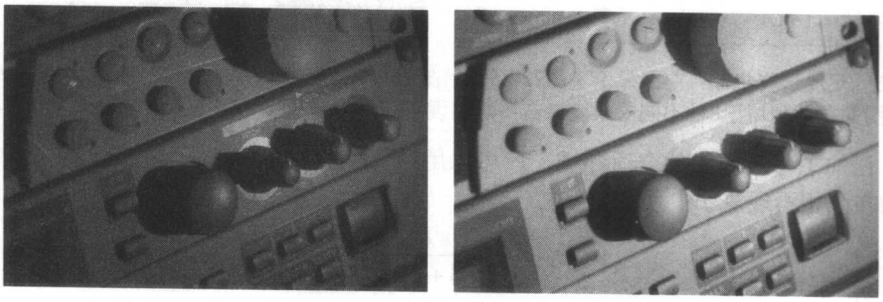


图22-28　以10为底的对数压缩（左图）以及可变底数的对数压缩（右图）

另一种选择是，将色调再现建立在直方图均衡的基础上。传统的直方图均衡针对的是输出图像中每个亮度值的出现概率相等。Greg Ward以保留对比度的方式精炼了这一方法（Ward Larson, Rushmeier, & Piatko, 1997）。

首先，计算高动态范围图像的亮度直方图。根据该直方图，计算累计直方图，使每个区间（bin）都包含亮度值小于或者等于该区间所代表的亮度值的所有像素数目。累计直方图是一个单调递增函数。按照每个区间所表示的亮度值绘制每个区间的值，可以产生一个函数。可以将该函数看作一个亮度映射函数。缩放该函数，使垂直轴支撑整个显示设备的范围，即可输出一个色调再现算子。这一技术称为直方图均衡。

Ward进一步精炼了该方法，确保了该函数的梯度从不超过1。这意味着，如果累计直方图中相邻值之间的差值太大，那么该差值将被钳制为1。这就避免了输入亮度的细微变化导致输出图像产生较大差值的问题。换言之，将累计直方图的梯度限制为1，对比度就不会言过其实。由此
542 形成的算法称为直方图调整（见图22-29）。

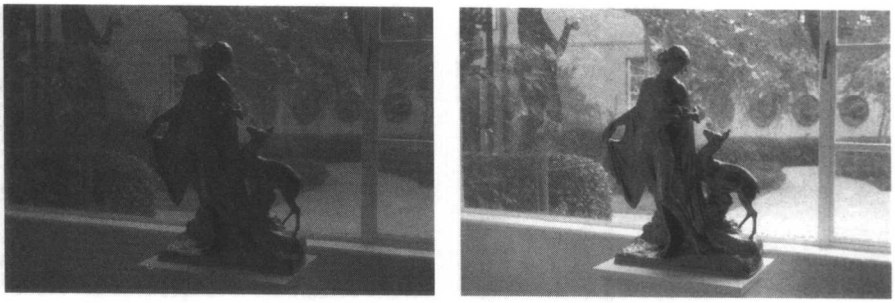


图22-29 线性缩放图像（左图）和直方图调整图像（右图）。该图像的创建得到了Florida州Winter Park的Albin Polasek博物馆的许可

22.11 夜景色调映射

迄今为止所讨论的色调再现算子几乎都假定图像呈现的场景处于适光（photopic）查看条件下，也就是说在正常的光照水平下查看。对于暗视（scotopic）场景，即十分暗的场景，人的视觉系统展示出完全不同的行为。特别是在感知到的对比度较低、视觉灵敏度（即能够区分的最小细节）也较低的情况下，所有东西都呈现稍微泛蓝的外观。

为了能够在处于适光条件下的监视器上正确查看这样的图像，需要对其进行预处理，使其看起来好像适应于非常暗的查看环境。通常采用的预处理形式有：降低亮度和对比度、降低图像的饱和度、蓝色偏移以及视觉灵敏度的降低，等等（Thompson, Shirley & Ferwerda, 2002）。

常见的方法是，先将图像从RGB转换到XYZ。然后针对每个像素计算暗视亮度V：

$$V = Y\left[1.33\left(1+\frac{Y+Z}{X}\right)-1.68\right]$$

然后可以对该单通道图像乘以一个凭经验选择的偏蓝的灰度。图22-30展示一个这样的例子。如果某些像素处于适光范围内，那么将偏蓝的灰度图像与输入图像混合起来就可以创建夜景图像。每个像素的混合比例取决于V值。

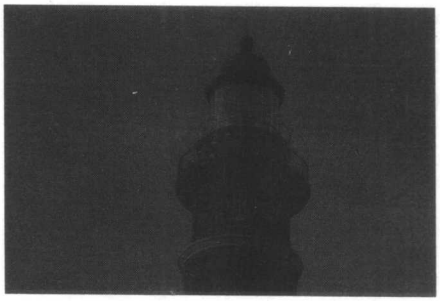


图22-30 使用图22-12所示图像模拟的夜景（另见彩图XI和彩图XV）

视觉灵敏度的降低可以通过夜景图像的低通滤波来建模，虽然这可能产生不正确的模糊场景。更好的办法是采用双向滤波器，在模糊较小细节的同时保留明显的边缘（Tomasi & Manduchi, 1998）。

最后，22.3节列举的颜色迁移技术也可用来将日光图像变换为夜景。这种方法的有效性取决于是否有合适的夜景图像可以进行颜色迁移。作为例子，图22-12中的图像被变换成图22-31所示的夜景图像。

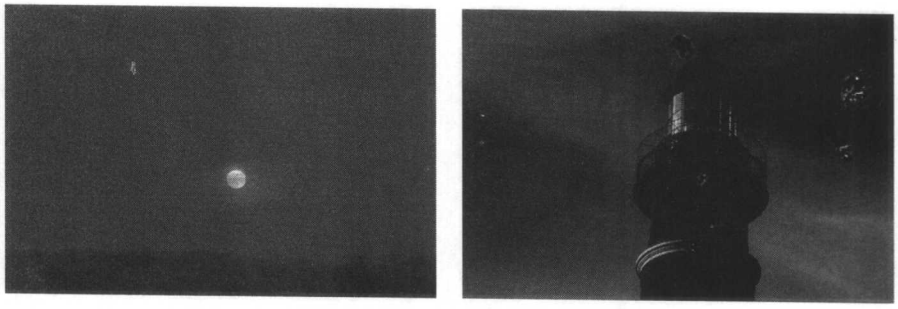


图22-31　左边的图像用来将图22-12中的图像变换成夜景，该夜景显示在右边（另见彩图XIV）

22.12　讨论

由于全局照明算法天然地产生高动态范围图像，因此不可能直接显示由此形成的图像。与其诉求于线性伸缩或钳位过程，不如使用色调再现算子。采用任何色调再现算子都比不用要好。合适的算子的选择取决于具体的应用需求。

例如，实时绘制应用可能采取简单的S形曲线压缩，因为这样的压缩速度快，可以实时运行。另外，它们的视觉质量通常也相当好。直方图调整技术（Ward Larson et al.,1997）也可能快到足以满足实时应用的要求。

对于动态范围特别高的场景，采用局部算子可以实现更好的压缩。然而，计算代价通常也很高，使得这些算子只适合于非交互式应用。局部算子中最快的是双向滤波器，因为该技术能够提供优化算法（Durand & Dorsey,2002）。

该滤波器本身可以作为一个色调再现算子，还可以用来计算S形曲线压缩函数中使用的局部自适应级别。不管哪种情况，该滤波器都尊重明显的对比变化，而将细微的对比平滑掉。这是一个有助于减小光晕失真的重要特性。光晕失真是局部算子的常见问题。

最小化光晕失真的另一种方法是照相色调再现算子所采用的尺度选择机制（Reinhard et al., 2002），虽然该技术计算速度比较慢。

总之，目前虽然有大量的色调再现算子，但是根本不同的方法只有少数。傅里叶域和梯度域的算子都源于图像的形成机理。空域算子要么是空间可变的（局部），要么本质上是全局的。这些算子通常都是建立在对人的视觉系统（和很多物种的视觉系统）的洞察基础之上。

544 ~ 545

第23章

全局照明

在现实世界中很多表面接受的入射光线大部分或者是全部来自其他反射表面，这通常称为间接光照（indirect lighting）或者是相互照明（mutual illumination）。例如，大部分房间天花板几乎没有直接来自光源（发光的物体）的照明。图23-1展现了照明的直接成分和间接成分。

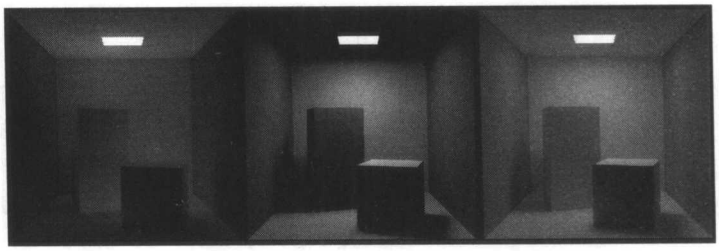


图23-1 左边和中间的图像中，间接和直接光照被分别隔离开来了。在右边的图像中，则将两个分量都加在一起。全局照明算法考虑了直接光照和间接照明

虽然考虑光在表面之间的相互反射是简单明了的问题，但是其潜在代价却很高，因为所有表面都可能反射任一给定的表面，其结果是N个表面的相互作用达到$O(N^2)$。因为整个全局物体数据库都可能照亮任一给定的物体，所以考虑间接照明的问题常被称为全局照明（global illumination）问题。

考虑全局照明问题的文献丰富而复杂（例如，Appel，1968; Goral, Torrance, Greenberg, & Battaile, 1984; Cook et al., 1984; Immel et al., 1986; Kajiya, 1986; Malley, 1988）。本章将选取两个算法作为例子加以讨论：粒子跟踪和路径跟踪。第一个算法可用于预排应用程序，例如迷宫游戏，并作为分批绘制的一部分。第二个算法可用于现实的分批绘制。其后，我们将讨论“直接”光照的分离。在直接光照中光只在照明器和照相机之间弹跳一次。

23.1 朗伯场景的粒子跟踪

请回忆19.2节的传输方程：

$$L_s(\boldsymbol{k}_o) = \int_{所有\boldsymbol{k}_i} \rho(\boldsymbol{k}_i, \boldsymbol{k}_o) L_f(\boldsymbol{k}_i) \cos\theta_i \mathrm{d}\sigma_i$$

该方程的几何关系显示在图23-2中。当被照亮的点为朗伯反射点时，该方程可以简化为

$$L_s = \frac{R}{\pi} \int_{所有\boldsymbol{k}_i} L_f(\boldsymbol{k}_i) \cos\theta_i \mathrm{d}\sigma_i$$

其中R为漫反射率。有限元方法是求该方程逼近解的方法之一。首先，可以将场景分成N个表面，每个都具有未知的表面辐射率L_i、反射率R_i以及发射辐射率E_i。这样可以得到N个同时发生的线性方程组：

$$L_i = E_i + \frac{R_i}{\pi}\sum_{j=1}^{N} k_{ij} L_j$$

其中k_{ij}是一个与原始积分表示有关的常量。求解该线性方程组，可以绘制N个颜色固定的多边形。
548 这一有限元方法通常被称为辐射度法（radiosity）。

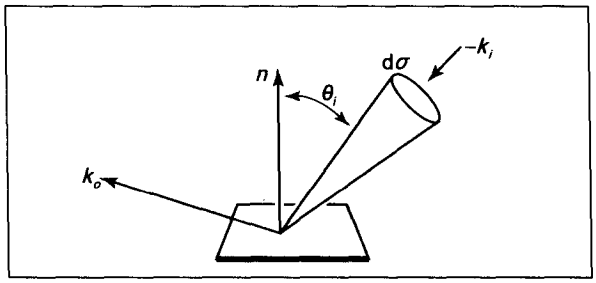


图23-2　方向性形式的传输方程的几何关系

可用来替换辐射度的方法是统计模拟方法。该方法随机地跟踪照明器发出的光“粒子”穿过整个环境。这种方法称为粒子跟踪（particle tracing）。有很多算法采用了粒子跟踪的形式。本书将讨论一种粒子跟踪形式。该跟踪将光存储在三角形的纹理中。首先，回顾一下基本的放射测量关系。面积为A的朗伯表面的辐射率L直接与单位面积上的入射功率成正比，即

$$L = \frac{\Phi}{\pi A} \tag{23-1}$$

其中Φ为表面的输出功率。注意，在本讨论中，所有辐射测量量不是光谱就是RGB，具体情况取决于实现。如果表面的发射功率为Φ_e，入射功率为Φ_i，且辐射率等于R，那么该方程将变为

$$L = \frac{\Phi_e + R\Phi_i}{\pi A}$$

如果给定模型上每个三角片的Φ_e和R，那就可以从每个照明器开始逐一计算每个照明器的粒子烘烤功率（firing power）。可以将一种纹理映像与每个三角形联合起来，存储累计的辐射率。开始时所有纹理单元（texel）都初始化为

$$L = \frac{\Phi_e}{\pi A}$$

如果三角形面积为A且包含n_t个纹理单元，有一个功率为ϕ的粒子击中该三角形，那么那个纹理单元的辐射率就会增加：

$$\Delta L = \frac{n_t \phi}{\pi A}$$

一旦某个粒子击中某个表面，就增加粒子所击中的纹理单元的辐射率，并随机地决定是否反射该粒子。如果反射该粒子，则选择一个方向并调整其功率。

注意该粒子必须在某个点上终止。对于每一个表面，都可以针对与之相互作用的每个表面赋予反射概率p。自然的选择是令$p = R$，因为R本质上就是与光相关的。然后，这样的粒子分散到整个环境中，在被吸收之前，既不会获得任何能量，也不会损失任何能量。当粒子是单一波长时，这种方法能工作得很好（Walter, Hubbard, Shirley, & Greenberg, 1997）。但是，当光线携带的是光谱或RGB三元组时，这也是通常所实现的情况，就不存在单一的R（Jensen，2001），此时应该为p值选择某种折中的结果。考虑到可能的粒子衰减，反射粒子的功率ϕ'应该调整为

$$\phi' = \frac{R\phi}{p}$$

549

注意 p 可以设置为任意小于1的正常量，而且对于每个交互作用该常量都可以不同。对于给定的波长，当 $p > R$ 时，粒子将获得该波长的功率；当 $p < R$ 时，则会丢失那一波长的功率。获得功率的情况将不会干扰收敛性，因为只要 $p < 1$，粒子就会停止分散并在某点终止。在本讨论的后续部分，将令 $p = 0.5$。图23-3展现了这样一个系统中的一个单一粒子的路径。

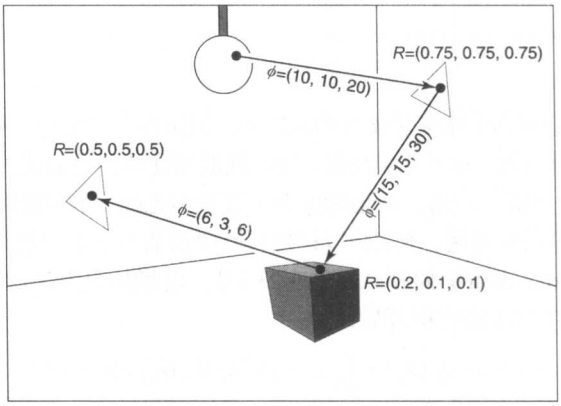


图23-3 生存概率为0.5的粒子路径。该粒子在最后一次交互作用中被吸收。每段路径旁边显示的是RGB功率

这一算法的关键是，针对朗伯表面以合适的概率分布散布光线。就像14.4.1节所讨论的那样，可以按照下式变换两个规范的随机数 (ξ_1, ξ_2)，得到一个余弦（朗伯）分布矢量：

$$\boldsymbol{a} = \left(\cos(2\pi\xi_1)\sqrt{\xi_2}, \sin(2\pi\xi_1)\sqrt{\xi_2}, \sqrt{1-\xi_2}\right) \tag{23-2}$$

注意这里假定了法向矢量平行于 z 轴。对于三角形，必须建立一个 $\boldsymbol{w}$ 平行于法向矢量的标准正交基。这可以按照下式来完成：

$$\boldsymbol{w} = \frac{\boldsymbol{n}}{\|\boldsymbol{n}\|}$$

$$\boldsymbol{u} = \frac{\boldsymbol{p}_1 - \boldsymbol{p}_0}{\|\boldsymbol{p}_1 - \boldsymbol{p}_0\|}$$

$$\boldsymbol{v} = \boldsymbol{w} \times \boldsymbol{u}$$

550

其中 $\boldsymbol{p}_i$ 为三角形的顶点。然后，按照定义，上述矢量在合适的坐标系下可以表示为：

$$\boldsymbol{a} = \cos(2\pi\xi_1)\sqrt{\xi_2}\,\boldsymbol{u} + \sin(2\pi\xi_1)\sqrt{\xi_2}\,\boldsymbol{v} + \sqrt{1-\xi_2}\,\boldsymbol{w} \tag{23-3}$$

对于单一照明器且 $p = 0.5$ 的情况，该算法的伪代码描述如下：

```
for ( Each of n particles) do
  RGB phi = Φ / n
  compute uniform random point a on luminaire
  compute random direction b with cosine density
  done = false
  while not done do
    if (ray a + tb hits at some point c) then
      add n_t R φ / (π A) to appropriate texel
      if ( ξ1 > 0.5) then
```

$\phi = 2R\phi$

$\boldsymbol{a} = \boldsymbol{c}$

$\boldsymbol{b}$ = random direction with cosine density

else

done = true

其中ξ_i是规范的随机数。只要运行了该代码，纹理映像就会存储每个三角形的辐射率，可以从任意视点直接绘制该纹理映像，而不需要任何附加运算。

23.2　路径跟踪

虽然粒子跟踪适合于预先计算漫射场景的辐射率，但是在创建带有一般BRDF的场景或者是包含很多物体的场景时会有问题。对于这类场景，最直观的创建图像的方式为路径跟踪（path tracing）（Kajiya，1986）。这是一种概率方法，从眼睛所在位置发出光线，然后跟踪它们直达照明器为止。通常路径跟踪只用来计算间接照明。本文将以捕捉所有照明的方式来介绍该方法。这种方式效率不太高，有时称为蛮力（brute force）路径跟踪。在23.3节，将增加更为高效的直接光照技术。

路径跟踪从下面的完全传输方程开始：

551

$$L_s(\boldsymbol{k}_o) = L_e(\boldsymbol{k}_o) + \int_{\text{所有}\mathbf{k}_i} \rho(\boldsymbol{k}_i, \boldsymbol{k}_o) L_f(\boldsymbol{k}_i) \cos\theta_i \mathrm{d}\sigma_i$$

对于每一条查看光线，都可以采用蒙特卡罗积分来逼近该方程的解。请回忆14.3的内容，在那里利用了随机采样来逼近一个积分：

$$\int_{x \in s} g(x)\mathrm{d}\mu \approx \frac{1}{N}\sum_{i=1}^{N}\frac{g(x_i)}{p(x_i)}$$

其中x_i是概率密度函数为p的随机点。如果令$N=1$将这直接应用于上述传输方程，可以得到

$$L_s(\boldsymbol{k}_o) \approx L_e(\boldsymbol{k}_o) + \frac{\rho(\boldsymbol{k}_i, \boldsymbol{k}_o) L_f(\boldsymbol{k}_i) \cos\theta_i \mathrm{d}\sigma_i}{p(\boldsymbol{k}_i)}$$

因此，如果能够按照已知的概率密度p，选择随机方向$\boldsymbol{k}_i$，则可以得到估计值。问题是$L_f(\boldsymbol{k}_i)$本身是未知的。幸运的是，可以朝那个方向发射一条光线，找到该方向上可见的表面，然后用递归方法从统计角度来估计$L_f(\boldsymbol{k}_i)$。当遇到一个照明器并且L_e不为零时就结束递归过程（见图23-4）。该方法假定照明器的反射率为零，否则就要继续递归过程。

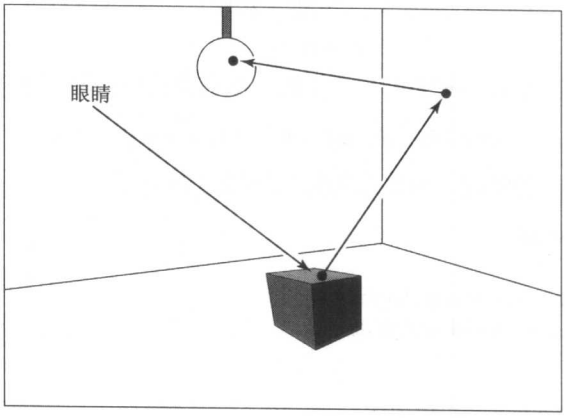


图23-4　光线跟踪。从眼睛发出的一条光线穿过一个像素后，在整个场景中散播，直到最后达到一个照明器为止

对于朗伯 BRDF（$\rho = R/\pi$）情况，可以选择余弦密度函数：

$$p(\boldsymbol{k}_i) = \frac{\cos\theta_i}{\pi}$$

552

该密度函数的方向可以按照式（23-3）来选择。这使估计中的一些余弦项可以抵消，即

$$L_s(\boldsymbol{k}_o) \approx L_e(\boldsymbol{k}_o) + RL_f(\boldsymbol{k}_i)$$

伪代码形式的朗伯表面路径跟踪器，在操作上类似于第10章描述的光线跟踪器。不过raycolor函数需要修改为

```
RGB raycolor (ray a +tb ,int depth)
if (ray hits at some point c ) then
  RGB c = Le (-b)
  if (depth < maxdepth) then
    compute random direction d
    return c + R raycolor(c + sd , depth+1)
else
  return background color
```

其结果将是一幅噪音特别多的图像，除非采用大照明器或者是采样数目非常巨大。注意照明器的颜色必须远大于1（有时上千，甚至好几万），以确保表面最终的颜色接近于1。因为只有那些偶然击中照明器的光线才有效，大部分光线对颜色的贡献几乎都为零。为了生成随机方向**d**，可以使用粒子所采用的同一技术（见式（23-2））。

在一般情况下，可能需要采用光谱颜色或者是更一般的BRDF。在实践中，应该使用材质类的成员函数来计算随机方向以及与该方向关联的概率p。这样材质就可以透明地添加到实现当中。

23.3 精确的直接光照

本节介绍一种直接光照方法。该方法比第9章介绍的方法更为物理化。这些方法将使全局照明算法更高效。核心思想是：按照第10章描述的方法向照明器投射阴影光线，但是在做的过程中根据前一章的传输方程进行细致的簿记。可以调整全局照明算法，确保只计算一次方向分量。例如，在粒子跟踪中，直接来自照明器的粒子将不予考虑，因此粒子将只会编码间接光照。在全局照明的范围内，这使得漂亮阴影的计算要比直接光照更为高效。

553

23.3.1 数学框架

为了计算一个照明器（luminaire）（发光体）对一个非发光表面的直接光照，需要求解19.2节的传输方程：

$$L_s(\boldsymbol{x},\boldsymbol{k}_o) = \int_{\text{所有}\boldsymbol{x}'} \frac{\rho(\boldsymbol{k}_i,\boldsymbol{k}_o)L_e(\boldsymbol{x}',-\boldsymbol{k}_i)v(\boldsymbol{x},\boldsymbol{x}')\cos\theta_i\cos\theta'}{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}\mathrm{d}A' \qquad (23\text{-}4)$$

其中L_e为照明器发射的辐射率；v为能见度函数，如果$\boldsymbol{x}$“看见”$\boldsymbol{x}'$，则等于1，否则等于0；其他变量如图23-5所示。

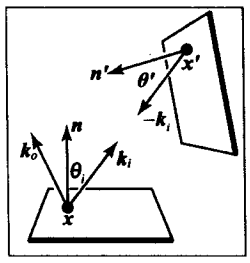


图23-5 式（23-4）的直接光照术语

如果采用蒙特卡罗积分对式（23-4）进行采样，需要在照明器表面随机选取一个密度函数为p的点$\boldsymbol{x}'$（因此有$\boldsymbol{x}' \sim p$）。仅将一个样本代入式（14-5），可以得到

$$L_s(\boldsymbol{x},\boldsymbol{k}_o)\approx\frac{\rho(\boldsymbol{k}_i,\boldsymbol{k}_o)L_e(\boldsymbol{x}',-\boldsymbol{k}_i)v(\boldsymbol{x},\boldsymbol{x}')\cos\theta_i\cos\theta'}{p(\boldsymbol{x}')\|\boldsymbol{x}-\boldsymbol{x}'\|^2} \tag{23-5}$$

如果在照明器上选取均匀分布的随机点，则有 $p=1/A$，其中 A 为照明器的面积，从而得到

$$L_s(\boldsymbol{x},\boldsymbol{k}_o)\approx\frac{\rho(\boldsymbol{k}_i,\boldsymbol{k}_o)L_e(\boldsymbol{x}',-\boldsymbol{k}_i)v(\boldsymbol{x},\boldsymbol{x}')A\cos\theta_i\cos\theta'}{\|\boldsymbol{x}-\boldsymbol{x}'\|^2} \tag{23-6}$$

使用式（23-6）可以直接采样平面（例如矩形）照明器，只需要在每个照明器上选取一个随机点。

单一照明器的代码如下：

```
color directLight (x, k_o, n)
pick random point x' with normal vector n' on light
d = x' − x
k_i = d / ||d||
if (ray x + td has no hits for t < 1− ∈) then
   return ρ(k_i,k_o)L_e(x',−k_i)(n·d)(−n'·d)/||d||^4
else
   return 0
```

554 上述代码需要一些额外的测试，例如将负的余弦值钳位为零。注意 $\|\boldsymbol{d}\|^4$ 项来源于距离的平方以及两个余弦值，例如 $\boldsymbol{n}\cdot\boldsymbol{d}=\|\boldsymbol{d}\|\cos\theta$，因为 $\boldsymbol{d}$ 不一定是单位向量。

图23-6展现了几个软阴影示例。

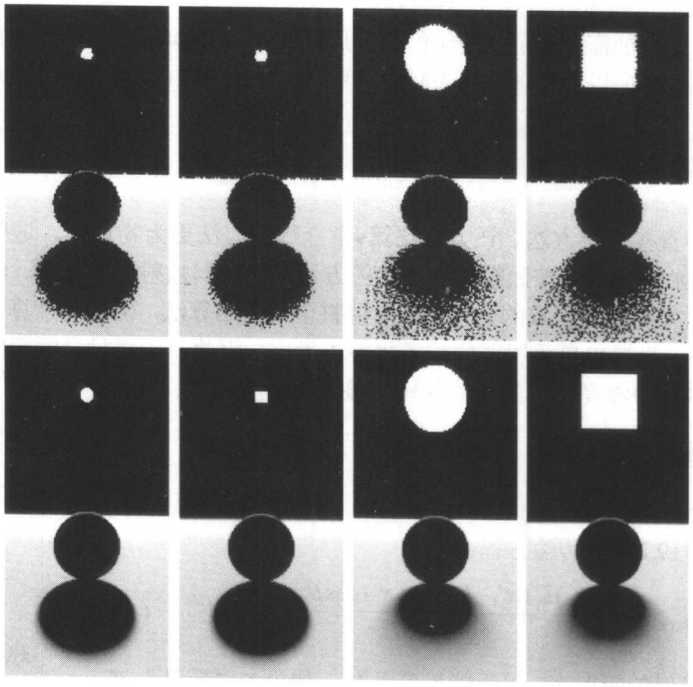


图23-6　使用方形光源和圆形光源从后面照射一个球体时的各种软阴影。上：1个采样；下：100个采样。注意：在决定阴影外观方面，光源的形状没有光源大小那么重要

23.3.2　球形照明器的采样

可以采用式（23-6）对中心 $\boldsymbol{c}$ 为半径为 R 的球进行采样。但是采样产生的图像噪音比较严重，因为很多样本都将在球的后面，而且 $\cos\theta'$ 项变化很大。实际上，可以采用更复杂的 $p(\boldsymbol{x}')$ 来减少

噪音。

第一个可以试用的非均匀密度为 $p(x')\sim\cos\theta'$。可以证明使用这个密度函数的采样复杂度与 $p(x')\sim\cos\theta'/\|x'-x\|^2$ 一样，所以在此采用后者进行讨论。按照这种方式采样照明器，等同于使用常量密度函数 $q(k_i)=$常量 采样。这一常量密度函数定义在点 x 所看到的照明器所指向的空间。 555
下面定义一个原点为 x 的坐标系，以及一组右旋标准正交基 $w=(c-x)/\|c-x\|$、$v=(w\times n)/\|w\times n\|$，如图23-7所示。另外定义相对于$uvw$坐标系的方位角和极角$(\alpha,\phi)$。

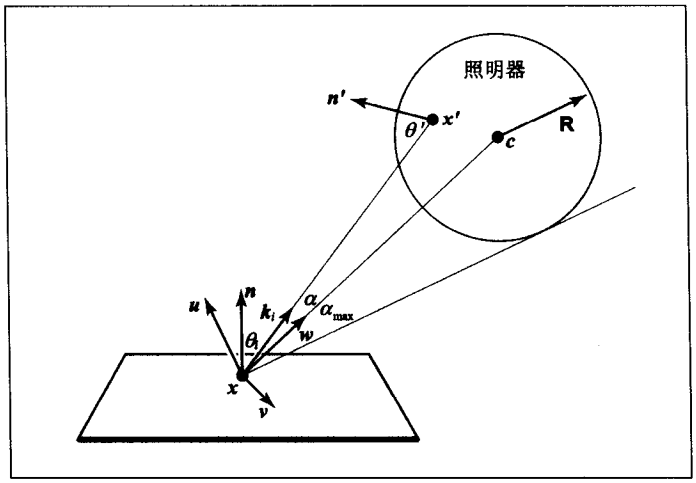


图23-7 球形照明器在点 x 处的直接光照的几何关系

包括球形照明器的最大α由下式给定：

$$\alpha_{\max}=\arcsin\left(\frac{R}{\|x-c\|}\right)=\arccos\sqrt{1-\left(\frac{R}{\|x-c\|}\right)^2}$$

这样，球所指方向（关于立体角）的圆锥体内的均匀密度刚好等于该球所指立体角$2\pi(1-\cos\alpha_{\max})$的倒数，即

$$q(k_i)=\frac{1}{2\pi\left(1-\sqrt{1-\left(\frac{R}{\|x-c\|}\right)^2}\right)}$$

进而可以得到

$$\begin{bmatrix}\cos\alpha\\ \phi\end{bmatrix}=\begin{bmatrix}1-\xi_1+\xi_1\sqrt{1-\left(\frac{R}{\|x-c\|}\right)^2}\\ 2\pi\xi_2\end{bmatrix}$$

上式给出了方向k_i。为了找到实际的点，需要先找到球体上该方向的第一个点。在该方向上的光线是$(x+tk_i)$，其中k_i为 556

$$k_i=\begin{bmatrix}u_x & v_x & w_x\\ u_y & v_y & w_y\\ u_z & v_z & w_z\end{bmatrix}\begin{bmatrix}\cos\phi\sin\alpha\\ \sin\phi\sin\alpha\\ \cos\alpha\end{bmatrix}$$

我们必须计算有关面积测度的概率密度函数 $p(\boldsymbol{x}')$（请回忆定义在立体角空间的密度函数 q）。因为 q 是一个有效的使用 ω 测度的概率密度函数，$\mathrm{d}\Omega = \mathrm{d}A(\boldsymbol{x}')\cos\theta' / \|\boldsymbol{x}' - \boldsymbol{x}\|^2$，所以可以将任何概率密度函数 $q(\boldsymbol{k}_i)$ 与相关的概率密度函数 $p(\boldsymbol{x}')$ 关联起来：

$$q(\boldsymbol{k}_i) = \frac{p(\boldsymbol{x}')\cos\theta'}{\|\boldsymbol{x}' - \boldsymbol{x}\|^2} \tag{23-7}$$

从而求解得到 $p(\boldsymbol{x}')$ 为

$$p(\boldsymbol{x}') = \frac{\cos\theta'}{2\pi\|\boldsymbol{x}' - \boldsymbol{x}\|^2 \left(1 - \sqrt{1 - \left(\frac{R}{\|\boldsymbol{x} - \boldsymbol{c}\|}\right)^2}\right)}$$

对此，图23-8展现了一种比较好的调试情况。

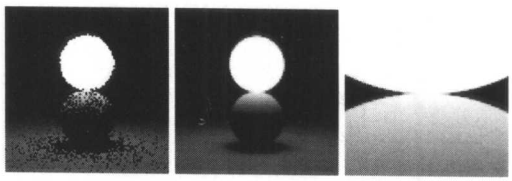


图23-8　一个 $L_e = 1$ 球接触一个反射率为1的球。在两个球接触的地方，反射球应该有 $L(\boldsymbol{x}') = 1$。左：1个采样；中：100个采样；右：100个采样，特写镜头

23.3.3　非漫射照明器

没有任何理由可以限制照明器的亮度不随方向和位置变化。例如，照明器为电视机时，亮度会随位置变化。对于车前灯以及其他方向光源，则会随方向变化。在这种情况下，前几节的分析只需要做少许修改，除了必须将 $L_e(\boldsymbol{x}')$ 修改为 $L_e(\boldsymbol{x}', -\boldsymbol{k}_i)$ 外。使光亮度随方向变化最简单的方法是，采用类似于Phong的有关法向矢量 $\boldsymbol{n}'$ 的模式。为了避免在总的光输出项中使用指数，可以采用如下形式：

557

$$L_e(\boldsymbol{x}', -\boldsymbol{k}_i) = \frac{(n+1)E(\boldsymbol{x}')}{2\pi}\cos^{(n-1)}\theta'$$

其中 $E(\boldsymbol{x}')$ 为点 $\boldsymbol{x}'$ 处的脱离辐射率（radiant exitance）（单位面积上的功率），n 为Phong指数。$n = 1$ 就是漫射光源。如果光源在面积上的分布不均匀，例如电视机，那么 E 将不是常量。

常见问题

- **如果像素值不处于切合实际的0到1的范围内，该如何显示？**

可以采用第22章介绍的任意一种色调再现技术来处理这种情况。

- **在实践中使用了什么样的全局照明技术？**

对于复杂场景的成批绘制，通常采用一级反射路径跟踪。为了增大路径跟踪，通常还采用粒子跟踪进行预处理，就像Jensen所写的书的章节注释所描述的那样。对于预排游戏（walkthrough game）通常采用某种现实空间（world-space）形式的预处理，例如本章所介绍的粒子跟踪。对于非常复杂的镜面传输场景，最佳的选择可能是优雅而棘手的“Metropolis Light Transport”方法（Veach & Guibas, 1997）。

- **环境分量如何与全局照明关联起来？**

对于漫射场景，表面辐射率正比与表面的辐射照度与反射率的乘积。将辐射照度除以π即可逼近环境分量。虽然这只是一个粗糙的毕竟，但是可以将这看作一种猜测方法（M.F. Cohen, Chen, Wallace, & Greenberg, 1988），而且胜于不做任何处理，即令环境分量为零。因为场景内的间接辐射照度可能变化很大，因此为了获得更好的结果，可以对每个表面都采用一个不同的常量，而不是一个全局的环境分量。

- **为什么大多数算法采用传统光线跟踪计算直接光照？**

虽然全局照明算法自动计算直接光照，但实际上让它们只计算间接照明时稍微有些复杂，而单独计算直接光照则更快。其中的原因有三。首先，与直接光照（见图23-1）相比，间接照明 558
有一种光滑趋势，因此可以采用比较粗糙的表示，例如粒子跟踪时低分辨率的纹理映像。第二个原因是光源一般都比较小，采用类似路径跟踪的“视点”方法，很难有机会命中它们。直接的阴影光线则是高效的。第三个原因是直接光照允许分层采样，与非分层采样相比，收敛更快。层次问题是Metropolis Light Transport采用阴影光线的原因，尽管其默认技术在处理直接光照时的稳定性等同于一种路径处理。

- **假定为理想的漫反射和镜面反射，会有什么样的失真？**

对于只有不光滑表面和镜面表面的场景，朗伯/镜面假设是可行的。图23-9展现了那一假设下的绘制结果与照片的比较。

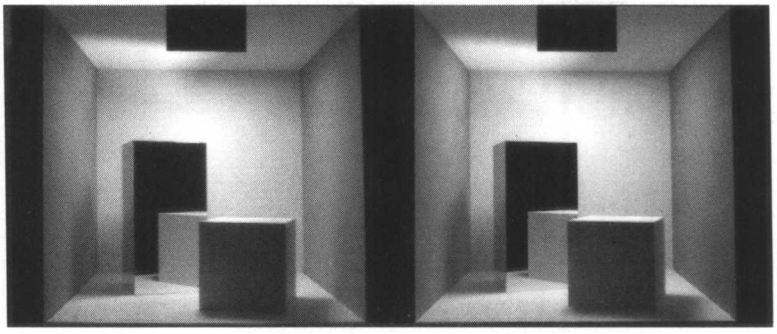


图23-9 照片与绘制结果的比较。图片经Sumant Pattanaik以及*Cornell Program of Computer Graphics*授权使用（另见彩图XXI）

- **每个像素需要多少阴影光线？**

一般在16到400之间。采用狭窄的半影、大的环境项（或者是大的间接分量）以及掩蔽纹理（Ferwerda, Shirley, Pattanaik, & Greenberg, 1997）可以降低所需要的数目。

- **如何对带有金属反射器的灯丝状照明器进行采样？其中大部分光都是从灯丝反射的。**

通常将整个灯替换为一个能逼近其整体效果的简单光源。对于查看光线，可以采用复杂的光源。这样车头灯对于观察者可能看起来复杂，但是照明代码可能将其当作一个简单的圆盘状发光体。 559

- **是否有某种类似于天空的照明器？**

是的，可以将天空当作一个照明器。不过，这么大的光源对直接光照可能没有帮助，而蛮力技术可能工作得更好。

文献注记

全局照明的根源在热传输和照明工程领域，文献*Radiosity: A Programmer's Perspective*

（Ashdown,1994）阐述了这一领域。其他与全局照明有关的好书包括*Radiosity and Global Illumination*（M. F. Cohen & Wallace, 1993）、*Radiosity and Realistic Image Synthesis*（Sillion & Puech, 1994）、*Principles of Digital Image Synthesis*（Glassner, 1995）、*Realistic Image Synthesis Using Photon Mapping*（Jensen, 2001）、*Advanced Global Illumination*（Dutré, Bala, & Bakaert, 2002），以及*Physically Based Rendering*（Pharr & Humphreys, 2004）。本章讨论的概率方法来自*Monte Carlo Techniques for Direct Lighting Calculations*（Shirley, Wang, & Zimmerman, 1996）。

习题

1. 一个封闭环境，其中每个表面都是一个漫反射器，同时也是一个反射率为R的发射器，发射辐射率为E，那么每个点的总辐射率为多少？提示：对于R=0.5，E=0.25，答案为0.5。这是一个优秀的调试案例。
2. 按照第19章的定义，验证式（23-1）。
3. 如果绘制一个一般大小的房间，纹理分辨率为平方厘米级，为了使每个纹理单元平均有大约1000的命中率，大约需要发射多少粒子？
4. 开发一个方法，对密度均匀的圆盘状物体进行随机采样。
5. 开发一个方法，对密度均匀的三角状物体进行随机采样。

560

6. 开发一个方法，对“天幕穹顶”（半球的内部）进行均匀随机采样。

第24章

反射模型

在第19章的讨论中提到，使用BRDF可以归纳表面的反射性质（Nicodemus, Richmond, Hsia, Ginsberg & Limperis, 1977; Cook & Torrance, 1982）。本章将讨论一些重要的材质性质，以及一些相对比较简单的可以用来捕捉材质性质的模型。计算机图形学使用的BRDF模型很多，本章所呈现的模型仅仅为了给出非漫射BRDF的思想。

24.1 现实世界的材质

很多真实材质在正常查看距离下都有可见的结构。例如，大多数地毯都有显而易见的丝绒面外观。对于我们而言，这样的结构并不是材质性质的组成部分，而是几何模型的组成部分。在正常查看距离下结构的细节是不可见的，但是它却决定了宏观的材质外观，是材质的性质之一。例如，纸中的纤维在放大的情况下呈现复杂的外观，但是当在手臂长度的距离上查看的时候，这种复杂的外观被模糊化为均匀相似的外观。这种包含在BRDF中的微观结构之间的差别，在某种程度上是任意的，而且依赖于视距灵敏度以及对“正常”查看距离的定义。但是在实践中已经证明这种区别非常有用。

本节将定义一些材质类别。稍后将介绍针对每一类材质的反射模型。在本章结尾的文献注记中还讨论了另外一些模型。这些模型考虑更为奇异的材质。

561

24.1.1 光滑绝缘材质和金属

绝缘材质是指光亮的折射光的材质，第10章概述了它们的基本属性。金属对光的反射和折射的方式非常类似于绝缘材质，但是它们对光的吸收非常非常快。这样，只有非常薄的金属片才是透明的，例如镀在一些玻璃物体上的薄金。对于光滑材质，只有两个重要属性：

(1) 在每个入射角和波长上有多少光被反射？

(2) 对于给定的距离和波长，光在穿越材质时有多大一部分被吸收？

透射光的数量等于未被反射的光的数量（能量守恒的结果）。对于金属，实际上可以假定所有的光都被立即吸收。对于绝缘材质，被吸收的部分由Beer定律所采用的常量决定，第10章讨论了这一内容。

反射光的数量由第10章讨论的菲涅尔公式决定。这些公式简单易懂，但是有些笨重。菲涅尔公式的主要效果是反射率随着入射角的增加而增加，特别是在入射余角附近。这一效果也适合于透射光（transmitted light）。图24-1以图表方式展现了这些思想。注意光的

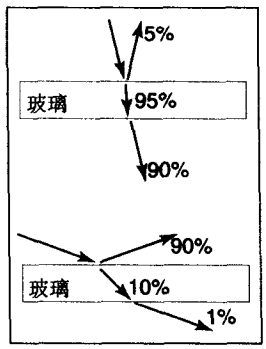


图24-1 玻璃在不同角度下反射光和透射光的数量

反射和折射是不断进行的，如图24-2所示。通常只有一两个反射图像是可以顺利看到的。

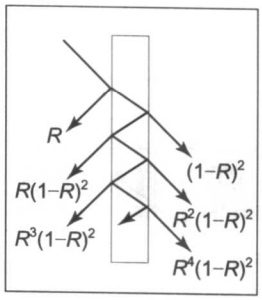


图24-2　光不断地被玻璃反射和折射，旁边显示的是能量比例

24.1.2　粗糙表面

如果将金属或绝缘体粗糙化到很小的程度，但是却不会出现衍射现象，那就可以将其看作具有*微平面*（microfacet）的表面（Cook & Torrance, 1982）。这样的表面在近距离上具有镜面行为，但是在较远的距离上查看则好像以分布状态将光传播开。对于金属，拉过绒的钢，或者是大多数铝箔的“模糊”面，都是这种粗糙表面的例子。

对于绝缘体，例如一张玻璃，刻痕或其他不规则表面特征将使该玻璃模糊反射图像和透射图像，而这些图像通常都是能看清的。如果表面严重刮伤，则将其称为*半透明*（translucent），而不是透明的。这种区分有点武断，但是将玻璃看作半透明还是透明，通常都是很清楚的。

562

24.1.3　漫射材质

如果一种材质不光滑，也就是不发亮，那就是漫射材质。我们所看到的很多表面都是漫射的。例如大部分石头、纸张以及未琢磨的木材。对于简单逼近，可以采用朗伯（常量）BRDF来逼近漫射表面。实际的漫射材质在入射余角处通常变为某种镜面反射。这是一种微妙的效应，但可能对现实很重要。

24.1.4　透明材质

很多薄物体，比如树叶和纸张，对光的反射和发射都很广泛。对于所有实际用意图，这些物体都不会透射清晰的图像。这些表面能够给透射光增添一个色调偏移。例如，红纸呈现红色，是因为在光透入纸内一个很短的距离后，非红光的光被滤除了，然后红光被散射回来。红纸还透射红色调的光，也是源于同样的机理，但是透射的光穿透了纸张。这一性质蕴含了透射系数在两个方向应该一样。

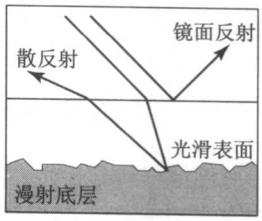


图24-3　击中分层表面的光可能被镜面反射，或者是透射到表面内部，然后在底层漫射开

24.1.5　分层材质

很多表面是由多“层”构成的，或者是嵌入了粒子使表面具有漫射性质的绝缘体（Phong，1975）。这类材质的表面能像图24-3所示的那样进行镜面反射，并且遵循菲涅尔公式。渗透进来的光要么被吸收，要么就是散射回绝缘体的表面，在那可能透射过去，也可能透射不过。透射光、散射光以及在相反方向上再次透射的光，一起构成散“反射”成分。

注意散射分量也随入射角度的增加而衰减，因为入射角增加时菲涅尔公式会使部分光线被反射回到表面内部，如图24-4所示。这样BRDF就不是一个常量，而是一个在入射余角附近突然消失的量。这样更合适。

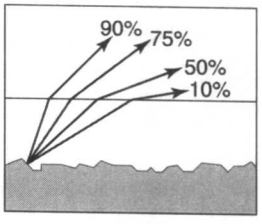


图24-4　随着入射角的增加，底层散射的光越来越少，最终可能会在表面上消失

24.2 反射模型的实现

19.1.6节介绍的BRDF模型将产生一种绘制方法。这种方法与基于点光源和类似Phong模型的绘制方法相比，有更多的物理基础。不幸的是，实际的BRDF一般都十分复杂，而且不能从最初的原理中推导出来。实际上，必须对它们进行测量并用原始数据来直接逼近它们，或者是以经验 563
方式粗略地逼近它们。通常采用的是经验策略。这种逼近模型的开发仍然是一个研究领域。本节将讨论这种逼近模型的一些理想性质。

第一，物理约束包含了BRDF模型的两个属性。第一个是能量守恒约束：

$$\text{对于所有}\boldsymbol{k}_i, R(\boldsymbol{k}_i)=\int_{\text{所有}\boldsymbol{k}_o}\rho(\boldsymbol{k}_i,\boldsymbol{k}_o)\cos\theta_o \mathrm{d}\sigma_o \leqslant 1$$

如果从任一方向 $\boldsymbol{k}_i$ 向表面投射一束光，那么在所有方向上反射的光的数量最多等于入射光的数量。第二个物理性质是期望所有BRDF都有相互作用的，即：

$$\text{对于所有}\boldsymbol{k}_i,\boldsymbol{k}_o, \rho(\boldsymbol{k}_i,\boldsymbol{k}_o)=\rho(\boldsymbol{k}_o,\boldsymbol{k}_i)$$

第二，希望清晰地分离漫射分量和镜面反射分量。其原因是：虽然理想的镜面反射分量在数学上有清楚的 δ 函数公式表示，但是在实践中 δ 函数必须当作一种特殊情况来实现。这种特殊情况只有在BRDF模型明确地指出哪些是镜面反射、哪些是漫射时才是可行的。

第三，期望直观的参数。例如，Phong模型使用这么长久的原因之一就是，该模型的漫射常量和指数都明确地与表面的直觉性质有关，也就是与表面颜色和高亮区大小有关。

最后，期望BRDF函数服从蒙特卡罗采样。请回忆第14章的内容，积分可以通过 N 个随机点 $x_i \sim p$ 对其进行采样，其中 p 定义为积分的测度：

$$\int f(x)\mathrm{d}\mu \approx \frac{1}{N}\sum_{j=1}^{N}\frac{f(x_j)}{p(x_j)}$$

另外由19.2节的内容可知，方向 $\boldsymbol{k}_o$ 上的表面辐射率是由传输公式给定的，即

$$L_s(\boldsymbol{k}_o)=\int_{\text{所有}\boldsymbol{k}_i}\rho(\boldsymbol{k}_i,\boldsymbol{k}_o)L_f(\boldsymbol{k}_i)\cos\theta_i \mathrm{d}\sigma_i$$

如果采用第23章讨论的pdf $p(\boldsymbol{k}_i)$ 对方向进行采样，则可以使用样本来逼近表面辐射率：

$$L_s(\boldsymbol{k}_o)\approx\frac{1}{N}\sum_{j=1}^{N}\frac{\rho(\boldsymbol{k}_j,\boldsymbol{k}_o)L_f(\boldsymbol{k}_j)\cos\theta_j}{p(\boldsymbol{k}_j)}$$ 564

只要在被积函数非零的地方 p 不为零，该逼近就会收敛。然而，只有被积函数与 p 的关系不是太大的话，该逼近才会收敛得好。理想的情况是，$p(\boldsymbol{k})$ 在形状上逼近于被积函数 $\rho(\boldsymbol{k}_j,\boldsymbol{k}_o)L_f(\boldsymbol{k}_j)\cos\theta_j$。实际中，$L_f$ 很复杂。能够实现的最佳选择是让 $p(\boldsymbol{k})$ 的形状在某种程度上类似于 $\rho(\boldsymbol{k},\boldsymbol{k}_o)L_f(\boldsymbol{k})\cos\theta$。

例如，如果表面为朗伯类型，那么BRDF就是常量，“理想的” $p(\boldsymbol{k})$ 正比于 $\cos\theta$。因为 p 的积分必须等于1，因此可以推导出前面的常量：

$$\int_{\text{所有}\theta<\pi/2\text{的}\boldsymbol{k}} C\cos\theta \mathrm{d}\sigma = 1$$

这蕴含了 $C=1/\pi$，从而得到

$$p(\boldsymbol{k})=\frac{1}{\pi}\cos\theta$$

只要 p 在被积函数非零的时候不是太小，就可以得到一个可以接受的高效实现。这样，下面的pdf常量也将具备条件：

$$p(\boldsymbol{k})=\frac{1}{2\pi}$$

这再次强调了对于一个给定的BRDF有很多可接受的pdf。

24.3　镜面反射模型

对于金属，通常指定正常入射角处的反射率$R_o(\lambda)$。该反射率应该按照菲涅尔公式变化，下式对此给出了一个很好的逼近（Schlick, 1994a）：

$$R(\theta,\lambda)=R_0(\lambda)+(1-R_0(\lambda))(1-\cos\theta)^5$$

该逼近允许根据数据或者目测来设置金属的正常反射率。

对于绝缘体，也有同样的反射率公式。不过，可以按照折射率$n(\lambda)$来设置$R_o(\lambda)$：

$$R_0(\lambda)=\left(\frac{n(\lambda)-1}{n(\lambda)+1}\right)^2$$

通常n不会随波长变化，但是对于散射很重要的应用场合，n可以变。经常用到的折射率包括水
565 （$n=1.33$）、玻璃（$n=1.4\sim1.7$）和钻石（$n=2.4$）。

24.4　光滑分层模型

诸如塑料或擦亮的木材之类的无光泽/有金属光泽的材质，其表面反射是由菲涅尔公式支配的，在表面下的反射则由散射决定。在图24-5的透视图中瓷砖里面可以看到这种反射例子。请注意镜面反射中的模糊大部分都是垂直的，这是由查看方向上外观间隔碰撞压缩所造成的。这一效用导致了垂直条纹反射。晴朗的天气下可以在湖面看到这一现象。对此，可以采用明确的微观几何以及简单的光滑表面反射模型来建模，也可以采用更一般的考虑了这种不对称特性的模型。

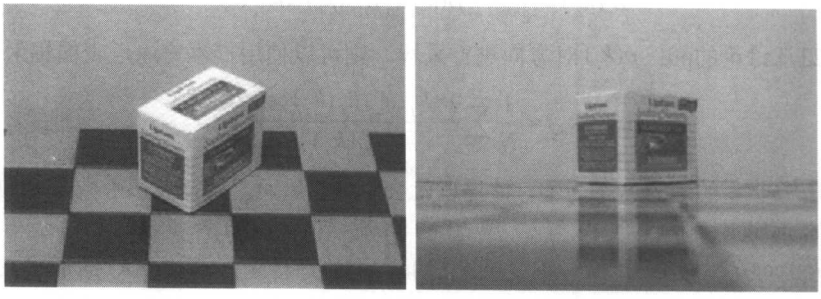


图24-5　抛光瓷砖采用耦合模型的绘制结果。这些图像是采用蒙特卡罗路径跟踪器建立的。漫射项的采样分布为$\cos\theta/\pi$

对于瓷砖可以采用传统的朗伯–镜面模型，线性混合镜面项和朗伯项。按照标准的辐射度术语，这可以表示为

$$\rho(\theta,\phi,\theta',\phi'\lambda)=\frac{R_d(\lambda)}{\pi}+R_s\rho_s(\theta,\phi,\theta',\phi')$$

其中$R_d(\lambda)$为不光滑项的半球反射率，R_s为镜面反射率，ρ_s为归一化的镜面BRDF（球面上的加权Dirac delta函数）。该公式是一个简化的BRDF版本，其中R_s独立于波长。波长的独立性导致了一个高亮区。该高亮区为照明器的颜色。因此获得擦亮的外观，而不是金属外观。Ward（G. J. Ward,

1992）建议令$R_d(\lambda)+R_s\leqslant 1$，以便保持能量守恒。然而，这种$R_s$为常量的模型，不能展现陡峭查看角度下镜面反射的增加。关键在于：现实世界中无光泽外观与有光泽外观的相对比例会随着查看角度而变化。 566

模拟这种无光泽外观变化的方式之一是，在R_s增加的时候，明确抑制$R_d(\lambda)$，即

$$\rho(\theta,\phi,\theta',\phi',\lambda)=R_f(\theta)\rho_s(\theta,\phi,\theta',\phi')+\frac{R_d(\lambda)(1-R_f(\theta))}{\pi}$$

其中$R_f(\theta)$为光滑表面与空气界面的菲涅尔反射率。该公式的问题是不互逆，交换θ和θ'可以发现这一点。由于乘了$(1-R_f(\theta))$，交换将会改变不光滑阻尼因子的值。镜面项为一缩放的Dirac delta函数，是互逆的，但是不能弥补不光滑项的非互逆性。虽然这一BRDF工作得很好，但是互逆性的缺乏可能导致某些绘制算法出现病态的解。

下面介绍一个能够在无光泽外观和有光泽外观之间进行均衡、同时保持能量守恒和互逆性的模型。由于这一新模型的关键特征是耦合了不光滑缩放系数和光滑缩放系数，因此被称为耦合（coupled）模型（Shirley, Smits, Hu, &Lafortune, 1997）。

外观有光泽的表面通常都是光亮的绝缘体，例如聚氨酯和油，具有某种表面下的结构。反射的镜面（类似于镜子的）分量是由光滑的绝缘表面引起的，而且独立于表面下的结构。该镜面分量的大小受菲涅尔公式支配。

在表面上没有被镜面反射掉的光透射到了表面内。透射光在那或者被下表面吸收，或者是被下表面或颜色反射，并透射回来穿过光滑表面。这一透射光形成了反射的不光滑分量。由于不光滑分量只能包含透射光，所以在总量上会自然地随着角度的增加而减少。

为了避免在物理上似是而非的模型与入射角范围具有很好定性行为的模型之间进行选择，请注意考虑了镜面项$R_f(\theta)$的菲涅尔公式，是直接从绝缘体–空气界面的物理特性推导出来的。因此，问题肯定在于不光滑项。就像所实现的那样，可以成熟地模拟下面散射。但是这一技术不仅代价高而且要求对下表面的结构有详细了解；而这通常是未知的或者是难以测量的。实际上，可以将不光滑项修改为一个简单的逼近项，以此来捕获图24-4所示的重要的角度定性行为。

假设不光滑项不是朗伯函数，而是别的只依赖于θ、θ'和λ的函数：$\rho_m(\theta,\theta',\lambda)$。为了简化，抛弃了依赖于$\phi$或$\phi'$的行为。尽量将公式适度简化，因为不光滑项的物理意义很复杂，有时还需 567
要一些未知参数。一般期望不光滑项接近于常量，并且大体上是旋转对称的（He et al., 1992）。

对于某常量k和不光滑反射率参数$R_m(\lambda)$，一个显而易见的针对不光滑分量$\rho_m(\theta,\theta',\lambda)$的候选者是可分离的$kR_m(\lambda)f(\theta)f(\theta')$。可以将$k$和$R_m(\lambda)$合并为一个单独项，而让他们分离是为了使$R_m(\lambda)$的设置更直观——所有波长的$R_m(\lambda)$都必须在0和1之间。可分离的BRDF有一些计算优势，因此采用如下可分离的模型：

$$\rho(\theta,\phi,\theta',\phi',\lambda)=R_f(\theta)\rho_s(\theta,\phi,\theta',\phi')+kR_m(\lambda)f(\theta)f(\theta')$$

不光滑分量只能包含未被表面（镜面）反射的能量。这意味着在$R_m(\lambda)=1$时入射能量与反射能量相同，从而有如下针对每个入射角θ和λ的BRDF约束：

$$R_f(\theta)+2\pi kf(\theta)\int_0^{\frac{\pi}{2}}f(\theta')\cos\theta'\sin\theta'\mathrm{d}\theta'=1 \qquad (24\text{-}1)$$

从中可用看到$f(\theta)$必须正比于$(1-R_f(\theta))$。如果假设吸收一些能量的不光滑分量具有理想的方向模式，则可以得到如下形式的BRDF：

$$\rho(\theta,\phi,\theta',\phi',\lambda)=R_f(\theta)\rho_s(\theta,\phi,\theta',\phi')+kR_m(\lambda)\left[1-R_f(\theta)\right]\left[1-R_f(\theta')\right]$$

现在可以插入完整的菲涅尔公式得到$R_f(\theta)$，然后通过能量守恒求解常量k。实际上将采用24.1.1节讨论的逼近，得到

$$f(\theta) \propto (1-(1-\cos\theta)^5)$$

运用式（24-1）产生

$$k = \frac{21}{20\pi(1-R_0)} \tag{24-2}$$

从而完整的耦合BRDF为

$$\rho(\theta,\phi,\theta',\phi',\lambda) = \left[R_0 + (1-\cos\theta)^5(1-R_0)\right]\rho_s(\theta,\phi,\theta',\phi') + kR_m(\lambda)\left[1-(1-\cos\theta)^5\right]\left[1-(1-\cos\theta')^5\right] \tag{24-3}$$

568

图24-5展现了耦合模型的运行结果。注意，对于高的视点，镜面反射几乎是不可见的，但是在低角度的照片图像中却是清晰可见的。在后一种情况下，不光滑的行为不太明显。

对于合理的折射率，R_0大约被限制在0.03到0.06的范围内（图24-5的$R_0 = 0.05$）。在传统的Phong模型中很难选择R_s的值，因为该值必须根据静态图像的视点或者是特殊的动画照相序列来调节。因此，该耦合模型在“脱手”模式下更容易使用。

24.5　粗糙分层模型

如果表面是光滑的，那么前面的模型就很好。但是，如果表面是非理想情况，则需要在镜面分量中增加一些扩散。本节介绍耦合模型针对这种情况的扩展（Ashikhmin & Shirley,2000）。表面任一给定点的BRDF都是两个方向的函数，一个方向指向光源，另一个方向指向查看者。一般期望有某个适合于“普通”表面的BRDF模型，比如金属和塑料，并且具有如下特性：

(1) **看似真实的**：按照Lewis的定义（Lewis，1994），这是指BRDF遵守能量守恒以及互逆性。

(2) **各向异性**：材质应该将简单的各向异性模型化，就像在擦拭过的金属上所看到的那样。

(3) **直观的参数**：对于诸如塑料之类的材质，应该有基底的反射率R_d、正常的镜面反射率R_s，以及两个粗糙程度参数n_u和n_v。

(4) **菲涅尔行为**：镜面反射应该随着入射角的减小而增加。

(5) **非朗伯漫射项**：材质应该允许有一个漫射项，但是该分量应该是非朗伯类型的，以确保在出现菲涅尔行为的情况下能量守恒。

(6) **蒙特卡罗友谊**：应该有一些合理的概率密度函数，允许针对BRDF进行直接了当的蒙特卡罗采样。

569

具有这些属性的BRDF有Phong类型的加权菲涅尔余弦波瓣模型。该模型是各向异性的。

可以再次将该BRDF分解为镜面分量和漫射分量（见图24-6）。相应地可以将该BRDF改写为经典的两部分相加的形式：

$$\rho(\boldsymbol{k}_1,\boldsymbol{k}_2) = \rho_s(\boldsymbol{k}_1,\boldsymbol{k}_2) + \rho_d(\boldsymbol{k}_1,\boldsymbol{k}_2) \tag{24-4}$$

其中第一项考虑了镜面反射（该内容在下一节介绍）。虽然在这个模型中可以为漫射项$\rho_d(\boldsymbol{k}_1,\boldsymbol{k}_2)$采用朗伯 BRDF，但是在24.5.2节将讨论一个更好的解决方案，并在24.5.3节介绍如何实现该模型。读者如果只想实现该模型可以直接跳到那一节。

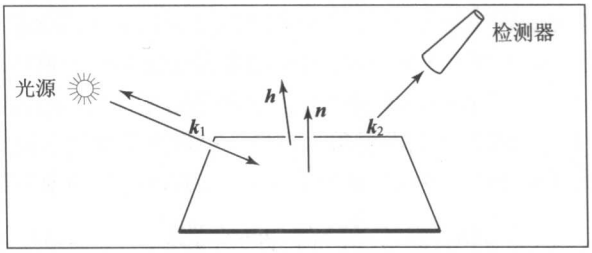


图24-6 反射的几何结构。注意 $\boldsymbol{k}_1$、$\boldsymbol{k}_2$ 和 $\boldsymbol{h}$ 共享一个平面，该平面通常不包括 $\boldsymbol{n}$

24.5.1 各向异性的镜面 BRDF

为了模拟镜面行为，可以采用Phong类型的镜面波瓣，使该波瓣各向异性，并将菲涅尔行为结合近来，与此同时尽量保持初始模型的简单特性。这样的BRDF就是：

$$\rho(\boldsymbol{k}_1,\boldsymbol{k}_2)=\frac{\sqrt{(n_u+1)(n_v+1)}}{8\pi}\frac{(\boldsymbol{n}\cdot\boldsymbol{h})^{n_u\cos^2\phi+n_v\sin^2\phi}}{(\boldsymbol{h}\cdot\boldsymbol{k}_i)\max(\cos\theta_i,\cos\theta_0))}F(\boldsymbol{k}_i\cdot\boldsymbol{h}) \tag{24-5}$$

再次将Schlick逼近应用于菲涅尔公式，可得

$$F(\boldsymbol{k}_i\cdot\boldsymbol{h})=R_s+(1-R_s)(1-(\boldsymbol{k}_i\cdot\boldsymbol{h}))^5 \tag{24-6}$$

其中 R_s 为材质在正常入射角的反射率。由于 $\boldsymbol{k}_i\cdot\boldsymbol{h}=\boldsymbol{k}_o\cdot\boldsymbol{h}$，所以该模型是互逆的。通过强制模型项互逆并且能量守恒，可以得到经验模型。在Ashikhmin的论文中给出完全有理化的项，本章的文献注记中列举了该论文。 570

式（24-5）中的镜面BRDF可以用来表示金属表面，其中反射的漫射分量非常小。图24-7展现了一组金属球。这些球处于一个映射了纹理的朗伯平面上。随着参数值 n_u 和 n_v 的改变，球的外观从粗糙的金属逐渐变成几乎完美的镜子，并且从高度地各向异性变化到更为熟悉的类似于Phong的行为。

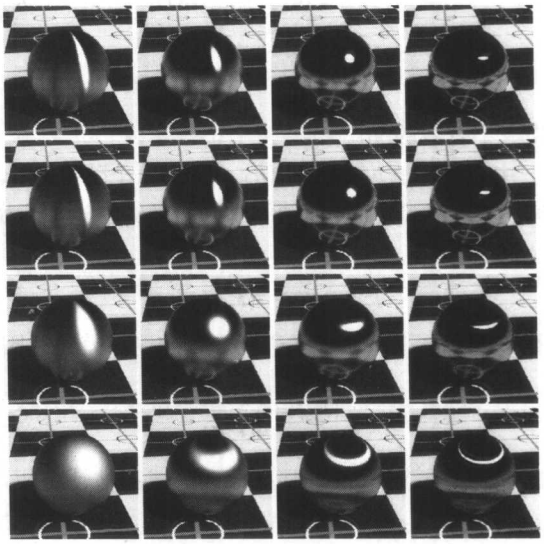


图24-7 金属球，从左到右从上到下各个球的指数依次按照10、100、1 000、10 000的顺序递增

24.5.2 各向异性 Phong 模型的漫射项

可以跟各向异性的镜面反射项一起，对漫射项采用朗伯 BRDF。这也正是大多数模型所采用 571

的办法。但是这不一定能够保持能量守恒。更好的方法是采用简单的依赖于角度的漫射分量，这样就可以考虑漫射传播可用能量变化的事实。其中的能量变化是由于镜面反射项总的反射率依赖于入射角而造成的。特别地，表面的漫射颜色在入射余角附近消失，因为总的镜面反射率接近于1。这一众所周知的效果不能通过朗伯漫射项来再现。因而大多数反射模型都错过了这一效果。

按照类似于耦合模型的方法，可以找到与各向异性的Phong波瓣兼容的漫射项：

$$\rho_d(\boldsymbol{k}_1,\boldsymbol{k}_2)=\frac{28R_{\mathrm{d}}}{23\pi}(1-R_s)\left(1-\left(1-\frac{\cos\theta_i}{2}\right)^5\right)\left(1-\left(1-\frac{\cos\theta_o}{2}\right)^5\right) \tag{24-7}$$

其中 R_d 为正常入射角的漫反射率，R_s 为Phong波瓣系数。图24-8展现了一个使用该模型的例子。

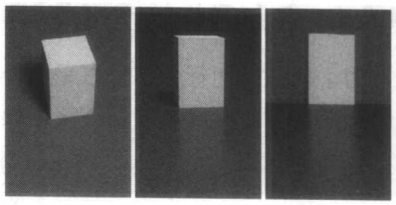


图24-8　带有漫射基底的三视图，$n_u=n_v=400$。请注意镜面反射强调的变化

24.5.3　模型的实现

BRDF由漫射分量和镜面反射分量构成：

$$\rho(\boldsymbol{k}_1,\boldsymbol{k}_2)=\rho_s(\boldsymbol{k}_1,\boldsymbol{k}_2)+\rho_d(\boldsymbol{k}_1,\boldsymbol{k}_2) \tag{24-8}$$

漫射分量由式(24-7)给出；镜面反射分量由式(24-5)给出。由于不一定要调用三角函数来计算指数，因此镜面反射分量可以改写为

572

$$\rho(\boldsymbol{k}_1,\boldsymbol{k}_2)=\frac{\sqrt{(n_u+1)(n_v+1)}}{8\pi}(\boldsymbol{n}\cdot\boldsymbol{h})^{\frac{(n_u(\boldsymbol{h}\cdot\boldsymbol{u})^2+n_v(\boldsymbol{h}\cdot\boldsymbol{v})^2/(1-(\boldsymbol{hn})^2)}{(\boldsymbol{h}\cdot\boldsymbol{k}_i)\max(\cos\theta_i,\cos\theta_o)}}F(\boldsymbol{k}_i\cdot\boldsymbol{h}) \tag{24-9}$$

在蒙特卡罗设置下，感兴趣的问题是：给定 $\boldsymbol{k}_1$，生成 $\boldsymbol{k}_2$ 的样本，要求分布形状类似于余弦加权的BRDF。注意，对于非常大的被积函数值，如果采样严重不足，会产生严重的问题；而对较小值采样过多在实践中则是可接受的。读者可以验证下面建议的密度函数具有这种性质。

一种合适的构建pdf用于采样的方式是，考虑引起BRDF的那半个矢量的分布。这样的函数有：

$$\rho_h(\boldsymbol{h})=\frac{\sqrt{(n_u+1)(n_v+1)}}{2\pi}(\boldsymbol{nh})^{n_u\cos^2\phi+n_v\sin^2\phi} \tag{24-10}$$

其中常量是为了确保其为一个有效的pdf。

可以调整式（24-10）中的概率密度函数 $p_h(\boldsymbol{h})$，产生一个随机的 $\boldsymbol{h}$。然而，为了计算绘制公式，我们需要反射矢量 $\boldsymbol{k}_o$ 和概率密度函数 $p(\boldsymbol{k}_o)$。因此需要特别注意，如果按照 $p_h(\boldsymbol{h})$ 产生 $\boldsymbol{h}$，然后变换为最终的 $\boldsymbol{k}_o$：

$$\boldsymbol{k}_o=-\boldsymbol{k}_i+2(\boldsymbol{k}_i\cdot\boldsymbol{h})\boldsymbol{h} \tag{24-11}$$

这样得到的 $\boldsymbol{k}_o$ 的密度并不是 $p_h(\boldsymbol{k}_o)$。其原因在于 $\boldsymbol{h}$ 和 $\boldsymbol{k}_o$ 的测量差异。因此实际的密度 $p(\boldsymbol{k}_o)$ 为：

$$p(\boldsymbol{k}_o)=\frac{p_h(\boldsymbol{h})}{4(\boldsymbol{k}_i\boldsymbol{h})} \tag{24-12}$$

注意实现的时候如果已知BRDF为这一模型，那么绘制公式的估计将相当简单，因为很多项都可以取消。

有可能产生一个向量 $\boldsymbol{h}$，使其相应的向量 $\boldsymbol{k}_o$ 指向表面的内侧，即 $\cos\theta_o<0$。这种样本的权重应该设置为0。这种情况对应于镜面波瓣中低于水平线的那一部分，是模型能量损失的主要根源。显然，当 n_u 和 n_v 逐渐变大时，该问题的严重性将逐渐降低。

现在唯一剩下的事情就是，如何利用式（24-10）所示的pdf产生 $\boldsymbol{h}$ 向量。开始时在

$(\theta,\phi)\in\left[0,\frac{\pi}{2}\right]\times\left[0,\frac{\pi}{2}\right]$的范围内使用球面角生成$\boldsymbol{h}$。注意这只是半球的第一象限。给定两个均匀分布在[0,1]之间的随机数(ξ_1,ξ_2)，可以选择

$$\phi=\arctan\left(\sqrt{\frac{n_u+1}{n_v+1}}\tan\left(\frac{\pi\xi_1}{2}\right)\right) \qquad (24\text{-}13)$$

573

然后利用这一ϕ值按照下式获取θ：

$$\cos\theta=(1-\xi_2)^{1/(n_u\cos^2\phi+n_v\sin^2\phi+1)} \qquad (24\text{-}14)$$

为了采样整个半球，可以采用标准操作，将ξ_1映射到四个可能的函数之一，具体的映射取决于ξ_1是否处在[0,0.25)、[0.25,0.5)、[0.5,0.75)或[0.75,1)区间内。例如，$\xi_1\in[0.25,0.5)$，则通过式(24-13)查找$\phi(1-4(0.5-\xi_1))$，然后按照坐标轴$\phi=\pi/2$对其进行“翻转”。这能确保完全覆盖以及层次化。

对于漫射项，可以采用更简单的方法，按照余弦分布产生样本。这十分接近于完全的漫射BRDF，相当大地减少了蒙特卡罗估计的方差。

常见问题

- **图像看起来太光滑，即使采用复杂的BRDF也是如此。问题出在哪？**

BRDF只捕捉子像素级的细节。这种细节太小了以至于眼睛难以分辨。大多数真实表面都有一些细小的可见的变化，例如皮肤皱纹。对于真正的现实，需要某种纹理映射或者是置换映射。

- **如何对带有纹理映射的BRDF进行积分？**

纹理映射可以用来控制表面的任何参数。因此BRDF使用的任何一种颜色或控制参数都是可编程的。

- **我有很漂亮的代码除了材质类以外。我犯了什么错误？**

你可能没犯任何错误。在每个人的程序中，材质类可能是最难看的。如果你找到好的处理办法，请告诉我！我自己的代码采用了着色器体系结构（Hanranhan & Lawson, 1990）。这一结构使材质类包括了大量绘制算法。

文献注记

相关文献描述了很多BRDF模型。本章介绍的只是其中的少数几个。其他模型包括（Cook & Torrance, 1982; He et al.,1992; G. J. Ward, 1992 Oren & Nayar, 1994; Schlick, 1994a; Lafortune, Foo, Torrance, & Greenberg, 1997; Stam, 1999; Ashikhmin Premože & Shirley, 2000; Ershov, Kolchin, & Myszkowski, 2001; Matusik, Pfister, Brand, & McMillan, 2003; Lawrence, Rusinkiewicz, & Ramamoorthi, 2004; Stark, Arvo, & Smits, 2005）。*Making Shaders More Physically Plausible*（Lewis, 1994）讨论了BRDF模型期望拥有的特性。

574

习题

1. 假设不采用朗伯 BRDF，而采用形式为$C\cos^e\theta_i$的BRDF，那么为了保持能量守恒，C该为何值？
2. 习题1的BRDF不是互逆的。能将其修改为互逆的吗？
3. 类似于交通信号的东西被称为*后向反射器*（retroreflector）。这意味着$\boldsymbol{k}_i$和$\boldsymbol{k}_o$互相接近的时候BRDF很大。请在Phong模型的启发下，建立一个模型捕捉后向反射器的行为，同时保持能量守恒和互逆。

575

第25章 基于图像的绘制

视觉真实（visual realism）与交互能力之间的冲突是计算机图形学的一个典型问题。处理该问题的方法之一是：通过一组拍摄的或者是预先计算的实际图像，针对新的视点（viewpoint），插值建立新的图像。这种方法就是基于图像的绘制（image-based rendering），简称为IBR。

577 IBR的基本思想如图25-1所示，图中给出了两幅图像在二维下的情形。给定两幅图像，可以模拟出一幅从新视点看到的图像。模拟图像的质量取决于两幅源图像的细节、物体的基本形状，以及三个视点之间的关系。本章将讨论最蛮力的IBR方法。该方法采用非常规则的样本，进行简单插值。

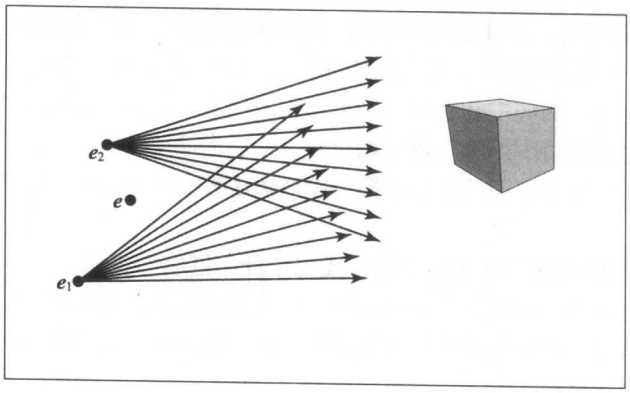


图25-1 给定从e_1和e_2看到的两幅图像，可以使用基于图像的绘制办法生成一幅将从新的视点e看到的逼近图像

25.1 光场

对于空间的每一点，在各个方向上都有光线穿过它。给定点$\boldsymbol{a}$和方向$\boldsymbol{d}$，光线的数量是通过辐射率来衡量的（见19.1.5节）。给定一个点集和所有的方向，可以将每一个位置/方向对上的辐射率描述为一个函数L，即

$$L(\boldsymbol{a},\boldsymbol{d}) \equiv \text{点}\boldsymbol{a}\text{在方向}\boldsymbol{d}\text{上的辐射率}$$

一幅图像只是该函数L在给定的视点$\boldsymbol{a}$和一组结构化的方向$\boldsymbol{d}_i$上的计算值。在技术上，该函数随波长（谱辐射）和时间而变化。通常我们考虑的是该辐射率的RGB矩，而且在时间上是不变
578 的。函数L没有标准的名称，但是在图形学中常称之为*光场*（Light Field）。

光场中有许多结构。最重要的是，在某个固定的方向上L保持不变。图25-2举例说明了这一点。在该图中，

$$L(\boldsymbol{a},\boldsymbol{d}) = L(\boldsymbol{b},\boldsymbol{d}) = L(\boldsymbol{c},\boldsymbol{d}) = L(\boldsymbol{e},\boldsymbol{d})$$

注意，如果在该方向上有某个物体，那么该物体两边点的光场可能不一样。

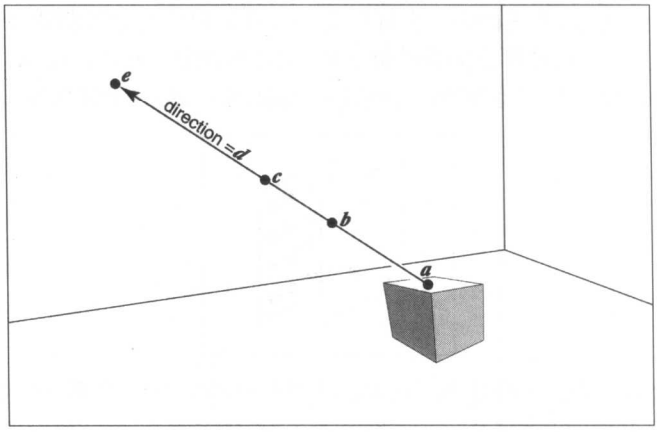


图25-2 光场中的几个点/方向对。由于处在同一条光线上，所以它们都具有同样的值

在逼近一个新视点的光场值之前，需要建立光场的维度（dimensionality）。乍看起来光场好像是5维的，因为它随三维的位置和二维的方向而改变。实际上，这只是特定媒体中光场函数的维度，例如在烟雾中。然而，由于该函数值在同一方向上保持不变，所以可以创建一个线空间（line-space），在三维空间的某个有向元（directed line）上计算该函数值，即

$$L(\boldsymbol{A}) \equiv \text{有向元}\boldsymbol{A}\text{上的辐射率}$$

三维空间的直线是一个4维实体（见14.1.3节）。这意味着可以将光线的辐射率采样存储为4维空间的点。下一节将分析这么做的具体方法。

25.2 根据一组图像创建一幅新图像

只使用一组预先计算的图像，创建新视点图像的关键在于组织用于这一目的的数据（Gortler, Grzeszczuk, Szeliski, & Cohen, 1996; Levoy & Hanrahan，1996）。为了简化，可以创建一个密集采样空间，并做一些简单的插值。像很多图形应用程序一样，我们需要判断直线样本和简单插值是否可行。由14.1.3节的介绍可知，参数化线空间的方法之一是：在一对平行平面上采用两组二维的直线坐标（rectilinear coordinate）。对于一组有限的查看方向，采用一对平行矩形上的直线坐标就可以了。图25-3阐述了这一思想，其中一条特定的光线与四元组(u,v,s,t)关联起来了。 579

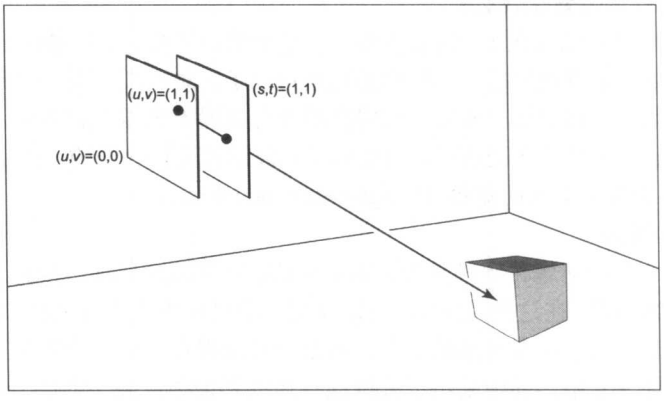


图25-3 任何一条穿过两个矩形的光线都可以参数化两组纹理坐标（texture coordinates）(u,v)和(s,t)，两者分别该光线与矩形的交点

采取这种方式存储辐射率采样的一个好处是，可以采用传统的绘制器（render）聚集采样数据库。对于给定的与单一(u,v)样本关联的位置e，可以像图25-4那样，在(s,t)平面上计算出一组样本（像素）。这样，每个(u,v)位置都可以绘制一幅图像，建立4维数据库。

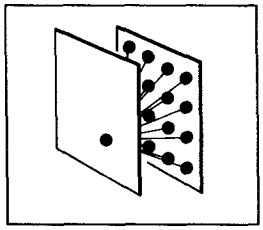


图25-4　使用简单的传统绘制方法创建所有与(u,v)样本关联的(s,t)值

给定一个(u,v,s,t)空间的直线辐射率采样数据库，以及一个新的视点位置，除了平行于参考平面以及位于参考平面后面的那些光线之外，所有光线都将有一个确定的(u,v,s,t)值，如图25-5所示。对于那些穿过两个矩形的光线，该4维点附近的辐射率已经被存储起来了，可以采用某种形式的插值来计算新值。

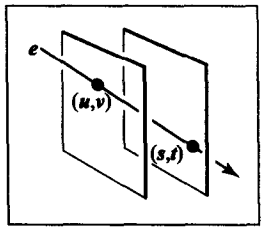


图25-5　从新视点发出穿过两个平面的光线将有一个良定义的(u,v,s,t)值

新图像的质量取决于所采用的插值方案。既可以在(u,v,s,t)空间中最接近的16个数据点之间进行完全四次线性（quadralinear）插值，也可以使用最邻近的插值准则。后者访问的数据量较少，不过会导致图像出现方块效应。

常见问题

- **存储图像的最佳位置是什么？**

组织图像的一个基本难题在于：既要求便于插值运算，又要求结构紧凑能够产生精确的运算结果。为了方便插值，本章描述的简单数据结构就可以了。对于紧凑性，则要求存储的图像尽可能接近所要展示的表面。按照这一限定，只需将图像直接置于物体表面即可，也就是进行纹理映射（texture mapping）。一旦这么处理了，传统的绘制算法就是最合适的。使用图像创建传统几何/纹理模型的过程称为基于图像的建模（image-based modeling）。

- **IBR的应用如何？**

580 这是一个价值千金的问题。到目前为止IBR的实际应用仍然有限。然而，IBR天生的应用可能是虚拟购物和网络教育。在这些应用中，可以使用一组照片来建立引人注目的交互体验。例如，在浏览待售的房屋时，为了获取房屋的印象，就必须移动视点。另一个天生应用就是体育。IBR已经被应用到职业足球赛中，其中采用了线性视点运动和最邻近插值准则（实际上没有插值）。在这样的应用中简单的技术可能起主导作用，因为在图像的捕捉和显示之间的处理时间非常短。也许最有前景的IBR应用是创建各种各样的纹理材质包藏效果（Dana, Ginneken, Nayar, &

Koenderink, 1999)。这不仅比传统的图形学方法更简单，而且能够产生更令人信服的细节。

- **光场定义在表面上吗？**

不透明物体表面上的点在每个方向上都有确定的辐射率。进来的方向是输入辐射率（incident radiance），离开方向是输出辐射率（outgoing radiance）。有时它们分别称为*场辐射率*（field radiance）和*表面辐射率*（surface radiance）。因此，光场就是按照这样的表面来定义的，尽管它不连续（光场在表面内侧为零，而表面光场被定义为表面外侧光场的极限函数）。对于电介质表面，表面辐射率和场辐射率都不同于全球面方向上的光场。在那种情况下，光场需要确切定义的方向集合。

- **什么是深度校正？**

对于不光滑的场景，有时需要利用逼近的几何信息，如果该信息可用的话。图25-6显示了这样一种情况。运用深度校正可以使图像的轮廓更鲜明，但是增加了插值方案的复杂度。

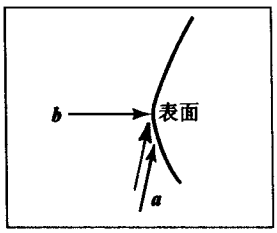


图25-6 虽然灰色的箭头更接近于*a*，但是采用*b*的值可能更合适，因为灰色箭头的入射点在*b*的附近

习题

1. 假定房间大小为5 m×5 m×3 m，按照1 cm^2的分辨率，需要多少像素（texel）才能建立地面、墙和天花板的纹理映射？如果需要存储该房间中心位置附近的光场，并用该光场重构无深度校正的图像，那么需要多少数据点才能重构与传统纹理映射精度一样的新图像？
2. 在4维线空间中，最近邻插值和全4维线性插值各需要多少操作？

581

第26章 可视化

可视化（Visualization）是计算机图形学的主要应用领域之一，在该领域图像被用来帮助用户理解数据（Hansen & Johnson, 2005）。有时这些数据中包含自然产生的几何分量，例如地球上某个区域的海拔数据。别的数据则可能没有明显的几何意义，例如股票市场的趋势。尽管如此，这类非几何数据仍可能从可视化表示中获益，因为人类视觉系统擅长于从图像中提取信息。可视化非几何数据的关键在于如何将数据映射到某种空间形式。一般的非几何数据可视化领域被称为信息可视化（Information Visualization）。本章将只介绍比较好理解的二维和三维标量场可视化问题，其中标量数据值是在一个连续的空间区域定义的。

26.1 二维标量场

为了简化，假设二维标量数据是按照下式定义的：

$$f(x,y)=\begin{cases}1-x^2-y^2, & \text{如果} x^2+y^2<1\\ 0 & \text{其他}\end{cases} \tag{26-1}$$

其中$(x,y)\in[-1,1]^2$。实际上，往往得到的是直线栅格上的采样点。通过对这些采样点的插值可以得到连续的数据场。为了简化，在二维情况下将忽略这一问题。

583

可视化二维场的方法之一是：在有限的取值集合$f(x,y)=f_i$上画线（图26-1显示了等式（26-1）的画线结果）。很多显示海拔高度的地形图都是这么做的。等值线（ioscontour）擅长于表达斜率，但是难以从“全局”的角度把握数据中的大趋势以及极值情况。

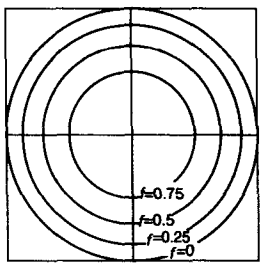


图26-1 函数$1-x^2-y^2$的四级轮廓图

另一种常用的二维数据可视化方法是采用小的伪随机点。这些点的密度与函数值成正比。图26-2显示了本章测试函数的伪随机点可视化结果。这样的随机密度图有利于在黑白媒体上显示，但对可视化而言通常并不是一种好的选择。当采用更多更小的点来维护整个密度时，随机密度图看起来将越来越光滑。当点的大小收缩到人眼的视觉灵敏度之下时，图像看起来就是光滑的。其结果就是该函数的连续灰度色调图（tone plot）。人眼难以阅读这样的图，因为人眼检测绝对强度级（intensity level）的能力有限。由于这一原因，所以经常采用彩色或者是阈值限定的方法。图26-3以灰度形式显示了这一情况。正规地讲，可以指定一个函数g将标量值映射到颜色：

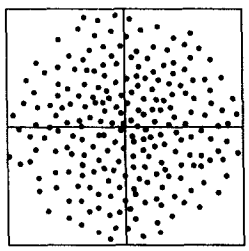


图26-2 函数$1-x^2-y^2$的四级随机密度图

$$g:\mathbb{R}\mapsto[0,1]^3$$

其中$[0, 1]^3$表示RGB立方体。一般的策略是将一组颜色映射到具体的值上，然后在这些值之间对颜色进行插值。常用的颜色集合一般光强度逐渐增加，而色度循环出现，就像下面取值于[0, 1]之间的颜色集合那样：

$$g(0.00) = (0.0, 0.0, 0.0)$$
$$g(0.25) = (0.0, 0.0, 1.0)$$
$$g(0.50) = (1.0, 0.0, 0.0)$$
$$g(0.75) = (1.0, 1.0, 0.0)$$
$$g(1.00) = (1.0, 1.0, 1.0)$$

这些图通常被称为*伪彩色*（pseudocolor）显示。此外还可以像图26-4 那样，将函数显示为高度图。这种图有利于揭示函数的形状。注意该图使函数的球形特征更显而易见。

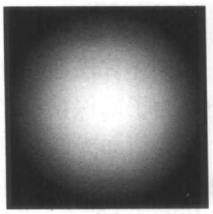


图26-3 函数$1-x^2-y^2$的灰度密度图

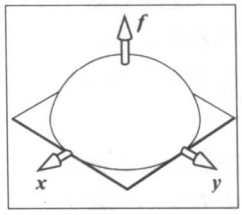


图26-4 函数$1-x^2-y^2$的高度图

一般可以在单一图像中将多种方法结合起来一起使用，如彩色化的或轮廓化的高度图。另一种常用的混合技术是，对高度图进行投影，以正交的方式从上面查看它。这就是立体投影地图（**shaded relief map**），在地理应用中经常采用这种图。

584

26.2 三维标量场

在三维情况下可以采用一些与二维相同的处理技术。例如，可以使用轮廓图，不过此时的轮廓为三维表面，被称之为等值面（isosurface）。还可以将随机密度图推广到三维，方法是在三维空间中散布粒子。如果采用二维伪彩色显示的约束限制，则可以得到直接体绘制（direct volume rendering）。这两种方法本节都涵盖了。但是如何一般化高度图则不明确，因为那已经超出了三维维度的范围。

26.2.1 等值面

给定一个三维标量场$f(x, y, z)$，可以创建一个$f(x, y, z) = f_0$的等值面。在实践中，f是以三维直线表格来定义的，通过该表格可以插值中间结果。图26-5给出了一个示例图像。

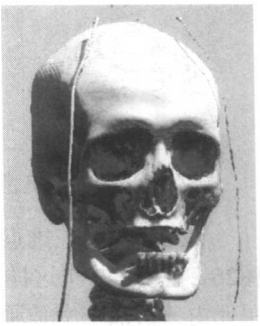


图26-5 源于NIH/NIM Visible Female数据集的等值面

有两种创建等值面图像的基本方法。第一种是显式地创建等值面的多边形表示，然后使用标准的绘制技术绘制该多边形表示。第二种方法是采用光线跟踪技术，通过直接相交计算创建图像。在光线跟踪中不需要计算明确的等值面。当数据集较小，或者是需要等值面本身而不仅仅是等值面图像时，显式的方法更好。对于大数据集而且只需要等值面的图像时，光线跟踪方法更好。

1. 创建多边形等值面

创建多边形等值面的基本思想是将每个直线单元都当作一个单独的问题处理（Wyvill, McPheeters & Wyvill,1986; Lorensen & Cline, 1987）。给定一个等值f_0，如果单元中8个顶点的最大最小值位于f_0的两边，那么该单元中就存在一个等值表面。等值面的形状取决于单元顶点值与f_0的上下位置关系。图26-6给出了三种不同的情况。

顶点与等值点的上下位置关系共有$2^8 = 256$种。可以将所有这些情况都列举在一个表格中，然后查表对照。还可以利用一些对称特性减小表的大小。例如，将上下顶点颠倒一下，就可以将表的大小降低一半。进一步考虑翻转和旋转，则可将该表格的大小减少到16，其中只有15种情况是多边形。

585

2. 光线跟踪

上述算法通常称为步进立方体（marching cubes），虽然简单优雅，但是为了确保精确结果必须采取一些特别措施（Nielson，2003）。

等值面光线相交算法分为三个阶段：投射一条光线穿过不包含等值面的单元；当与包含等值面的体素相交的时候，以解析方式计算该等值面；对得到的交点进行投影操作（Lin & Ching, 1996; Parker, Parker, et al., 1999）。该过程将分别针对屏幕上的每个像素点不断重复。

为了找到交点，光线 $\boldsymbol{a}+t\boldsymbol{b}$ 将穿过体数据中的单元，检查每个单元的数据范围是否包含等值点。如果包含，则进行解析计算，求解与等值面相交处的光线参数 t，即令

$$\rho(x_a+tx_b, y_a+ty_b, z_a+tz_b)-\rho_{\text{iso}}=0$$

当采用离散栅格点之间的三次线性插值逼近 ρ 时，该方程将展开成以 t 为变量的三次多项式。然后可以按闭环形式求解该三次方程，找到该光线与单元中等值面的交点。只有那些包含在单元内部的多项式的根才需要检查。可能存在多重根，对应着多个交点。在这种情况下，使用最小的 t 值（最接近于眼睛的）。多项式也可能没有解，此时光线错过了该单元中的等值面。

586

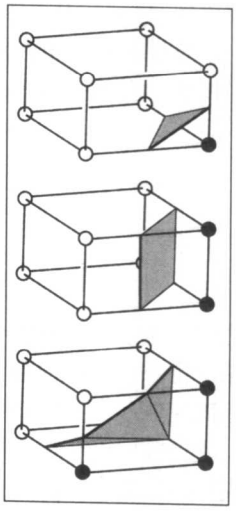


图26-6　多边形等值面的三种情况。黑色顶点在等值点的一侧，白色的在另一侧

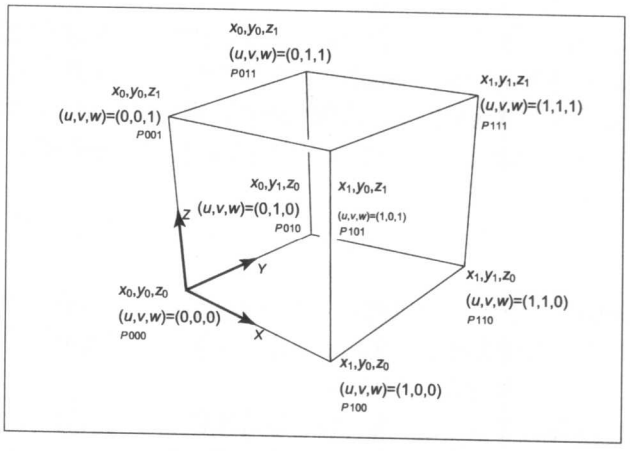


图26-7　单元的几何形状。为了使插值计算更清晰，使用了一个“漂亮的”uvw坐标系

直线体是由三维点采样数组构成的，采样点按照笛卡儿坐标对齐，均匀分布在指定的空间中。图26-7显示了一个这样的体单元。将该图中的指标 (i,j,k) 分别替换为0和1，可以生成别的单元。单元内部的点密度是通过三线性（trilinear）插值得到的：

$$\begin{aligned}\rho(u,v,w)=&(1-u)(1-v)(1-w)\rho_{000}\\&+(1-u)(1-v)(\quad w)\rho_{001}\\&+(1-u)(\quad v)(1-w)\rho_{010}\\&+(\quad u)(1-v)(1-w)\rho_{100}\\&+(\quad u)(1-v)(\quad w)\rho_{101}\\&+(1-u)(\quad v)(\quad w)\rho_{011}\\&+(\quad u)(\quad v)(1-w)\rho_{110}\\&+(\quad u)(\quad v)(\quad w)\rho_{111}\end{aligned}\qquad (26\text{-}2)$$

其中

$$u=\frac{x-x_0}{x_1-x_0}$$
$$v=\frac{y-y_0}{y_1-y_0} \tag{26-3}$$
$$w=\frac{z-z_0}{z_1-z_0}$$

注意

$$1-u=\frac{x_1-x}{x_1-x_0}$$
$$1-v=\frac{y_1-y}{y_1-y_0} \tag{26-4}$$
$$1-w=\frac{z_1-z}{z_1-z_0}$$

如果再令$u_0=1-u$，$u_1=u$，v_0,v_1,w_0,w_1保持不变，则有（见图26-8）

$$\rho=\sum_{i,j,k=0,1}u_i v_j w_k \rho_{ijk}$$

真正的三线性等值面可能相当复杂。图26-9给出了一种特殊情况，立方体的两个对角点处于等值面上与其他六个顶点不同的一边。这完全不同于多边形等值面给出的两个三角形。与三线性表面直接求交的一个好处就是可以消除这种模糊。 587

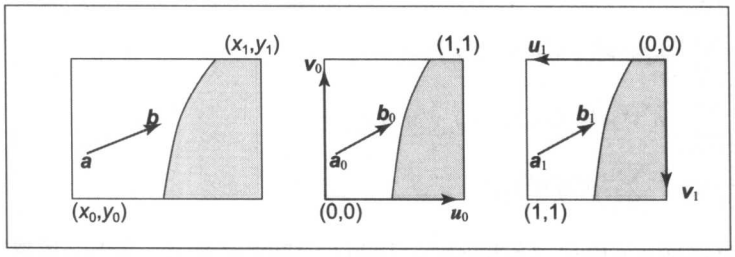


图26-8 插值与相交所采用的各种坐标系

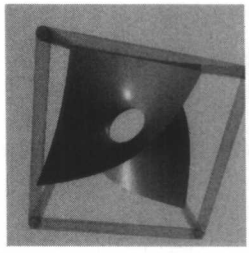


图26-9 采用直接光线跟踪生成的真实的三线性等值面

给定单元中的点(x,y,z)，该点的表面法线是由梯度给出的，即

$$N=\vec{\nabla}\rho=\left(\frac{\partial\rho}{\partial x},\frac{\partial\rho}{\partial y},\frac{\partial\rho}{\partial z},\right)$$

从而，法向量$(N_x,N_y,N_z)=\vec{\nabla}\rho$为

$$N_x=\sum_{i,j,k=0,1}\frac{(-1)^{i+1}v_j w_k}{x_1-x_0}\rho_{ijk}$$
$$N_y=\sum_{i,j,k=0,1}\frac{(-1)^{j+1}u_i w_k}{y_1-y_0}\rho_{ijk}$$
$$N_z=\sum_{i,j,k=0,1}\frac{(-1)^{k+1}u_i v_j}{z_1-z_0}\rho_{ijk}$$

给定光线 $\boldsymbol{p}=\boldsymbol{a}+t\boldsymbol{b}$，将与等值面相交于 $\rho(\boldsymbol{p})=\rho_{\text{iso}}$。将该光线转换到由 (u_0,v_0,w_0) 确定的坐标系中，可得 $\boldsymbol{p}_0=\boldsymbol{a}_0+t\boldsymbol{b}_0$，$\boldsymbol{p}_1=\boldsymbol{a}_1+t\boldsymbol{b}_1$。这样该光线就处在两个坐标系中（如图26-8所示）：

$$\boldsymbol{a}_0=(u_0^a,v_0^a,w_0^a)=\left(\frac{x_1-x_a}{x_1-x_0},\frac{y_1-y_a}{y_1-y_0},\frac{z_1-z_a}{z_1-z_0}\right)$$

588

$$\boldsymbol{b}_0=(u_0^b,v_0^b,w_0^b)=\left(\frac{x_b}{x_1-x_0},\frac{y_b}{y_1-y_0},\frac{z_b}{z_1-z_0},\right)$$

这两个方程是不同的，因为 $\boldsymbol{a}_0$ 表示的是位置，而 $\boldsymbol{b}_0$ 表示方向。对于 $\boldsymbol{a}_1$ 和 $\boldsymbol{b}_1$ 也有类似的结果：

$$\boldsymbol{a}_1=(u_1^a,v_1^a,w_1^a)=\left(\frac{x_a-x_0}{x_1-x_0},\frac{y_a-y_0}{y_1-y_0},\frac{z_a-z_0}{z_1-z_0}\right)$$

$$\boldsymbol{b}_1=(u_1^b,v_1^b,w_1^b)=\left(\frac{-x_b}{x_1-x_0},\frac{-y_b}{y_1-y_0},\frac{-z_b}{z_1-z_0}\right)$$

注意，所有三条光线中的 t 值都是一样的，穿过单元格进行简单的代数求解即可得到 t 值。与等值面的交点 $\rho(\boldsymbol{p})=\rho_{\text{iso}}$ 出现在

$$\rho_{\text{iso}}=\sum_{i,j,k=0,1}\left(u_i^a+tu_i^b\right)\left(v_i^a+tv_i^b\right)\left(w_i^a+tw_i^b\right)\rho_{ijk}$$

上式可以简化为 t 的三次多项式：

$$At^3+Bt^2+Ct+D=0$$

其中

$$A=\sum_{i,j,k=0,1}u_i^bv_i^bw_i^b\rho_{ijk}$$

$$B=\sum_{i,j,k=0,1}\left(u_i^av_i^bw_i^b+u_i^bv_i^aw_i^b+u_i^bv_i^bw_i^a\right)\rho_{ijk}$$

$$C=\sum_{i,j,k=0,1}\left(u_i^bv_i^aw_i^a+u_i^av_i^bw_i^a+u_i^av_i^aw_i^b\right)\rho_{ijk}$$

$$D=-\rho_{\text{iso}}+\sum_{i,j,k=0,1}u_i^av_i^aw_i^a\rho_{ijk}$$

三次多项式的求解在*Cubic and Quartic Roots*（Schwarze，1990）一书中有介绍。在多个图像学宝典（Graphics Gems）存档网站上都可以找到该书作者的代码。使用该代码需要做两个修改：线性解（原来代码假定A是非零的），并将EQN_EPS参数设置为1.0e-30，以便为大的系数提供最大的稳定性。

26.2.2　直接体绘制

589

另一种创建三维变量场图像的方法是，采用小的非透明球体绘制三维随机密度图。为了简化，可以将小球的颜色看作是固定的，也就是没有反射的光辐射体。这样的随机密度图可以采用小球体和光线跟踪技术直接实现，或者是按照三维点采用传统的图形学API来实现。在二维中，可以令球的大小趋近于零而取极限，其结果就是伪彩色显式的三维模拟，通常被称之为**直接体绘制**（direct volume rendering）（Levoy, 1988; Drebin, Carpenter & Hanrahan, 1988; Sabella,1988; Upson & Keeler, 1988）。

有两个参数影响体绘制的效果，即，球的颜色和球的密度。这些都是由用户指定的**传输函数**（transfer function）控制的：

$$\text{颜色} = c(\rho)$$

$$\text{数目密度} = d(\rho)$$

这里的数目密度（number density）为单位体积内的球体数目。如果假定球体的截面积a很小，沿着视线方向考察一个厚度为Δs的小碟形区域，在该区域中球体互不重叠（如图26-10所示），那么颜色将是：

$$L(s+\Delta s) = (1-F)L(s) + Fc$$

其中F表示从视点方向看过去，碟形区域被小球覆盖的比例。因为该碟形区域非常薄，因此可以忽略球体在视觉上的覆盖效果。这样，覆盖比例可以近似为总的球体截面积与碟形区域面积的比值，即：

$$F = \frac{\mathrm{d}aA\Delta s}{A} = \mathrm{d}a\Delta s$$

从而有：

$$L(s+\Delta s) = (1-\mathrm{d}a\Delta s)L(s) + \mathrm{d}a\Delta sc$$

重排上式的有关项，可得到类似于导数的定义形式：

$$\frac{L(s+\Delta s)-L(s)}{\Delta s} = -\mathrm{d}aL(s) + \mathrm{d}ac$$

如果取极限$\Delta s \to 0$，则得微分方程：

$$\frac{\mathrm{d}L}{\mathrm{d}s} = -\mathrm{d}aL(s) + \mathrm{d}ac$$

对于常量d和c，该方程的解为：

$$L(s) = L(0)\mathrm{e}^{-\mathrm{d}as} + c(1-\mathrm{e}^{-\mathrm{d}as})$$

590

这使得我们可以解析方式计算密度和颜色都固定的区域的色彩。然而，实际上d和c都随光线而变化，因此上述微分方程没有解析解。在实践中通常采用数值方法求解。简单的处理办法是从光线的后面开始，沿着该光线逐步推移，如图26-11所示。

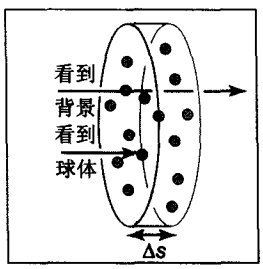


图26-10 充满非透明球体的薄板

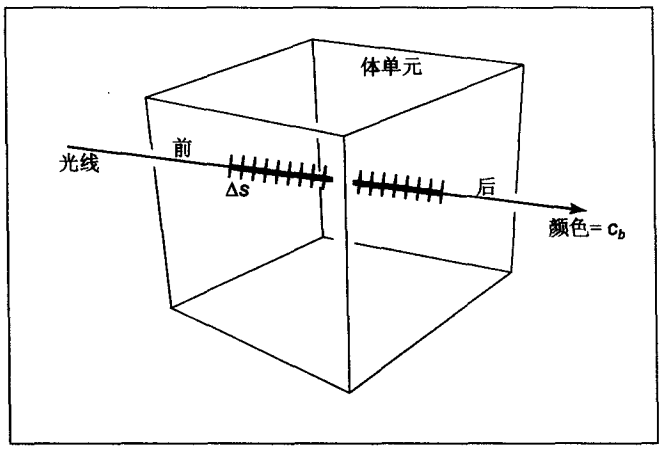


图26-11 对于直接体绘制，可以采用固定步长，沿光线方向进行数值积分

这样可以针对每一个Δs切片运用原始方程，即

$$L(s+\Delta s) = (1-d(x,y,z)a\Delta s)L(s) + d(x,y,z)a\Delta sc(x,y,z)$$

在伪编码情况下，可以将颜色初始化为背景颜色c_b，然后从后到前穿过整个体单元，具体过程如下：

find volume entry and exit points $\boldsymbol{a}$ and $\boldsymbol{b}$
$L = c_b$
Δs = distance ($\boldsymbol{a}$, $\boldsymbol{b}$)
$\boldsymbol{p} = \boldsymbol{b}$
For i = 1 to N **do**
　$\boldsymbol{p} = \boldsymbol{p} - \Delta s(\boldsymbol{b} - \boldsymbol{a})$
　$L = L + (1 - \mathrm{d}(\boldsymbol{p})a\Delta sL + \mathrm{d}(\boldsymbol{p})a\Delta sc(\boldsymbol{p})$

步长Δs将决定积分质量。为了减少变量数目，可以使用新的密度函数$g(\boldsymbol{p}) = \mathrm{d}(\boldsymbol{p})a$。

在有些应用中，采用直接体绘制来绘制一些类似于等值面的内容。在这种情况下，密度传输
591 函数只有“开”或“关”两种状态，而数目密度的梯度被用来计算表面投影法线。这种方法可以产生伪表面图像，该图像对噪声的敏感性弱于传统的等值面绘制图像。

另一种体绘制方法是最大强度投影（maximum-intensity projection）。该方法将每个像素点设置为沿光线方向遇到的最大密度值。这样沿光线的积分被转换为效率更高的搜索。图26-12给出了一幅由最大强度投影产生的图像。



图26-12　NIH/NIM Visible Female数据集的最大强度投影结果。每个像素都包含一个灰度值，该值对应于沿光线方向所遇到的最大密度值。图像经Steve Parker授权使用

常见问题

- **直接体绘制的最佳传输函数是什么？**

该答案取决于具体应用以及数据的特性。可以采用一些经验测试的结果，在（Pfister et al.，2001）中可以找到这方面的内容。（Max, 1995）中介绍了直接体绘制所采用的各种光学模型。

- **如何可视化向量或张量数据？**

向量数据通常采用流线、箭头，或者是线积分卷积（line-integral convolution，LIC）方法进行可视化。Interrante和Grosch（1997）的文献概述了这些方法。张量数据更困难一些。哪怕是最简单的扩散张量数据也难以高效可视化，因为需要映射的数据维数超过了显示维数的范围。关于这方面的内容，请参阅文献（Kindlmann, Weinstein, & Hart, 2000）。

- **如何交互式地改变等值数据查看数据体？**

方法之一是在并行机上采用光线跟踪技术。另外可以采用带预处理的多边形等值面绘制技术，通过预处理帮助搜索包含等值面的单元。可以使用文献（Livnat, Shen, & Johnson，1996）中的数据结构实现这一搜索。

- **体数据是非结构化的四面体，如何提取等值面或者是进行体绘制？**

等值面提取仍然可以按照多边形的方式来实现，但是有少数情况需要预处理。光线跟踪既可用于等值面提取，也可用于体绘制，但是需要改进遍历算法以适应这一非结构化的数据。这既可以通过邻域指针来实现（Garrity, 1999），也可以将单元附加到一个高效的结构上来实现（Parker, Parker, et al., 1999）。 592 ~ 593

- **直接体绘制中的“泼溅（splatting）”是什么意思？**

泼溅（Splatting）是指采用某种绘画算法将半透明的体素投影到屏幕上（Laur & Hanrahan, 1991）。

习题

1. 假设有一个四面体，每个顶点都有一个密度值，那么对于多边形等值面共有多少种情况？
2. 假定在某数据体中有 n^3 个数据元素。如果数据体内的密度是“性能良好的”，那么对于特定的参考值将有多少个单元包含等值面？
3. 是否应该给直接体绘制增加投影操作？为什么？ 594

参 考 文 献

Adelson, E. H. (1999). Lightness Perception and Lightness Illusions. In M. S. Gazzaniga (Ed.), *The New Cognitive Neurosciences* (Second ed., pp. 339-351). Cambridge, MA: MIT Press.

Akenine-Möller, T., & Haines, E. (2002). *Real-Time Rendering* (Second ed.). Wellesley, MA: A K Peters.

Amanatides, J., & Woo, A. (1987). A Fast Voxel Traversal Algorithm for Ray Tracing. In *Proceedings of Eurographics* (pp. 1-10). Amsterdam: Elsevier Science Publishers.

American National Standard Institute. (1986). *Nomenclature and Definitions for Illumination Engineering.* ANSI Report (New York). (ANSI/IES RP-16-1986)

Angel, E. (2002). *Interactive Computer Graphics: A Top-Down Approach with OpenGL* (Third ed.). Reading, MA: Addison-Wesley.

Appel, A. (1968). Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the AFIPS Spring Joint Computing Conference* (Vol. 32, pp. 37-45). AFIPS.

Arvo, J. (1995). *Analytic Methods for Simulated Light Transport.* Unpublished doctoral dissertation.

Ashdown, I. (1994). *Radiosity: A Programmer's Perspective.* New York: John Wiley & Sons.

Ashikhmin, M. (2002). A Tone Mapping Algorithm for High Contrast Images. In *EGRW '02: Proceedings of the 13th Eurographics Workshop on Rendering* (pp. 145-155). Aire-la-Ville, Switzerland: Eurographics Association.

Ashikhmin, M., Premoże, S., & Shirley, P. (2000). A Microfacet-Based BRDF Generator. In *Proceedings of SIGGRAPH* (pp. 65-74). Reading, MA: Addison-Wesley Longman.

Ashikhmin, M., & Shirley, P (2000). An Anisotropic Phong BRDF Model. *journal of graphics tools, 5*(2), 25-32.

Bartels, R. H., Beatty, J. C., & Barsky, B. A. (1987). *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* San Francisco, CA: Morgan Kaufmann.

Baumgart, B. (1974, October). *Geometric Modeling for Computer Vision* (Tech-Rep. No. AIM-249). Palo Alto, CA: Stanford University AI Laboratory.

Bayer, B. E. (1976). *Color Imaging Array.* (U.S. Patent 3,971,065)

Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace Change* (Second ed.). Reading, MA: Addison-Wesley.

Berlin, B., & Kay, P. (1969). *Basic Color Terms: Their Universality and Evolution.* Berkeley, CA: University of California Press.

Blinn, J. (1996). *Jim Blinn's Corner.* San Francisco, CA: Morgan Kaufmann.

Blinn, J., & Newell, M. (1978). Clipping Using Homogeneous Coordinates. In *Proceedings of SIGGRAPH '78* (pp. 245-251). New York: ACM Press.

Blinn, J. F. (1976). Texture and Reflection in Computer Generated Images. *Communications of the ACM, 19*(10), 542-547.

Blinn, J. F. (1978). Simulation of Wrinkled Surfaces. In *Proceedings of SIGGRAPH '78* (pp. 286-292). New York: ACM Press.

Bresenham, J. E. (1965). Algorithm for Computer Control of a Digital Plotter . *IBM Systems Journal, 4*(1), 25-30.

Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., et al. (2004). Brook for GPUs: Stream Computing on Graphics Hardware. *ACM Transactions on Graphics (TOG), 23*(3), 777-786. (ACM SIGGRAPH 2004)

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996) . *Pattern-Oriented Software Architecture* (Vols. 1, A System of Patterns) . New York: John Wiley & Sons.

Campagna, S., Kobbelt, L., & Seidel, H.-P. (1998). Directed Edges—A Scalable Representation for Triangle Meshes. *journal of graphics tools, 3*(4), 1-12.

Carr, N. A., Hall, J. D., & Hart, J. C. (2002). The Ray Engine. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (pp. 37-46). Aire-la-Ville, Switzerland: Eurographics Association.

Catmull, E. (1975). Computer Display of Curved Surfaces. In *IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures* (pp. 11-17). Los Alamitos, CA: IEEE Press.

Chen, S. E., & Williams, L. (1993). View Interpolation for Image Synthesis. In *SIGGRAPH '93: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 279-288). New York: ACM Press.

Chiu, K., Herf, M., Shirley, P., Swamy, S., Wang, C., & Zimmerman, K. (1993). Spatially Nonuniform Scaling Functions for High Contrast Images. In *Proceedings of Graphics Interface '93* (pp. 245-253). Wellesley, MA: A K Peters & Canadian Human-Computer Communications Society.

Choudhury, P., & Tumblin, J. (2003). The Trilateral Filter for High Contrast Images and Meshes. In *EGRW '03: Proceedings of the 14th Eurographics Workshop on Rendering* (pp. 186-196). Aire-la-Ville, Switzerland: Eurographics Association.

Cleary, J., Wyvill, B., Birtwistle, G., & Vatti, R. (1983). A Parallel Ray Tracing Computer. In *Proceedings of the Association of Simula Users Conference* (pp. 77-80).

Cohen, E., Riesenfeld, R. F., & Elber, G. (2001). *Geometric Modeling with Splines: An Introduction.* Wellesley, MA: A K Peters.

Cohen, M. F., Chen, S. E., Wallace, J. R., & Greenberg, D. P. (1988). A Progressive Refinement Approach to Fast Radiosity Image Generation. In *SIGGRAPH '88: Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 75-84). New York: ACM Press.

Cohen, M. F, & Wallace, J. R. (1993). *Radiosity and Realistic Image Synthesis.* Cambridge, MA: Academic Press, Inc.

Comaniciu, D., & Meer, P. (2002). Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(5), 603-619.

Cook, R. L. (1984). Shade Trees. *Computer Graphics, 18*(3), 223-231. (SIGGRAPH '84)

Cook, R. L., Carpenter, L., & Catmull, E. (1987). The Reyes Image Rendering Architecture. *SIGGRAPH Comput. Graph., 21*(4), 95-102.

Cook, R. L., Porter, T., & Carpenter, L. (1984). Distributed Ray Tracing. *Computer Graphics, 18*(3), 137-145. (SIGGRAPH '84)

Cook, R. L., & Torrance, K. E. (1982). A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics, 1*(1), 7-24.

Coombe, G., Harris, M. J., & Lastra, A. (2004). Radiosity on Graphics Hardware. In *GI '04: Proceedings of the 2004 Conference on Graphics Interface* (pp.161-168). Wellesley, MA: A K Peters & Canadian Human-Computer Communications Society.

Crow, F. C. (1978). The Use of Grayscale for Improved Raster Display of Vectors and Characters. In *SIGGRAPH '78: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 1-5). New York: ACM Press.

Crowe, M. J. (1994). *A History of Vector Analysis.* Mineola, NY: Dover.

Curless, B., & Levoy, M. (1996). A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of SIGGRAPH 96* (pp. 303-312). Reading, MA: Addison-Wesley.

Cyrus, M., & Beck, J. (1978). Generalized Two- and Three-Dimensional Clipping. *Computers and Graphics, 3*(1), 23-28.

da Vinci, L. (1970). *The Notebooks of Leonardo da Vinci* (Vol. 1). Mineola, NY: Dover Press.

Dachsbacher, C., Vogelgsang, C., & Stamminger, M. (2003). Sequential Point Trees. *ACM Transactions on Graphics, 22*(3), 657-662. (SIGGRAPH 2003)

Dana, K. J., Ginneken, B. van, Nayar, S. K., & Koenderink, J.J. (1999). Reflectance and Texture of Real-World Surfaces. *ACM Transactions on Graphics, 18*(1), 1-34.

Debevec, P. E., & Malik, J. (1997). Recovering High Dynamic Range Radiance Maps from Photographs. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 369-378). Reading, MA: Addison-Wesley.

De Boor, C. (1978). *A Practical Guide to Splines.* Berlin: Springer-Verlag.

De Boor, C. (2001). *A Practical Guide to Splines.* Berlin: Springer-Verlag.

DeRose, T. (1989). *A Coordinate-Free Approach to Geometric Programming* (Tech. Rep. No. 89-09-16). Seattle, WA: University of Washington.

Dobkin, D. P., & Mitchell, D. P. (1993). Random-Edge Discrepancy of Supersampling Patterns. In *Proceedings of Graphics Interface* (pp. 62-69). Wellesley, MA: A K Peters & Canadian Human-Computer Communications Society.

Dooley, D., & Cohen, M. F. (1990). Automatic Illustration of 3D Geometric Models: Lines. In *SI3D '90: Proceedings of the 1990 Symposium on Interactive 3D Graphics* (pp. 77-82). New York: ACM Press.

Doran, C., & Lasenby, A. (2003). *Geometric Algebra for Physicists.* Cambridge UK: Cambridge University Press.

Drago, F., Myszkowski, K., Annen, T., & Chiba, N. (2003). Adaptive Logarithmic Mapping for Displaying High Contrast Scenes. *Computer Graphics Forum, 22*(3), 419-426.

Drebin, R. A., Carpenter, L., & Hanrahan, P. (1988). Volume Rendering. *Computer Graphics,* 22(4), 64-75. (SIGGRAPH '88)

Durand, F., & Dorsey, J. (2002). Fast Bilateral Filtering for the Display of High-Dynamic-Range Images. *ACM Transactions on Graphics, 21*(3), 257-266.

Dutré, P., Bala, K., & Bekaert, P. (2002). *Advanced Global Illumination.* Wellesley, MA: A K Peters.

Eberly, D. (2000). *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics.* San Francisco, CA: Morgan Kaufmann.

Eberly, D. (2004). *3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic.* San Francisco. CA: Morgan Kaufmann.

Eckman, P., & Friesen, W. V. (1978). *Facial Action Coding System.* Palo Alto, CA: Consulting Psychologists Press.

Ershov, S., Kolchin, K., & Myszkowski, K. (2001). Rendering Pearlescent Appearance Based on Paint-Composition Modelling. *Computer Graphics Forum, 20*(3), 227-238.

Fairchild, M. D. (2005). *Color Appearance Models* (Second ed.). New York: John Wiley & Sons.

Fairchild, M. D., & Johnson, G. M. (2002). Meet iCAM: An Image Color Appearance Model. In *IS&T/SID 10 th Color Imaging Conference* (pp. 33-38). Springfield, VA: Society for Imaging Science & Technology.

Fairchild, M. D., & Johnson, G. M. (2004). The iCAM Framework for Image Appearance, Image Differences, and Image Quality. *Journal of Electronic Imaging, 13,* 126-138.

Farin, G. (2002). *Curves and Surfaces for CAGD: A Practical Guide.* San Francisco, CA: Morgan Kaufmann.

Farin, G., & Hansford, D. (2004). *Practical Linear Algebra: A Geometry Toolbox.* Wellesley, MA: A K Peters.

Farin, G., Hoschek, J., & Kim, M.-S. (Eds.). (2002). *Handbook of Computer Aided Geometric Design.* Amsterdam: Elsevier.

Fattal, R., Lischinski, D., & Werman, M. (2002). Gradient Domain High Dynamic Range Compression. *ACM Transactions on Graphics, 21*(3), 249-256.

Fernando, R. (Ed.). (2004). *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics.* Reading, MA: Addison-Wesley.

Fernando, R., & Killgard, M. J. (2003). *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics.* Reading, MA: Addison-Wesley.

Ferwerda, J. A., Pattanaik, S., Shirley, P., & Greenberg, D. P. (1996). A Model of Visual Adaptation for Realistic Image Synthesis. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (pp. 249-258). New York: ACM Press.

Ferwerda, J. A., Shirley, P., Pattanaik, S. N., & Greenberg, D. P. (1997). A Model of Visual Masking for Computer Graphics. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 143-152). Reading, MA: Addison-Wesley.

Foley, J. D., Van Dam, A., Feiner, S. K., & Hughes, J. F. (1990). *Computer Graphics: Principles and Practice* (Second ed.). Reading, MA: Addison-Wesley.

Forsyth, D. A., & Ponce, J. (2002). *Computer Vision: A Modern Approach.* Englewoods Cliffs, NJ: Prentice Hall.

Francis S. Hill, J. (2000). *Computer Graphics Using OpenGL* (Second ed.). Englewood Cliffs, NJ: Prentice Hall.

Fuchs, H., Kedem, Z. M., & Naylor, B. F. (1980). On Visible Surface Generation by A Priori Tree Structures. *Computer Graphics, 14*(3), 124-133. (SIGGRAPH '80)

Fujimoto, A., Tanaka, T., & Iwata, K. (1986). ARTSccelerated Ray-Tracing System. *IEEE Computer Graphics & Applications, 6*(4), 16-26.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns—Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley.

Garrity, M. P. (1990). Raytracing Irregular Volume Data. In *VVS '90: Proceedings of the 1990 Workshop on Volume Visualization* (pp. 35-40). New York: ACM Press.

Gibson, J. J. (1950). *The Perception of the Visual World.* Cambridge, MA: Riverside Press.

Gilchrist, A. L., Kossyfidis, C., Bonato, F., Agostini, T., Cataliotti, J., Li, X., et al. (1999). An Anchoring Theory of Lightness Perception. *Psychological Review, 106*(4), 795-834.

Glassner, A. (1984). Space Subdivision for Fast Ray Tracing. *IEEE Computer Graphics & Applications, 4*(10), 15-22.

Glassner, A. (1988). Spacetime Ray Tracing for Animation. *IEEE Computer Graphics & Applications, 8*(2), 60-70.

Glassner, A. (Ed.). (1989). *An Introduction to Ray Tracing.* London: Academic Press.

Glassner, A. (1995). *Principles of Digital Image Synthesis.* San Francisco, CA: Morgan Kaufmann.

Goldberg, A., & Robson, D. (1989). *Smalltalk-80: The Language.* Reading, MA: Addison-Wesley.

Goldman, R. (1985). Illicit Expressions in Vector Algebra. *ACM Transactions on Graphics, 4*(3), 223-243.

Goldsmith, J., & Salmon, J. (1987). Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics & Applications, 7*(5), 14-20.

Gooch, A., Gooch, B., Shirley, P., & Cohen, E. (1998). A Non-Photorealistic Lighting Model for Automatic Technical Illustration. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 447-452). New York: ACM Press.

Goral, C. M., Torrance, K. E., Greenberg, D. P., & Battaile, B. (1984). Modeling the Interaction of Light between Diffuse Surfaces. *Computer Graphics, 18*(3), 213-222. (SIGGRAPH '84)

Gortler, S. J., Grzeszczuk, R., Szeliski, R., & Cohen, M. F. (1996). The Lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (pp. 43-54). New York: ACM Press.

Gouraud, H. (1971). Continuous Shading of Curved Surfaces. *Communications of the ACM, 18*(6), 623-629.

Gregory, R. L. (1997). *Eye and Brain: The Psychology of Seeing* (Fifth ed.). Princeton, NJ: Princeton University Press.

Hammersley, J., & Handscomb, D. (1964). *Monte-Carlo Methods.* London: Methuen.

Hanrahan, P., & Lawson, J. (1990). A Language for Shading and Lighting Calculations. In *SIGGRAPH '90: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 289-298). New York: ACM Press.

Hansen, C. D., & Johnson, C. R. (Eds.). (2005). *The Visualization Handbook.* Amsterdam: Elsevier.

Hanson, A. J. (2005). *Visualizing Quaternions.* San Francisco, CA: Morgan Kaufmann.

Harris, M. J. (2004). Fast Fluid Dynamics Simulation on the GPU. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics* (chap. 38). Reading, MA: Addison-Wesley.

Harris, M. J., Baxter, W. V., Scheuermann, T., & Lastra, A. (2003). Simulation of Cloud Dynamics on Graphics Hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (pp. 92-101). Aire-la-Ville, Switzerland: Eurographics Association.

Hausner, M. (1998). *A Vector Space Approach to Geometry.* Mineola, NY: Dover.

Havran, V. (2000). *Heuristic Ray Shooting Algorithms.* Unpublished doctoral dissertation, Czech Technical University in Prague.

He, X. D., Heynen, P. O., Phillips, R. L., Torrance, K. E., Salesin, D. H., & Greenberg, D. P. (1992). A Fast and Accurate Light Reflection Model. *Computer Graphics, 26*(2), 253-254. (SIGGRAPH '92)

Hearn, D., & Baker, M. P. (1986). *Computer Graphics.* Englewood Cliffs, N.J.: Prentice Hall.

Heidrich, W., & Seidel, H.- P. (1998). Ray-Tracing Procedural Displacement Shaders. In *Proceedings of Graphics Interface* (pp. 8-16). Wellesley, MA: A K Peters & Canadian Human-Computer Communications Society.

Hoffmann, B. (1975). *About Vectors.* Mineola, NY: Dover.

Hood, D. C., Finkelstein, M. A., & Buckingham, E. (1979). Psychophysical Tests of Models of the Response Function. *Vision Research, 19*, 401-406.

Hoppe, H. (1994). *Surface Reconstruction from Unorganized Points.* Unpublished doctoral dissertation.

Hoppe, H. (1999). Optimization of Mesh Locality for Transparent Vertex Caching. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 269-276). Reading, MA: Addison-Wesley.

Horn, B. K. P. (1974). Determining Lightness from an Image. *CVGIP, 3*, 277-299.

Hughes, J. F., & Möller,T. (1999). Building an Orthonormal Basis from a Unit Vector. *journal of graphics tools, 4*(4), 33-35.

Hunt, R. W. G. (2004). *The Reproduction of Color.* New York: John Wiley & Sons. (Sixth)

IEEE Standards Association. (1985). *IEEE Standard for Binary Floating-Point Arithmetic* (Tech. Rep.). IEEE Report (New York). (ANSI/IEEE Std 754-1985)

Immel, D. S., Cohen, M. F., & Greenberg, D. P. (1986). A Radiosity Method for Non-Diffuse Environments. *Computer Graphics, 20*(4), 133-142. (SIGGRAPH '86)

Interrante, V., & Grosch, C. (1997). Strategies for Effectively Visualizing 3D Flow with Volume LIC. In *VIS '97: Proceedings of the 8th Conference on Visualization '97* (pp. 421-ff.). Los Alamitos, CA: IEEE Computer Society Press.

ITU. (1990). *International Telecommunication Union ITU-R Recommendation BT. 709, Basic Parameter Values for the HDTV Standard for the Studio and for International Programme Exchange.* Geneva. (Formerly CCIR Rec. 709)

Jackson, R., MacDonald, L., & Freeman, K. (1994). *Computer Generated Colour: A Practical Guide to Presentation and Display.* New York: John Wiley & Sons.

Jansen, F. W. (1986). Data Structures for Ray Tracing. In *Proceedings of a Workshop Eurographics Seminars on Data Structures for Raster Graphics* (pp. 57-73). New York: Springer-Verlag.

Jensen, H. W. (2001). *Realistic Image Synthesis Using Photon Mapping.* Wellesley, MA: A K Peters.

Jensen, H. W., Marschner, S. R., Levoy, M., & Hanrahan, P. (2001). A Practical Model for Subsurface Light Transport. In *Proceedings of SIGGRAPH 2001* (pp. 511-518). Reading, MA: Addison-Wesley.

Johansson, G. (1973). Visual Perception of Biological Motion and a Model for Its Analysis. *Perception & Psychophysics, 14*, 201-211.

Johnson, G. M., & Fairchild, M. D. (2003). Rendering HDR Images. In *IS&T/SID 11 th Color Imaging Conference* (pp. 36-41). Springfield, VA: Society for Imaging Science & Technology.

Kainz, F., Bogart, R., & Hess, D. (2003). The OpenEXR Image File Format. In *SIGGRAPH Technical Sketches.* (see also: http://www.openexr.com/)

Kajiya, J. T. (1986). The Rendering Equation. *Computer Graphics, 20*(4), 143-150. (SIGGRAPH '86)

Kalos, M., & Whitlock, P. (1986). *Monte Carlo Methods, Basics.* New York: Wiley-Interscience.

Kay, D. S., & Greenberg, D. (1979). Transparency for Computer Synthesized Images. *Computer Graphics, 13*(2), 158-164. (SIGGRAPH '79)

Kernighan, B. W., & Pike, R. (1999). *The Practice of Programming.* Reading, MA: Addison-Wesley.

Kersten, D., Mamassian, P., & Knill, D. C. (1997). Moving Cast Shadows Induce Apparent Motion in Depth. *Perception, 26*(2), 171-192.

Kindlmann, G., Reinhard, E., & Creem, S. (2002). Face-Based Luminance Matching for Perceptual Colormap Generation. In *VIS '02: Proceedings of the Conference on Visualization '02* (pp. 299-306). Washington, DC: IEEE Computer Society.

Kindlmann, G., Weinstein, D., & Hart, D. (2000). Strategies for Direct Volume Rendering of Diffusion Tensor Fields. *IEEE Transactions on Visualization and Computer Graphics, 6*(2), 124-138.

Kirk, D., & Arvo, J. (1988). The Ray Tracing Kernel. In *Proceedings of Ausgraph.*

Klatzky, R. L. (1998). Allocentric and Egocentric Spatial Representations: Definitions, Distinctions, and Interconnections. In C. Freksa, C. Habel, & K. F. Wender (Eds.), *Spatial Cognition—An Interdiciplinary Approach to Representation and Processing of Spatial Knowledge* (Vol. 5, pp. 1-17). Berlin: Springer-Verlag.

Knill, D. C. (1998). Surface Orientation From Texture: Ideal Observers, Generic Observers and the Information Content of Texture Cues. *Vision Research, 38*, 1655-1682.

Kollig, T., & Keller, A. (2002). Efficient Multidimensional Sampling. *Computer Graphics Forum, 21*(3), 557-564.

Lacroute, P., & Levoy, M. (1994). Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Proceedings of SIGGRAPH 94* (pp. 451-458). New York: ACM Press.

Lafortune, E. P. F., Foo, S.-C., Torrance, K. E., & Greenberg, D.P. (1997). Non-Linear Approximation of Reflectance Functions. In *Proceedings of SIGGRAPH '97* (pp. 117-126). Reading, MA: Addison-Wesley.

Lakos, J. (1996). *Large-Scale C++ Software Design.* Reading, MA: Addison-Wesley.

Larson, G. W., Rushmeier, H., & Piatko, C. (1997). A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes. *IEEE Transactions on Visualization and Computer Graphics, 3*(4), 291-306.

Lasseter, J. (1987). Principles of Traditional Animation Applied to 3D Computer Animation. *Computer Graphics, 21*(4), 35-44. (SIGGRAPH '87)

Lastra, A., Molnar, S., Olano, M., & Wang, Y. (1995). Real-Time Programmable Shading. In *SI3D '95: Proceedings of the 1995 Symposium on Interactive 3D Graphics* (pp. 59-66). New York: ACM Press.

Laur, D., & Hanrahan, P. (1991). Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. *Computer Graphics, 25*(4), 285-288. (SIGGRAPH '91).

Lawrence, J., Rusinkiewicz, S., & Ramamoorthi, R. (2004). Efficient BRDF Importance Sampling Using a Factored Representation. *ACM Transactions on Graphics, 23*(3), 496-505. (SIGGRAPH '04)

Lee, D. N., & Reddish, P. (1981). Plummeting Gannets: A Paradigm of Ecological Optics. *Nature, 293,* 293-294.

Lefohn, A., Kniss, J., & Owens, J. (2005). Implementing Efficient Parallel Data Structures on GPUs. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General Purpose Computation*(chap. 33). Reading, MA: Addison-Wesley.

Lefohn, A. E., Kniss, J. M., Hansen, C. D., & Whitaker, R. T. (2003). Interactive Deformation and Visualization of Level Set Surfaces Using Graphics Hardware. In *IEEE Visualization* (pp. 75-82). Los Alamitos, CA: IEEE Press.

Leung, T., & Malik, J. (1997). On Perpendicular Texture: Why Do We See More Flowers in the Distance? In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (pp. 807-813). Los Alamitos, CA: IEEE Press.

Levoy, M. (1988). Display of Surfaces from Volume Data. *IEEE Computer Graphics & Applications, 8*(3), 29-37.

Levoy, M. (1990). Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics, 9*(3), 245-261.

Levoy, M., & Hanrahan, P. (1996). Light Field Rendering. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (pp. 31-42). New York: ACM Press.

Lewis, R. R. (1994). Making Shaders More Physically Plausible. *Computer Graphics Forum, 13*(2), 109-120.

Liang, Y.-D., & Barsky, B. A. (1984). A New Concept and Method for Line Clipping. *ACM Transactions on Graphics, 3*(1), 1-22.

Lin, C.-C., & Ching, Y.-T. (1996). An Efficient Volume-Rendering Algorithm with an Analytic Approach. *The Visual Computer, 12*(10), 515-526.

Livnat, Y., Shen, H.-W., & Johnson, C. R. (1996). A Near Optimal Isosurface Extraction Algorithm Using the Span Space. *IEEE Transactions on Visualization and Computer Graphics,* 2(1), 73-84.

Lorensen, W. E., & Cline, H. E. (1987). Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics, 21*(4), 163-169.(SIGGRAPH '87)

Malley, T. (1988). *A Shading Method for Computer Generated Images.* Unpublished master's thesis, Computer Science Department, University of Utah.

Marshall, J. A., Burbeck, C. A., Arely, D., Rolland, J. P., & Martin, K. E. (1999). Occlusion Edge Blur: A Cue to Relative Visual Depth. *Journal of the Optical Society of America A, 13,* 681-688.

Matusik, W., Pfister, H., Brand, M., & McMillan, L. (2003). A Data-Driven Reflectance Model. *ACM Transactions on Graphics, 22*(3), 759-769. (SIGGRAPH '03)

Max, N. (1995). Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics, 1*(2), 99-108.

McCool, M., Toit, S. D., Popa, T., Chan, B., & Moule, K. (2004). Shader Algebra. *ACM Transactions on Graphics, 23*(3), 787-795.(SIGGRAPH '04)

Meyer, G. W., & Greenberg, D. P. (1988). Color-Defective Vision and Computer Graphics Displays. *IEEE Computer Graphics & Applications, 8*(9), 28-40.

Meyers, S. (1995). *More Effective C+ +: 35 New Ways to Improve Your Programs and Designs.* Reading, MA: Addison-Wesley.

Meyers, S. (1997). *Effective C++: 50 Specific Ways to Improve Your Programs and Designs* (Second ed.). Reading, MA: Addison-Wesley.

Mitchell, D. P. (1996). Consequences of Stratified Sampling in Graphics. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (pp. 277-280). New York: ACM Press.

Mitchell, D. P., & Netravali, A. N. (1988). Reconstruction Filters in Computer Graphics. *Computer Graphics, 22*(4), 221-228.

Möller,T., & Haines, E. (1999). *Real-Time Rendering.* Wellesley, MA: A K Peters.

Möller,T., & Hughes, J. (1999). Efficiently Building a Matrix to Rotate One Vector to Another. *journal of graphics tools, 4*(4), 1-4.

Molnar, S., Eyles, J., & Poulton, J. (1992). Pixelflow: High-Speed Rendering Using Image Composition. *Computer Graphics, 26*(2), 231-240. (SIGGRAPH '92)

Mortenson, M. (1985). *Geometric Modeling.* New York: John Wiley & Sons.

Muuss, M. J. (1995). Towards Real-Time Ray-Tracing of Combinatorial Solid Geometric Models. In *Proceedings of BRL-CAD Symposium.*

Nicodemus, F. E., Richmond, J. C., Hsia, J. J., Ginsberg, I., & Limperis, T. (1977). *Geometrical Considerations and Nomenclature for Reflectance* (Tech. Rep. No. 160). Washington, D.C.: National Bureau of Standards.

Nielson, G. M. (2003). On Marching Cubes. *IEEE Transactions on Visualization and Computer Graphics, 9*(3), 283-297.

Olano, M., & Lastra, A. (1998). A Shading Language on Graphics Hardware: The Pixelflow Shading System. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (pp.159-168). New York: ACM Press.

Oppenheim, A. V., Schafer, R., & Stockham, T. (1968). Nonlinear Filtering of Multiplied and Convolved Signals. *Proceedings of the IEEE, 56*(8), 1264-1291.

Oren, M., & Nayar, S. K. (1994). Generalization of Lambert's Reflectance Model. *In Proceedings of SIGGRAPH '94* (pp. 239-246). New York: ACM Press.

Osterberg, G. (1935). Topography of the Layer of Rods and Cones in the Human Retina. *Acta Ophthalmologica, 6*(1), 11-97. (Supplement)

Paeth, A. W. (1990). A Fast Algorithm for General Raster Rotation. In *Graphics Gems* (pp. 179-195). Boston, MA: Academic Press.

Palmer, S. E. (1999). *Vision Science—tons to Phenomenology.* Cambridge,MA: MIT Press.

Parker, S., Martin, W., Sloan, P., Shirley, P., Smits, B., & Hansen, C. (1999). Interactive Ray Tracing. In *ACM Symposium on Interactive 3D Graphics* (pp. 119-126). New York: ACM Press.

Parker, S., Parker, M., Livnat, Y., Sloan, P.-P., Hansen, C., & Shirley, P. (1999). Interactive Ray Tracing for Volume Visualization. In *IEEE Transactions on Visualization and Computer Graphics* (Vol. 5).

Pashler, H. E. (1998). *The Psychology of Attention.* Cambridge, MA: MIT Press.

Pattanaik, S. N., Ferwerda, J. A., Fairchild, M. D., & Greenberg, D. P. (1998). A Multiscale Model of Adaptation and Spatial Vision for Realistic Image Display. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 287-298). New York: ACM Press.

Pattanaik, S. N., & Yee, H. (2002). Adaptive Gain Control for High Dynamic Range Image Display. In *SCCG '02: Proceedings of the 18th Spring Conference on Computer Graphics* (pp. 83-87). New York: ACM Press.

Patterson, J., Hoggar, S., & Logie, J. (1991). Inverse Displacement Mapping. *Computer Graphics Forum, 10*(2), 129-139.

Peachey, D. R. (1985). Solid Texturing of Complex Surfaces. *Computer Graphics, 19*(3), 279-286. (SIGGRAPH '85)

Peercy, M. S., Olano, M., Airey, J., & Ungar, P. J. (2000). Interactive MultiPass Programmable Shading. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 425-432). Reading, MA: Addison-Wesley.

Penna, M., & Patterson, R. (1986). *Projective Geometry and Its Applications to Computer Graphics.* Englewood Cliffs, NJ: Prentice Hall.

Perlin, K. (1985). An Image Synthesizer. *Computer Graphics, 19(3),* 287-296. (SIGGRAPH '85)

Perlin, K., & Hoffert, E. M. (1989). Hypertexture. *Computer Graphics, 23*(3), 253-262. (SIGGRAPH '89)

Pfister, H., Lorensen, B., Bajaj, C., Kindlmann, G., Schroeder, W., Avila, L. S., et al. (2001). The Transfer Function Bake-Off. *IEEE Computer Graphics & Applications, 21*(3), 16-22.

Pharr, M., & Fernando, R. (Eds.). (2005). *GPU Gems 2: Programming Techniques for High-Performance Graphics and General Purpose Computation.* Reading, MA: Addison-Wesley.

Pharr, M., & Hanrahan, P. (1996). Geometry Caching for Ray-Tracing Displacement Maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96* (pp. 31-40). London, UK: Springer-Verlag.

Pharr, M., & Humphreys, G. (2004). *Physically Based Rendering.* San Francisco, CA: Morgan Kaufmann.

Pharr, M., Kolb, C., Gershbein, R., & Hanrahan, P. (1997). Rendering Complex Scenes with Memory-Coherent Ray Tracing. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 101-108). Reading, MA: Addison-Wesley.

Phong, B.-T. (1975). Illumination for Computer Generated Images. *Communications of the ACM,* 18(6), 311-317.

Pineda, J. (1988). A Parallel Algorithm for Polygon Rasterization. *Computer Graphics, 22*(4), 17-20. (SIGGRAPH '88)

Pitteway, M. L. V. (1967). Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotter. *Computer Journal, 10*(3), 282-289.

Pixar. (2000). *The RenderMan Interface Specification.* Emeryville, CA.

Plauger, P. J. (1991). *The Standard C Library.* Englewood Cliffs, NJ: Prentice Hall.

Porter, T., & Duff, T. (1984). Compositing Digital Images. In *SIGGRAPH '84: Proceedings of the 11 th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 253-259). New York: ACM Press.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing* (Second ed.). Cambridge, UK: Cambridge University Press.

Prosise, J. (1999). *Programming Windows with MFC* (Second ed.). Microsoft Press.

Proudfoot, K., Mark, W. R., Tzvetkov, S., & Hanrahan, P. (2001). A Real-Time Procedural Shading System for Programmable Graphics Hardware. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 159-170). New York: ACM Press.

Purcell, T. J., Buck, I., Mark, W. R., & Hanrahan, P. (2002). Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics, 21*(3), 703-712. (SIGGRAPH '02)

Rahman, Z., Jobson, D. J., & Woodell, G. A. (1996). A Multiscale Retinex for Color Rendition and Dynamic Range Compression. In *SPIE Proceedings: Applications of Digital Image Processing XIX* (Vol. 2847). Bellingham, WA: SPIE.

Rea, M. S. (Ed.). (1993). *The Illumination Engineering Society Lighting Handbook* (8th ed.). New York: Illumination Engineering Society.

Reeves, W. T. (1983). Particle Systems—A Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics, 2*(2), 91-108.

Reinhard, E. (2003). Parameter Estimation for Photographic Tone Reproduction. *journal of graphics tools, 7*(1), 45-51.

Reinhard, E., Ashikhmin, M., Gooch, B., & Shirley, P. (2001). Color Transfer Between Images. *IEEE Computer Graphics and Applications, 21,* 34-41.

Reinhard, E., & Devlin, K. (2005). Dynamic Range Reduction Inspired by Photoreceptor Physiology. *IEEE Transactions on Visualization and Computer Graphics, 11* (1), 13-24.

Reinhard, E., Stark, M., Shirley, P., & Ferwerda, J. (2002). Photographic Tone Reproduction for Digital Images. *ACM Transactions on Graphics, 21*(3), 267-276. (SIGGRAPH '02)

Reinhard, E., Ward, G., Debevec, P., & Pattanaik, S. (2005). *High Dynamic Range Imaging.* San Francisco: Morgan Kaufmann.

Reynolds, C. W. (1987). Flocks, Herds and Schools: A Distributed Behavioral Model. *Computer Graphics, 21*(4), 25-34. (SIGGRAPH '87)

Riesenfeld, R. F. (1981, January). Homogeneous Coordinates and Projective Planes in Computer Graphics. *IEEE Computer Graphics & Applications, 1* (1), 50-55.

Roberts, L. (1965, May). *Homogenous Matrix Representation and Manipulation of N-Dimensional Constructs* (Tech. Rep. No. MS-1505). Lexington, MA: MIT Lincoln Laboratory.

Rogers, D. F. (1985). *Procedural Elements for Computer Graphics.* New York: McGraw Hill.

Rogers, D. F. (1989). *Mathematical Elements for Computer Graphics.* New York: McGraw Hill.

Rogers, D. F. (2000). *An Introduction to NURBS: With Historical Perspective.* San Francisco, CA: Morgan Kaufmann.

Rogers, S. (1995). Perceiving Pictorial Space. In W. Epstein & S. Rogers (Eds.), *Perception of Space and Motion* (Vol. 5, pp. 119-163). San Diego: Acad- emic Press.

Rost, R. J. (2004). *OpenGL Shading Language.* Reading, MA: Addison Wesley.

Roth, S. (1982). Ray Casting for Modelling Solids. *Computer Graphics and Image Processing, 18*(2), 109-144.

Rubin, S. M., & Whitted, T. (1980). A 3-Dimensional Representation for Fast Rendering of Complex Scenes. *Computer Graphics, 14*(3), 110-116. (SIGGRAPH '80)

Ruderman, D. L., Cronin, T. W., & Chiao, C. (1998). Statistics of Cone Responses to Natural Images: Implications for Visual Coding. *J. Opt. Soc. Am. A, 15*(8), 2036-2045.

Rusinkiewicz, S., & Levoy, M. (2000). QSplat: A Multiresolution Point Rendering System for Large Meshes. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 343-352). Reading, MA: Addison-Wesley.

Sabella, P. (1988). A Rendering Algorithm for Visualizing 3D Scalar Fields. *Computer Graphics,* 22(4), 51-58. (SIGGRAPH '88)

Saito, T., & Takahashi, T. (1990). Comprehensible Rendering of 3-D Shapes. *Computer Graphics, 24*(4), 197-206. (SIGGRAPH '90)

Salomon, D. (1999). *Computer Graphics and Geometric Modeling.* New York: Springer-Verlag.

Sbert, M. (1997). *The Use of Global Random Directions to Compute Radiosity. Global Monte Carlo Techniques.* PhD. thesis, Universitat Politènica de Catalunya.

Schlick, C. (1994a). An Inexpensive BRDF Model for Physically-Based Rendering. *Computer Graphics Forum, 13*(3), 233-246.

Schlick, C. (1994b). Quantization Techniques for the Visualization of High Dynamic Range Pictures. In P. Shirley, G. Sakas, & S. Müller (Eds.), *Photorealistic Rendering Techniques* (pp. 7-20). Berlin: Springer-Verlag.

Schwarze, J. (1990). Cubic and Quartic Roots. In (pp. 404-407). San Diego, CA: Academic Press Professional, Inc.

Sederberg, T. W., & Parry, S. R. (1986). Free-Form Deformation of Solid Geometric Models. *Computer Graphics, 20*(4), 151-160. (SIGGRAPH '86)

Seetzen, H., Heidrich, W., Stuerzlinger, W., Ward, G., Whitehead, L., Trentacoste, M., et al. (2004). High Dynamic Range Display Systems. *ACM Transactions on Graphics, 23*(3), 760-768. (SIGGRAPH '04)

Seetzen, H., Whitehead, L. A., & Ward, G. (2003). A High Dynamic Range Display Using Low and High Resolution Modulators. In *The Society for Information Display International Symposium.* San Jose, CA: Society for Information Display.

Segal, M., Korobkin, C., Widenfelt, R. van, Foran, J., & Haeberli, P. (1992). Fast Shadows and Lighting Effects Using Texture Mapping. *Computer Graphics, 26*(2), 249-252. (SIGGRAPH '92)

Shannon, C. E., & Weaver, W. (1964). *The Mathematical Theory of Communication.* Urbana, IL: University of Illinois Press.

Shene, C.-K. (2003). *CS 3621 Introduction to Computing with Geometry Notes.* Available from World Wide Web. (http://www.cs.mtu.edu/shene/COURSES/cs3621/NOTES/notes .html)

Shirley, P. (1991). *Physically Based Lighting Calculations for Computer Graphics.* Unpublished doctoral dissertation, University of Illinois, Urbana- Champaign.

Shirley, P., Smits, B., Hu, H., & Lafortune, E. (1997). A Practitioners' Assessment of Light Reflection Models. In *PG '97: Proceedings of the 5th Pacific Conference on Computer Graphics and Applications* (pp. 40-49). Los Alamitos, CA: IEEE Computer Society.

Shirley, P., Wang, C., & Zimmerman, K. (1996). Monte Carlo Techniques for Direct Lighting Calculations. *ACM Transactions on Graphics, 15*(1), 1-36.

Shreiner, D. (Ed.). (2004). *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.4* (Fourth ed.). Reading, MA: Addison-Wesley.

Shreiner, D., Neider, J., Woo, M., & Davis, T. (2004). *OpenGL Programming Guide* (Fourth ed.). Reading, MA: Addison-Wesley.

Sillion, F. X., & Puech, C. (1994). *Radiosity and Global Illumination.* San Francisco, California: Morgan Kaufmann Publishers, Inc.

Smits, B. E., Shirley, P., & Stark, M. M. (2000). Direct Ray Tracing of Displacement Mapped Triangles. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (pp. 307-318). London, UK: Springer-Verlag.

Snyder, J. M., & Barr, A. H. (1987). Ray Tracing Complex Models Containing Surface Tessellations. *Computer Graphics, 21*(4), 119-128. (SIGGRAPH '87)

Sobel, I., Stone, J., & Messer, R. (1975). *The Monte Carlo Method.* Chicago, IL: University of Chicago Press.

Solomon, H. (1978). *Geometric Probability.* Philadelphia, PA: SIAM Press.

Stam, J. (1999). Diffraction Shaders. In *SIGGRAPH '99: Proceedings of the 26th Annual Conference On Computer Graphics And Interactive Techniques* (pp.101-110). Reading, MA: Addison-Wesley.

Stark, M. M., Arvo, J., & Smits, B. (2005). Barycentric Parameterizations for Isotropic BRDFs. *IEEE Transactions on Visualization and Computer Graphics, 11*(2), 126-138.

Stockham, T. (1972). Image Processing in the Context of a Visual Model. *Proceedings of the IEEE, 60*(7), 828-842.

Strang, G. (1988). *Linear Algebra and Its Applications* (Third ed.). Florence, KY: Brooks Cole.

Sutherland, I. E., Sproull, R. F., & Schumacker, R. A. (1974). A Characterization of Ten Hidden-Surface Algorithms. *ACM Computing Surveys, 6*(1), 1-55.

Thompson, W. B., & Pong, T. C. (1990). Detecting Moving Objects. *International Journal of Computer Vision, 4*(1), 39-57.

Thompson, W. B., Shirley, P., & Ferwerda, J. (2002). A Spatial Post-Processing Algorithm for Images of Night Scenes. *journal of graphics tools, 7*(1), 1-12.

Tomasi, C., & Manduchi, R. (1998). Bilateral Filtering for Gray and Color Images. In *Proc. IEEE International Conference on Computer Vision* (pp. 836-846). Washington, DC: IEEE.

Tumblin, J., & Turk, G. (1999). LCIS: A boundary Hierarchy for DetailPreserving Contrast Reduction. In A. Rockwood (Ed.), *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 83-90). Reading, MA: Addison Wesley Longman.

Turk, G., & Levoy, M. (1994). Zippered Polygon Meshes from Range Images. In *Proceedings of SIGGRAPH 94* (pp. 311-318). New York: ACM Press.

Turkowski, K. (1990). Properties of Surface-Normal Transformations. In *Graphics Gems* (pp. 539-547). Boston: Academic Press.

Upson, C., & Keeler, M. (1988). V-Buffer: Visible Volume Rendering. *Computer Graphics, 22*(4), 59-64. (SIGGRAPH '88)

Upstill, S. (1985). *The Realistic Presentation of Synthetic Images: Image Processing in Computer Graphics.* Unpublished doctoral dissertation, University of California at Berkeley.

Van Aken, J., & Novak, M. (1985). Curve-Drawing Algorithms for Raster Displays. *ACM Transactions on Graphics, 4*(2), 147-169.

Veach, E., & Guibas, L. J. (1997). Metropolis Light Transport. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 65-76). Reading, MA: Addison-Wesley.

Wald, I., Slusallek, P., Benthin, C., & Wagner, M. (2001). Interactive Distributed Ray Tracing of Highly Complex Models. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (pp. 277-288). London, UK: Springer-Verlag.

Walter, B., Hubbard, P. M., Shirley, P., & Greenberg, D. F. (1997). Global Illumination Using Local Linear Density Estimation. *ACM Transactions on Graphics, 16*(3), 217-259.

Wandell, B. A. (1995). *Foundations of Vision.* Sunderland, MA: Sinauer Associates.

Wann, J. P., Rushton, S., & Mon-Williams, M. (1995). Natural Problems for Stereoscopic Depth Perception in Virtual Environments. *Vision Research, 35*(19), 2731-2736.

Ward, G., & Simmons, M. (2004). Subband Encoding of High Dynamic Range Imagery. In *First ACM Symposium on Applied Perception in Graphics and Visualization (APGV)* (pp. 83-90). NY: ACM Press.

Ward, G. J. (1992). Measuring and Modeling Anisotropic Reflection. *Computer Graphics, 26*(2), 265-272. (SIGGRAPH '92)

Ward, G. J. (1994). The RADIANCE Lighting Simulation and Rendering System. In A. Glassner (Ed.), *SIGGRAPH '94: Proceedings of the 21stAnnual Conference on Computer Graphics and Interactive Techniques* (pp. 459-472). New York: ACM Press.

Ward Larson, G., Rushmeier, H., & Piatko, C. (1997). A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes. *IEEE Transactions on Visualization and Computer Graphics, 3*(4), 291-306.

Ward Larson, G., & Shakespeare, R. A. (1998). *Rendering with Radiance.* San Francisco, CA: Morgan Kaufmann Publishers.

Warn, D. R. (1983). Lighting Controls for Synthetic Images. *Computer Graphics, 17*(3), 13-21. (SIGGRAPH '83)

Watt, A. (1991). *Advanced Animation and Rendering Techniques.* Reading, MA: Addison-Wesley.

Watt, A. (1993). *3D Computer Graphics.* Reading, MA: Addison-Wesley.

Wei, L.-Y., & Levoy, M. (2000). Fast Texture Synthesis Using Tree-Structured Vector Quantization. In *Proceedings of SIGGRAPH 2000* (pp. 479-488). Reading, MA: Addison-Wesley.

Whitted, T. (1980). An Improved Illumination Model for Shaded Display. *Communications of the ACM, 23*(6), 343-349.

Williams, A., Barrus, S., Morley, R. K., & Shirley, P. (2005). An Efficient and Robust Ray-Box Intersection Algorithm. *journal of graphics tools, 10*(1) 49-54.

Williams, L. (1978). Casting Curved Shadows on Curved Surfaces. *Computer Graphics, 12*(3), 270-274. (SIGGRAPH)

Williams, L. (1983). Pyramidal Parametrics. *Computer Graphics, 17*(3), 1-11. (SIGGRAPH '83)

Williams, L. (1991). Shading in Two Dimensions. In *Proceedings of Graphics Interface* (pp. 143-151). Wellesley, MA: A K Peters & Canadian Human-Computer Communications Society.

Wyszecki, G., & Stiles, W. (1992). *Color Science: Concepts and Methods, Quantitative Data and Formulae* (Second ed.). New York: Wiley.

Wyvill, B., McPheeters, C., & Wyvill, G. (1986). Data Structure for Soft Objects *The Visual Computer, 2*(4), 227-234.

Yantis, S. (Ed.). (2000). *Visual Perception: Essential Readings.* London, UK: Taylor & Francis Group.

Yessios, C. I. (1979). Computer Drafting of Stones, Wood, Plant and Ground Materials. *Computer Graphics, 13*(2), 190-198. (SIGGRAPH '79)

Yonas, A., Goldsmith, L. T., & Hallstrom, J. L. (1978). The Development of Sensitivity to Information from Cast Shadows in Pictures. *Perception, 7,* 333-342.

索　引

索引中的页码为英文原书的页码，与书中边栏的页码一致。

A

A/D converter（模数转换器），73
adjoint matrix（伴随矩阵），127
aerial perspective（空间透视），509
aliasing（走样、走样），67, 72, 74, 75, 98, 99, 112-114, 117
ambient shading（环境明暗处理），193
amodal completion（非模态完成），508
analog-to-digital converter（模数转换器），73
angle（角度），20
  cosine（余弦），21
  sine（正弦），21
animation（动画），2
anti-umbra（非阴影区），231
antialiasing（反走样），67, 98, 99
aperture problem（孔径问题），493
API（应用程序接口），1,3
apparent motion（视动），493
application program interface（应用程序接口），3
arc length（弧长），305
arc-length parameterized curve（弧长参数化曲线），37
array（队列）
  padded（填补），274
  tiling（平铺），274
artistic shading（艺术化明暗处理），197
assert(), 11
associativity（结合律），80, 83
attribute variables（属性变量），391
average（平均值），281

B

B-spline（B样条）
  control points（控制点），335
  curve（曲线），334, 335
  filter（滤波器），90,99
  function（函数）
    Fourier transform（傅里叶变换），110
  interpolation（插值），342
  non-uniform（非均匀），340
  NURBS（非均匀有理B样条），344
  repeated knots（重复的节点），342
  uniform cubic（均匀三次B样条），339
  uniform quadratic（均匀二次B样条），337
Bézier curves（贝塞尔曲线），327
backface elimination（背向面删除），265
barycentric coordinates（重心坐标），43, 44, 63, 300
basic execution model（基本执行模型），390
basis（基），24
  function（函数），310
  matrix（矩阵），313
  vectors（向量），24
Bernstein basis polynomials（伯恩斯坦基本多项式），329
bidirectional reflectance distribution function（双向反射率分布函数），458, 561
  Lambertian（朗伯表面）459
bijection（双射），17
binary space partitioning (BSP) tree（二叉空间划分（BSP）树），177, 236
blending function（混合函数），310, 316
blurring（模糊），96
bounding box（包围盒），219
bounding volume（包围体），265
  hierarchy（层次），222
box filter（盒式滤波器），67, 78, 83, 89, 99, 100
box function（盒式函数），78, 83
  Fourier transform（傅里叶变换），109
BRDF（双向反射率分布函数），458, 561
  Lambertian（朗伯表面），459
bricking（分块），274
BSP tree（BSP树），177, 236
  ray tracing（光线跟踪），227
built-in variables（内建变量），391
bump mapping（凹凸映射），392

C

CAD/CAM（计算机辅助设计/计算机辅助制造），3
callbacks（回调），4
camera（摄像机）
  digital（数码摄像机），68, 466
candela（烛光（发光强度单位）），462, 489
canonical，160
canonical form（规范形式），310, 312
canonical view volume（标准视体），160
cardinal spline（基数样条），325
cardinality（基数），48

Cartesian coordinates（笛卡儿坐标），24
Cartesian product（笛卡儿积），16
Catmull-Rom filter（Catmull-Rom滤波器），90, 91
Catmull-Rom splines（Catmull-Rom 样条），325
CCD, 68
cellular automata（细胞自动机），371
chromatic adaptation（色调自适应），537
chromaticity（色度），469
CIE scotopic sensitivity function（CIE暗视灵敏度函数），472
circle（圆），20
clipping（裁剪），175, 259
  line segment（线段），264
  triangle（三角形），264
closed interval（闭区间），17
CMOS, 68
coefficients（系数），310
cofactor matrix（余子式矩阵），127
cofactors，125
color（颜色），465, 486
  24 bit（24位），55
  additive（相加的），55
  constancy（不变性），488
  hue（色调），475
  opponent（对立的），474
  RGB, 51, 54, 465
  saturation（饱和度），475
  space（空间）
    CIE, 487
    HSV（色调/饱和度/光亮度模型），488
    RGB, 472, 473, 487
  trichromatic（三原色的），466
  value（估计值），475
  state（状态），383
column vector（列向量），123
commutativity（交换律），80, 81, 83
compositing（合成），56
compression（压缩），68
  lossless（无损的），68
  lossy（有损的），68
  sigmoidal（S形的），537.
cones（视锥），486
conformational parameters（构象参数），365
constant（常数）
  semi-saturation（半饱和），535, 536
constructive solid geometry（构造实体几何），229
continuity（连续性），92, 308
continuous motion（连续运动），494
control driven programming（控制驱动的编程），405
control point（控制点），311, 335
convex hull（凸包），333
convolution（卷积），75, 76, 108, 112
  and Fourier transform（傅里叶变换），108
  by separable filter（可分滤波器），94, 95
  continuous（连续），82-84
  discrete（离散），76-81
  discrete-continuous（离散-连续），85, 86
  of an image（图像的），95
  properties（性质），79, 80, 83
  smoothing（平滑），76
  two-dimensional（二维的），87, 88
coordinate frame（坐标系），29
coordinates
  global（全局坐标），272
  local（局部坐标），272
  polar（极坐标），21
correspondence problem（对应问题），500
cosine（余弦），21
Cox-de Boor recurrence（Cox-de Boor循环），341
Cramer's rule（克莱姆法则），121,129, 207
cross product（叉积），26
crossed disparity（交叉视差），501
CRT（阴极射线管），52
CSG (constructive solid geometry)（构造实体几何） 229
cube map（立方体图），254
cubic polynomials（三次多项式），315
cues（线索）
  dynamic occlusion（动态遮挡），503
culling（剔除），265
  backface elimination（背向面删除），265
curve（曲线），30, 301
  arc-length parameterized（弧长参数化），37
  B-spline（B样条），334,335
  Bézier（贝塞尔），327
  implicit（隐式的），30
  parametric（参数），36, 40
  tangent（切），31
cycle (viewing)（循环（观察的）），178

D

D/A converter（数模转换器），73
data bus（数据总线），384
data sink（数据接收），391
data source（数据来源），391
de Casteljau algorithm（de Casteljau算法），333
debugging（调试），12
degree of continuity（连续程度），92
delta function（δ函数），84, 111
density representation（密度表示法），529
dependency scheme（依赖模式），319
depth buffer（深度缓冲器），186
  interger（整数），186
depth of field（景深），233
derivative（导数），31
determinant（行列式），119, 125
  Laplace's expansion（拉普拉斯展开式），125
diagonal matrix（对角矩阵），124
diagonalization（对角化），130
dielectric（电介质），213
difference equation（差分方程），368, 369

explicit schemes（显式模式），369
implicit schemes（隐式模式），369
differential geometry（微分几何），308
diffuse reflection（漫反射），459
digital audio（数字音频），72
digital camera（数码摄像机），68
digital-to-analog converter（数模转换器），73
Dirac delta function（狄拉克δ函数），84, 111
direct lighting（直接光照），554
direct volume rendering（直接体绘制），590
Direct3D, 4
directional hemispherical reflectance（半球面方向反射系数），459
directional light（方向性光照），192
discontinuous motion（非连续运动），493
discrete impulse（离散脉冲），80, 81
discriminant（判别式），19
disparity（视差），500
displacement map（置换映射），253
display（显示器）
CRT（阴极射线管），52
interlaced（隔行），495
LCD, 52
raster（光栅），51
resolution（分辨率），51
display list（显示列表），267, 385
distal stimulus（远侧刺激），478
distance（距离）
signed（有向的），35
distribution ray tracing（分布式光线跟踪），229
domain（定义域），16
dot product（点积），25
double（双精度），9
drop shadow（阴影），97

E

efficiency（效率），8
eigenvalues（特征值），129
eigenvectors（特征向量），129
environment maps（环境映照），254
event-driven programming（事件驱动的编程），408
expressive parameters（表情参数），365
expected value（期望值），285, 288
explicit curve（显式曲线），302
eye（眼睛）
cones（视锥），486
retina（视网膜），466
rods（视杆），486

F

familiar size（习惯大小），512
Fechner's law（Fechner定律），484
field radiance（场辐射率），456, 581
field-of-view（视域），173, 490
filter（滤波器）
B-spline（B样条），90, 99
bilateral（双向），530
box，67, 78, 83, 84, 89, 99, 100
Catmull-Rom（Catmull-Rom滤波器），90, 91
degree of continuity（连续程度），92
edge-preserving smoothing（保留边缘的平滑（算子）），531
Gaussian（高斯），90, 94, 99
identity（单位函数），80, 84
impulse response（脉冲响应），91
interpolating（插值），90, 91
Mitchell-Netravali（Mitchell-Netravali滤波器），91
overshoot（过冲），92
resampling（重采样），101, 102
ringing（振铃），92
ripple free（无波动的），92
separable（可分的），93, 103
separable tent（可分帐篷式），93
shifted（移位的），81
tent（帐篷式），83, 84, 89, 93, 99
trilateral（三向），530
filtering（滤波），78
flexors（屈肌），364
flicker fusion（闪烁融合），494
float（单精度），9
focus of expansion（膨胀中心），514
forward kinematics（正向运动学），362
Fourier series（傅里叶级数），105
Fourier transform（傅里叶变换），105-110
fovea（凝视陷），490
fragment program（片段程序），388
fragment shader（片段着色器），388
frame of reference（参考坐标系），29
allocentric（以环境为中心的），497
egocentric（以自我为中心的），497
exocentric（外中心的），497
frame rate（帧率），380, 495
frame update rate（帧更新速率），495
free-form curve（自由形式曲线），302
frequency-based operators（基于频率的算子），529
Fresnel equations（菲涅尔公式），214, 562
full multigrid method（完全多栅格法），531
function（函数），15
bijection（双射），17
domain（定义域），16
inverse（逆），17
inversion（逆），292
range（值域），16

G

gamma（伽马γ），53, 472
gamma correction（γ校正），54
Gaussian（高斯）
Fourier transform（傅里叶变换），110
filter（滤波器），90, 94, 99

gaze direction（凝视方向），164
geometric continuity（几何连续性），309
geometric surface，381
gif（可交换的图像文件格式），68
global coordinates（全局坐标），28,272
global illumination（全局照明），547
Gouraud color interpolation（Gouraud颜色插值），63
GPU（图形处理单元），389
  general purpose computation（通用计算），398
gradient（梯度），31
gradient domain（梯度域）
  compression（压缩），531
  operator（算子），531
graphics hardware（图形硬件），379
  processor（处理器），389
Graphics Processing Unit（图形处理单元），389
graphics state（图形状态），383
GUI（图形用户界面）
  API（应用程序接口），411
  elements（元素），403, 419
  events（事件），412
    callback（回调），419
    paint, 415
    redraw, 415
    RedrawRoutine(), 416
    service registration（服务注册），419, 434
    service routines（服务例程），416, 419, 420
    timer（定时器），413

H

haloing（光晕），534
hard shadow（硬阴影），231
Hermite cubic（埃尔米特三次），324
  polynomials（多项式），324
Hermite form（埃尔米特形式），315
hidden surface elimination（隐藏面消除），177
  BSP tree（BSP树），177
  ray tracing（光线跟踪），203
  z-buffer（z缓冲器），186
high dynamic range images（高动态范围图像），521
highlight（高光），194
histogram adjustment（直方图调整），542
histogram equalization（直方图均衡），542
homogeneous coordinates（齐次坐标），152, 169
homogenization（齐次化），169
horizon ratio（地平线比率），513
hue（色调），469, 475, 488
hypotenuse（直角三角形的）斜边, 20

I

IBR（基于图像的绘制），577
identity filter（单位滤波器）
  continuous（连续），84
  discrete（离散），80
identity matrix（单位矩阵），123
IEEE floating point（IEEE浮点），5
illuminance（照度），525
illumination（照明）
  global（全局），547
image（图像），9, 71
  blurring（模糊），96
  compression（压缩），68
  convolution（卷积），95
  drop shadow（阴影），97
  Gaussian-blurred（高斯模糊），539
  high dynamic range（高动态范围），521
  processing（处理），2
  reconstruction（重构），99-102
  resampling（重采样），99, 100, 102
  resmpling，101
  sharpening（锐化），97
  unsharp mask（模糊掩模），97
image-based modeling（基于图像的建模），580
image-based rendering（基于图像的绘制），577
immediate mode rendering（立即模式绘制），384
implicit curve（隐式曲线），30, 302
  gradient（梯度），31
implicit function（隐式函数），302
importance sampling（重要性采样），289
impulse（脉冲）
  discrete（离散），80, 81
  response（响应），91
  train（链），111
include guards（包含保护符），11
incremental computation（增量计算），59
independent random variable（独立的随机变量），286
indirect lighting（间接光照），547
information visualization（信息可视化），3, 583
infrared light（红外线），462
inline functions（内联函数），10
instancing（实例化），216
integral geometry（积分几何学），281
integration（积分），280
  Monte Carlo（蒙特卡罗），288, 552
  quasi-Monte Carlo（准蒙特卡罗），290
intensity（光强度），463
interpolate（插值），308
interpolating filter（插值滤波器），90,91
interpolation（插值），42
  B-spline（B样条），342
  linear（线性），42
  normal vector（法向量），197
intersection（相交）
  ray-bounding box（光线与包围盒），219
  ray-implicit（光线与隐式表面），205
  ray-parametric（光线与参数表面），206
  ray-sphere（光线与球），205
  ray-triangle（光线与三角形），206
interval（区间），17
  closed（闭），17

open（开），17
set operations（集合运算），17
inverse Fourier transform（傅里叶逆变换），106
inverse function（反函数），17
inverse kinematics（逆向运动学），363
inverse matrix（逆矩阵），124,128
inverse optics（逆向光学），496
inversion（逆）
function（函数），292
irradiance（辐射照度），454
isosurface（等值面），585

J

Jacobian（雅可比行矩阵），363
jaggies（锯齿），98
Java（Java语言），4
jittering（抖动），231
jpeg（一种压缩图像格式），69

K

kernel（内核），398
kinetic depth effect（运动深度效应），502
knot（节点），318
vector（向量），319

L

Lagrange form（拉格朗日形式），317
Lambert's consine law（朗伯余弦定律），192
Lambertian（朗伯），191
BRDF（双向反射率分布函数），459
Law of Large Numbers（大数定律），288
LCD（液晶显示器），52
level-of-detail (LOD)（细节等级), 5
light（光）
directional（方向性），192
infrared（红外线），462,
ultraviolet（紫外线），462
light field（光场），578
lighting（光照），265
direct（直接），554
indirect（间接），547
two-sided（双面的），193
lightness（光亮度），475,481, 487, 531
constancy（不变性），481
limit curve（极限曲线），331
line（直线）
drawing（画），57
incremental（增量式的），59
implicit（隐式的），33
normal coordinates（法向坐标），283
Parametric（参数），37, 40, 61
segment（线段），41
slope（斜率），33
slope-intercept form（斜截式），33
vector form（向量形式），40
linear independence（线性独立），24
linear interpolation（线性插值），42
linear perspective（线性透视），503
linear scaling（线性缩放），523, 534
linear system（线性方程组）
solution（解），129
Cramer's rule（克莱姆法则），121
local coordinates（局部坐标），29, 272
location，23
LOD (level-of-detail)（细节等级），5
logarithm（对数），18
natural（自然），18
luminaire（光源），547
non-diffuse（非漫反射），558
luminance（亮度），462, 524
scotopic（暗视的），472, 543
luminance distribution（亮度分布），485
luminance mapping function（亮度映射函数），542
luminous efficiency function（发光效率函数），462

M

mappings（映射），15
marching cubes（步进立方体），586
material（材质）
dielectric（电介质），213
matrix（矩阵），121
adjoint（伴随），127
cofactor（余子式），127
cofactors（余子式），125
determinant（行列式），119
diagonal（对角），124
diagonal form（对角形式），130
identity（单位矩阵），123
inverse（逆），124, 128
inversion using SVD（用SVD方法求逆矩阵），154
multiplication（乘法），122
product with scaler（矩阵与常数相乘），122
projection（投影），172
square（方阵），122
stack（栈），273
symmetric（对称矩阵），123
transpose（转置），123
maximum-intensity projection（最大强度投影），593
measure（度量）
2D lines（二维直线），282
3D lines（三维直线），283
zero measure sets（零度量集合），280
mesh（网格），248
metamer（条件等色），467, 468, 486, 487
Metropolis sampling（Metropolis采样），295
microfacets（微平面），562
midpoint algorithm（中点算法），57
MIP-mapping（MIP映射），256
Mitchell–Netravali filter（Mitchell–Netravali滤波器），91
modeling（建模），2, 4

modelview–controller（模型–视图–控制器），421, 422, 423
  component（组件），421
  framework（框架），422
  model（模型），422
  view（视图），422, 423
moiré pattern（莫尔图案），75, 98, 99, 103, 230
monitor（显示器）
  gamma（γ），53
  phosphors（荧光粉），472
  tristimulus values（三刺激值），473
  white point（白点），472
Monte Carlo integration（蒙特卡罗积分），288, 552
  importance sampling（重要性采样），289
  stratified sampling（分层采样），289
morph（变形），365
motion blur（运动模糊），234
motion capture（运动捕捉），72, 366
  electromagnetic（电磁式），366
  optical（光学式），366
motion parallax（运动视差），502
moving average（滑动平均），76
mutual illumination（相互照明），547

N

nanometer（纳米），452
natural cubic（自然三次），323
  splines（样条），323
natural logarithm（自然对数），18
non-photorealistic rendering（非真实感绘制），197
non-uniform B-spline（非均匀B样条），340
normal coordinates（正则坐标），283
normal direction（法向），383
normal interpolation（法线插值），197
normal vector（法向量），31, 149
  at vertices（顶点处的），193
number density（数目密度），590
NURBS（非均匀有理B样条），344
Nyquist frequency（尼奎斯特频率），113, 114
Nyquist–Shannon sampling theorem（尼奎斯特–香农采样定律），114

O

object recognition（物体识别），509
occlusion cues（遮挡线索），503
ocularmotor cue（眼球运动线索）
  accomodation（适应性调节），498
  convergence（会聚（度）），498
  vergence（聚散（度）），498
offset（偏移量），23
open interval（开区间），17
OpenGL, 4
operator（算子），10
  frequency-based（基于频率的），529
  gradient-domain（梯度域），531
  spatial（空间的），532
  tone reproduction（色调再现），542
opponent colors（对立颜色），474
optic flow（光流），501
origin（原点），23
orthogonal（正交），24
orthographic projection（正投影），162
orthographic view volume（正射视体），162
orthonormal（标准正交），24
  basis（基），27
overshoot（过冲），92

P

padded array（填补数组），274
painter's algorithm（画家算法），178
parallelepiped（平行六面体）
  volume（体积），119
parallelogram（平行四边形）
  area（面积），119, 125
parallelogram rule（平行四边形法则），23
parameterization（参数化），304
  natural（自然的），304
parametric（参数）
  continuity（连续性），310
  curve（曲线），36, 40, 303
  function（函数），302
  line（直线），40
  surface（曲面），41
parametrization（参数化）
  arc-length（弧长），305
particle tracing（粒子跟踪），549
path tracing（路径跟踪），551
penumbra（半影），231
persistent application state（持久稳固的应用程序状态），422
perspective（透视）
  projection（投影），166
  three point（三点），166
Phong exponent（Phong指数），196
Phong normal interpolation（Phong法线插值），197
Phong shading（Phong明暗处理），194
photometry（光度测定），451, 462
photon（光子），452
photoreceptors（光感受器），486, 524
physics-based animation（基于物理的动画），367
picking（拾取），201, 236
pictorial depth cues（绘画深度线索），503
pixel（像素），51
plane, implicit（隐式平面），38
point（点），23
polar coordinates（极坐标），21
polygon（多边形）
  ray intersection（光线相交），208
polynomial（多项式）
  Hermite cubic（埃尔米特三次多项式），324
position（位置），23
power（效率），453

ppm（一种图像格式），69
primitives（图元），380, 381
probability（概率），284
  density function（密度函数），284
  expected value（期望值），285
procedural curve（过程式曲线），302
programming models（编程模型）
  main(), 404
  control-driven programming（控制驱动的编程），404
  event-driven programming（事件驱动的编程），404
  external control model（外部控制模型），404
  internal control model（内部控制模型），404
  MainEventLoop(), 408
projection（投影）
  orthographic（正射），164
projection matrix（投影矩阵），172
property（性质）
  global（全局），308
  local（局部），308
proximal stimulus（近侧刺激），478
pseudocolor plot（伪彩色图），584
psychophysics（心理物理学），485
Pythagorean theorem（毕达哥拉斯定律），20
quadratic equation（二次方程），19, 205
quadric curve（二次曲线），36
quasi-Monte Carlo integration（准蒙特卡罗积分），290

R

radiance（辐射率），455, 578,
  field（场），456, 581
  surface（表面），456, 581
radiant exitance（辐射脱离），455
radiometry（辐射测量），451
radiosity（辐射度），463, 548
random point density plot，584
random sampling（随机采样），230
random variable（随机变量），286
  independent identically distributed（独立同分布的），288
range（值域），16
raster display（光栅显示器），51
rasterization（光栅化）
  triangle（三角形），63
ray（光线），41
  shadow（阴影），211
ray tracing（光线跟踪），201, 209
  distribution（分布式），229
  instancing（实例化），216
  object-oriented（面向对象的），210
re-parameterization（重新参数化），304
real-time graphics（实时图形学），380
reconstruction（重构），71, 85, 86, 99-102, 112-114, 117
  artifact（干扰），73, 74
reflect transform（反射变换），140
reflectance（反射系数），525
  directional hemispherical（半球面方向），459
reflection（反射）
  Lambertian（朗伯反射），191
  specular（镜面反射），212
  total internal（完全内部反射），214
refraction（折射），216
refresh rate（刷新速率），495
regular sampling（规则采样），230
rejection method（拒绝方法），294
relative size（相对大小），512
rendering（绘制），2
  non-photorealistic（非真实感），197
rendering equation（绘制方程），461
renormalization（重新规范化），103
resampling（重采样），99-102, 104, 117
  separable（可分的），104
rasampling filter（重采样滤波器），101, 102
resolution（分辨率），51
retina（视网膜），466
retroreflection（后向反射），575
RGB color（RGB颜色），9, 51, 54, 465
right-handed coordinates（右手坐标系），27
rigid body transforms（刚体变换），153
rigid skinning（刚性蒙皮），364
ringing（振铃），92
ripple free（无波动的），92
rods（杆状体），486
rotation transform（旋转变换），138
row vector（行向量），123

S

saccades（扫视），490
sampling（采样），71
  aliasing（走样），72, 74, 75
  jittering（抖动），231
  random（随机），230
  regular（规则），230
  sample rate（采样频率），73, 75
  stratified（分层），231
  undersampling（欠采样），73, 74
sampling theory（采样定律），104
saturation（饱和度），475, 487
scalar product（标量积），25
scale selection mechanism（尺度选择机制），540
scale transform（缩放变换），136
scaling（缩放）
  linear（线性），534
scene graph data structure（场景图数据结构），383
Schlick approximation（Schlick近似），214
scotopic luminance（暗视亮度），543
semi-saturation constant（半饱和常量），535, 536
separable filter（可分滤波器），93, 103, 104
separable tent filter（可分帐篷式滤波器），93
set（集合），16
  Cartesian product（笛卡儿积），16
  operations（运算）

on intervals（在区间上），17
shading（明暗处理），191, 265
ambient（环境），193
artistic（艺术化），197
diffuse（漫反射），191
Lambertian（朗伯），191
Phong, 194
shadow（阴影）
anti-umbra（非阴影区），231
attached（附着的），505
detached（分离的），505
hard（硬的），231
penumbra（半影），231
soft（软的），231
umbra（本影），231
shadow map（阴影图），255
shadow ray（阴影光线），211
shape constancy（形状不变性），514
shared-point scheme（共享点模式），319
sharpening（锐化），97
shear transform（切变变换），137
shifted filter（移位滤波器），81
sigmoid（S形曲线），534, 535
sigmoidal compression（S形压缩），537
signal processing（信号处理），71
signed distance（有向距离），35
silhouettes（轮廓线），197
sine（正弦），21
single instruction, multiple data (SIMD) processor(单指令多数据(SIMD)处理器），389
singular value decomposition (SVD)(奇异值分解），131,144
for matrix inversion（逆矩阵），154
singular values（奇异值），131
site（位置），308
size constancy（大小不变性），514
slant（倾斜度），497, 505
slope（斜率），31
smooth skinning（平滑蒙皮），364
smoothing（平滑），76
soft shadow（软阴影），231
software engineering（软件工程），8
solid texture（实体纹理）
turbulence（扰动），245
spatial operator（空间算子），532
spectral distribution（谱分布），485
spectral energy（光谱能量），453
spectral power（光谱功率），454
spectrum（光谱），485
specular highlights（镜面高光），525
specular reflection（镜面反射），212
sphere（球体）
implicit form（隐式形式），205
normal vector（法向量），206
vector form（向量形式），205
splatting（泼溅），594
spline（样条），307
curve（曲线），307
spot light（点光源），255
square matrix（方阵），122
state attributes（状态属性），382
steady state（稳定状态），453
step function（阶跃函数），78, 92
Steven's law（Steven定律），484
stratified sampling（分层采样），231, 289
stream processor（流处理器），389
structure-from-motion（运动结构），502
subdivision scheme（细分模式），331
subjective contour（主观轮廓），483
support（支撑），77
surface radiance（表面辐射率），456, 581
surface（曲面）
implicit（隐式），38
parametric（参数），41
SVD（奇异值分解），131, 144
symmetric matrix（对称矩阵），123

T

tangent（切），31
vector（向量），31
temporal aliasing（时域走样），495
tent filter（帐篷式滤波器），83, 89, 93, 99
tent function（帐篷函数），83, 84
Fourier transform（傅里叶变换），110
texture（纹理）
marble（大理石），245
texture mapping（纹理映射），239
MIP-mapping（MIP映射），256
three-point perspective（三点透视），166
tiff（标签图像文件格式），69
tiling（平铺），274
tilt（坡度），497, 505
TIN（不规则三角形网格），248
tone reproduction（色调再现），522
tone reproduction operator（色调再现算子），542
tonemapping（色调映射），522
total internal reflection（完全内部反射），214
transfer function（传输函数），590
transformation（变换）
matrix（矩阵），135
reflection（反射），140
rigid body（刚体），153
rotate（旋转），138
scale（缩放），136
shear（剪切），137
transport equation（输运方程），551, 554
transpose（转置），123
triangle（三角形），43
barycentric coordinates（重心坐标），43
fans（扇区），266
mesh（网格），248, 381

winged-edge data structure（翼边数据结构），270
random point selection（随机点选择），300
ray intersection（光线相交），206
rasterization（光栅化），63
strips（条带），266
triangular irregular network (TIN)(不规则三角形网格)，248
trichromatic color theory（三原色颜色理论），466
tristimulus values（三刺激值），469
turbulence function（扰动函数），245
two-sided lighting（双面光照），193

U

ultraviolet light（紫外线），462
umbra（本影），231
uncrossed disparity（不交叉的视差），501
undersampling（欠采样），73, 74, 113
uniform cubic B-spline（均匀三次B样条），339
uniform quadratic B-spline（均匀二次B样条），337
uniform variables（一致性变量），391
unit vector（单位向量），23
units（单位）
extensive（扩展），453
intensive（强度），453
unsharp mask（模糊掩模），97
user-defined variables（用户定义变量），391

V

value（估计值），475
variable（变量）
attribute（性质），391
built-in（内建），391
uniform（一致性），391
user-defined（用户定义），391
variance（方差），287
variation diminishing property（变差缩减性），330
varying data（可变数据），391
vector（向量），23
addition（加法），23
basis（基），24
cross product（叉积），26
dot product（点积），25
equality（相等），23
length（长度），23, 24
linear independence（线性独立），24
normal（法向量），31, 38, 149
orthogonal（正交），24
orthonormal（标准正交），24
projection（投影），25
subtraction（减法），24
unit（单位），23
zero（零），23
vertex（顶点）
buffer objects（缓冲对象），385
function calls（函数调用），382
normal（法线），193
program（程序），388
shader（着色器），388
video games（视频游戏），3
view volume（视体）
orthographic（正射），162
view-up vector（向上向量），164
viewing（观察），164
field-of-view（视域），173
virtual reality（虚拟现实），2
vision（视觉）
foveal（凝视），491
high acuity（高灵敏度），489
photopic（适应光的），489
scotopic（暗视的），489
visual acuity（视觉灵敏度），480
visual attention（视觉注意力），516
visual cue（视觉线索），496, 498
disparity（视差），496
motion（运动），496
ocularmotor（眼球运动），496
pictorial（绘画的），496
visualization（可视化），2, 583
information（信息），583
volume rendering（体绘制），585, 590
maximum-intensity projection（最大强度投影），593
transfer function（传输函数），590

W

wagon wheel illusion（货车车轮幻觉），495
Weber's law（Weber定律），484
winged-edge data structure（翼边数据结构），270

Z

z-buffer（z缓冲器），186
integer（整数），186
zero measure sets（零度量集合），280
zero vector（零向量），23